Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Corso di Laurea Specialistica in Informatica

Gender prediction tramite Artificial Neural Network

Studente:

Alessio Mancinelli

Professoressa:

Prof. Valentina Poggioni

Anno Accademico 2019–2020

Dichiarazione anti plagio

I certify that project "Gender prediction tramite Artificial Neural Network" is my own work, based on my personal study and research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, open source codes, web sites and any other kind of document, electronic or personal communication. I also certify that it has not previously been submitted for assessment in any other exam, as previously agreed with the teacher. Moreover I declare that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons.

Introduzione

Il progetto si pone l'obiettivo di proporre dei modelli di artificial neural network che siano in grado di mettere in relazione la sintassi ed il modo di scrittura delle persone con il loro sesso. Tutte le ANN saranno addestrate con un dataset ottenuto dal sito Evalita, contenente circa 11000 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

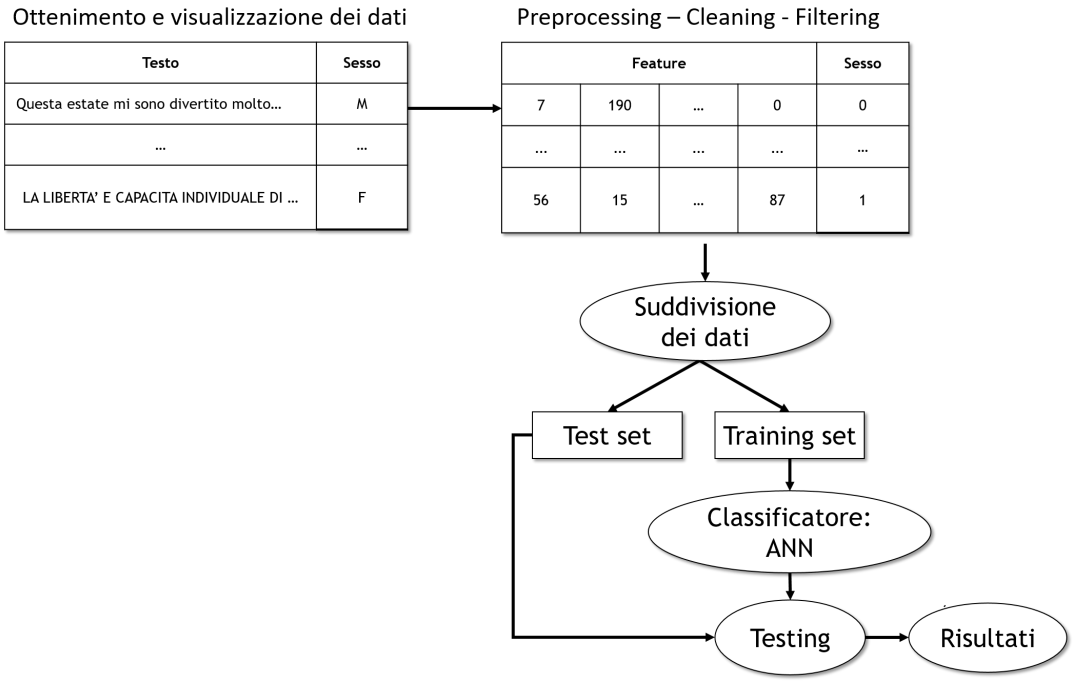

Figura 2.1: Workflow della soluzione

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.

1. Ottenimento del dataset dal sito di Evalita 2018.
2. Preprocessing - Cleaning - Filtering: Trasformazione dei dati e ottenimento delle Feature.
3. Suddivisione dei dati al fine di ottenere il training e test set.
4. Classificatore: Prototipazione delle ANN.
5. Fase di testing: Applicazione dei modelli.
6. Risultati: Ottenimento e visualizzazione dei risultati.

2.1 Dataset e preprocessing dei dati

I dati presi in considerazione sono stati distribuiti in formato XML e provengono da cinque differenti fonti:

- Commenti di Twitter
- Commenti di Youtube
- Diari personali
- Testi scritti da bambini
- Notizie/Giornalismo

In figura 2.2 vengono mostrati degli esempi di testo all'interno del dataset; il primo proviene da un diario personale, ed è stato scritto da un uomo (gender='M'), il secondo invece, proveniva da un commento di Twitter, ed

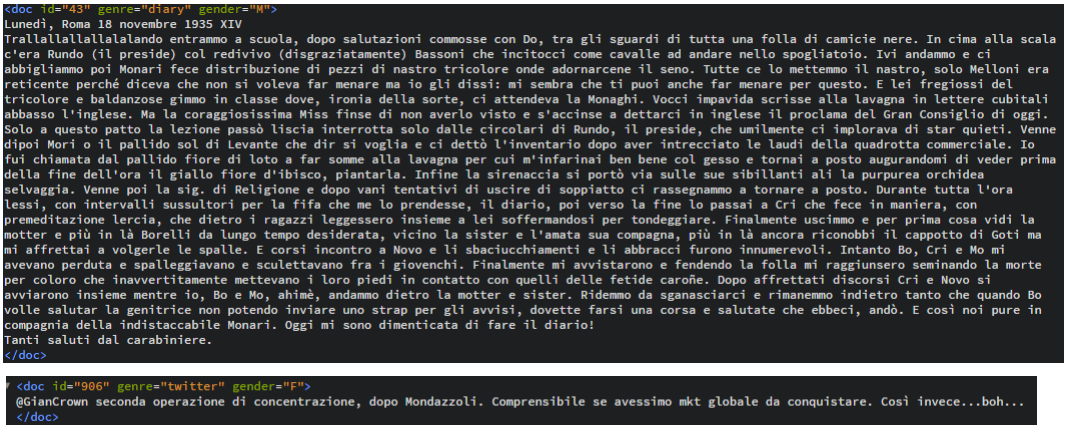

Figura 2.2: Esempi di frasi nel Dataset

è stato scritto da una donna (gender=’F’). Per ognuno dei cinque generi poi, è presente un differente numero di righe rappresentative del dataset in questione, visualizzabili in figura 2.3. Per aumentare i dati su cui addestrare i modelli e migliorare le capacità di generalizzazione, ho scelto di unire di tutti i dataset proposti, ottenendo un quantitativo di righe pari a 11000. Alcune righe però, risultavano duplicate, quindi per eliminare completamente la possibilità che i modelli possano prevedere record già visti durante la fase di addestramento, ho scelto di eliminare tutte le righe duplicate, facendo scendere il numero esatto di record utilizzati a 10837.

Dataset	F	M	Tokens
Children	100	100	65986
Diaries	100	100	82989
Journalism	100	100	113437
Twitter	3000	3000	101534
Youtube	2200	2200	90639

Figura 2.3: Numero di righe per ogni dataset

2.2 Apertura del Dataset

```
xtree = et.parse("Dataset_def.xml")
xroot = xtree.getroot()

df_cols = ["testo", "sesso"]
righe = []

for node in xroot:
    id = node.attrib.get("id")
    genere = node.attrib.get("genre")
    sesso = node.attrib.get("gender")
    testo = node.text
    righe.append({"testo": testo, "sesso": sesso})

df = pd.DataFrame(righe, columns = df_cols)
df['testo'].drop_duplicates(keep=False, inplace = True)
```

Figura 2.4: Apertura del dataset in Python ed eliminazione delle righe duplicate

Non essendo presente una funzione della libreria Pandas per poter inserire direttamente il contenuto di un file xml all'interno di un dataframe (come avviene per esempio con i formati CSV o JSON), ho effettuato il parsing tramite la libreria *xml.etree.ElementTree* andando ad identificare la radice del documento e iterando tutto il suo contenuto per ottenere i dati necessari all'addestramento. Successivamente ho inserito il testo ed il sesso (che rappresenterà la label da prevedere) all'interno di un array chiamato "righe" che sarà inserito poi in un dataframe per facilitare la suddivisione e gestione dei dati. Non si sono presi in considerazione gli attributi 'id' e 'genre' perchè non si sono ritenuti utili ai fini della previsione del sesso. Vengono infine mostrate le prime cinque righe del dataframe appena creato:

	testo	sesso
0	\n… Questa estate mi sono divertito molto perc...	M
1	\nBenvenuti a "TGDVD" i marò sono stati libera...	M
2	\nRoma, 9 Giungo 2012\nCaro diario,\nOggi vorr...	M
3	\nRoma 9 Giugno 2012\nCaro diario,\nil mio ult...	M
4	\nIo non so bene come definirmi, ma secondo me...	M

2.3 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 tutti i simboli ed i link ad altre pagine, principalmente presenti nei commenti su Twitter. Particolare attenzione è stata posta alle emoji (presenti in alcuni commenti sotto i video di Youtube) che possono portare informazione utile per la previsione del sesso. Inoltre sono stati eliminati tutti i riferimenti ad altri utenti all'interno dei tweets (ad esempio @user) e mantenuti gli hashtag, ai quali è stato rimosso solamente il simbolo # iniziale. 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.

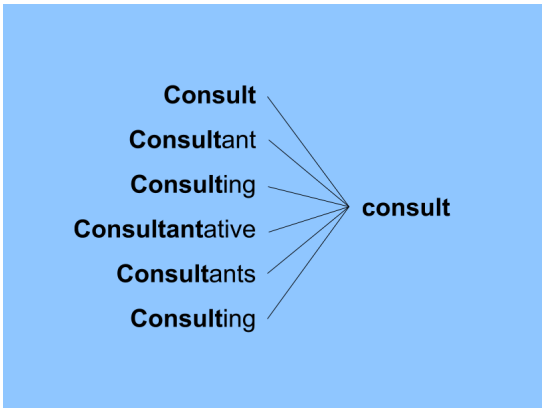

Figura 2.5: Esempio di Stemming

In questo specifico caso però, ho scelto di non utilizzare questa tecnica perchè l'obiettivo è quello di classificare il sesso di una persona, quindi ad esempio, dire "sono *stato*" o "sono *stata*" ha un significato nettamente diverso e di fondamentale importanza per la previsione del sesso. Vengono quindi mostrate in figura 2.6 alcune applicazioni della funzione implementata in python per la pulizia del testo.

```
Testo originale:

Ho caricato una nuova traccia, "3) TI SENTO DENTRO ME - ANGELA DONATO - mP3", su #spreaker #angela
#dolcezza #mia https://www. spreaker.com/user/angela3/3-ti-sento-dentro-me-angela-donato-mp3?autoplay=1 …

Testo filtrato:

caricato nuova traccia sento dentro me angela donato spreaker angela dolcezza


________
Testo originale:

Tante chiacchiere sull'innovazione. E dimentichiamo ciò che serve davvero: http://www. agendadigitale.eu/
startup/tante- chiacchiere-sull-innovazione-e-dimentichiamo-cio-che-serve-
davvero_1758.htm#.Vwj1WWufudw.twitter … #startup #innovazione

Testo filtrato:

tante chiacchiere innovazione dimentichiamo ciò serve davvero startup innovazione
________
Testo originale:

UN VERSETTO AL GIORNO TRATTO DALLA BIBBIA (vedi nota sul mio profilo) * Cronache 1, 27 [13] Decimo, per il...
http:// fb.me/4x3ezczjI

Testo filtrato:

versetto giorno tratto bibbia vedi nota profilo cronache decimo


________
Testo originale:

un giorno di scuola, in terza media, pochi mesi fa,c'è stata una conferenza con Alberto Mieli, sopravvissuto ai
campi di concentramento. Non scorderò mai quel giorno dove ho potuto incontrarlo,rivolgergli le domande, sentire le
parole toccanti e terrificanti allo stesso tempo che uscivano dalla sua bocca, non dimenticherò il numero tatuato
sul suo braccio. Grazie di tutto Alberto perché ci fai ricordare ciò che l uomo vorrebbe dimenticare.

Testo filtrato:

giorno scuola terza media pochi mesi stata conferenza alberto mieli sopravvissuto campi concentramento scorderò mai
quel giorno potuto incontrarlo rivolgergli domande sentire parole toccanti terrificanti stesso tempo uscivano bocca
dimenticherò numero tatuato braccio grazie alberto ricordare ciò uomo vorrebbe dimenticare
________
Testo originale:

Fai un video di assaggi giapponesi???😍

Testo filtrato:

video assaggi giapponesi smilingfacewithhearteyes
```

Figura 2.6: Esempi di pulizia del testo

2.4 Ulteriori studi sul dataset

In questa sezione viene mostrato uno studio relativo all'analisi delle 20 parole più utilizzate dai rispettivi sessi, incluse le emoji.

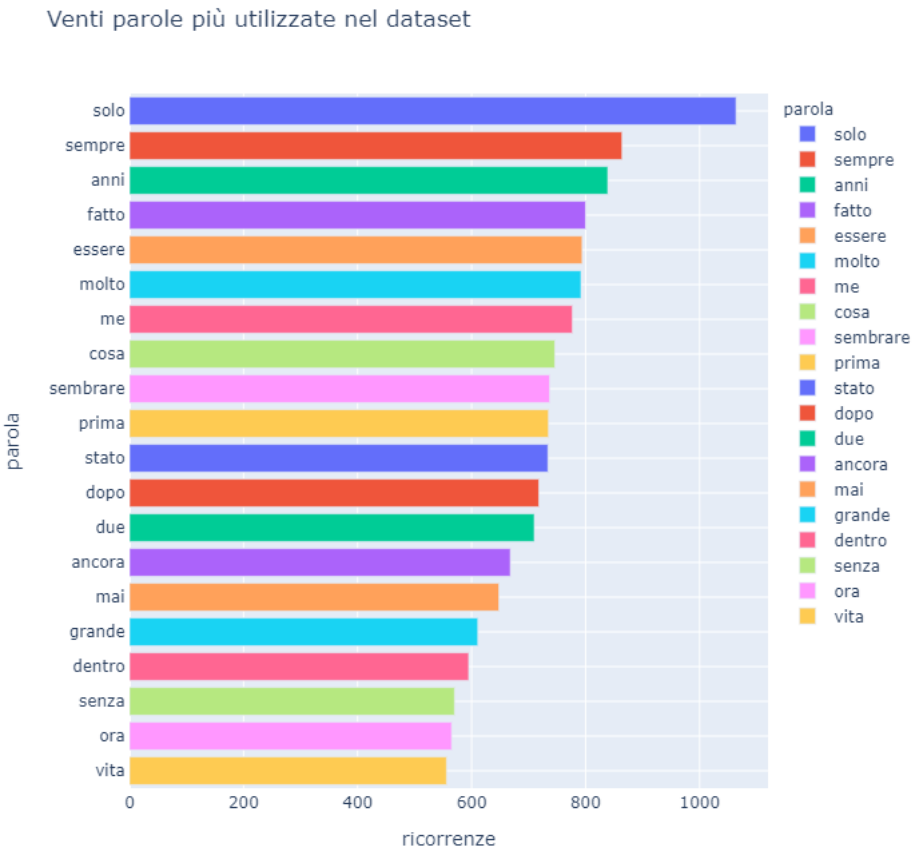

Figura 2.7: Venti parole più utilizzate in generale nel dataset

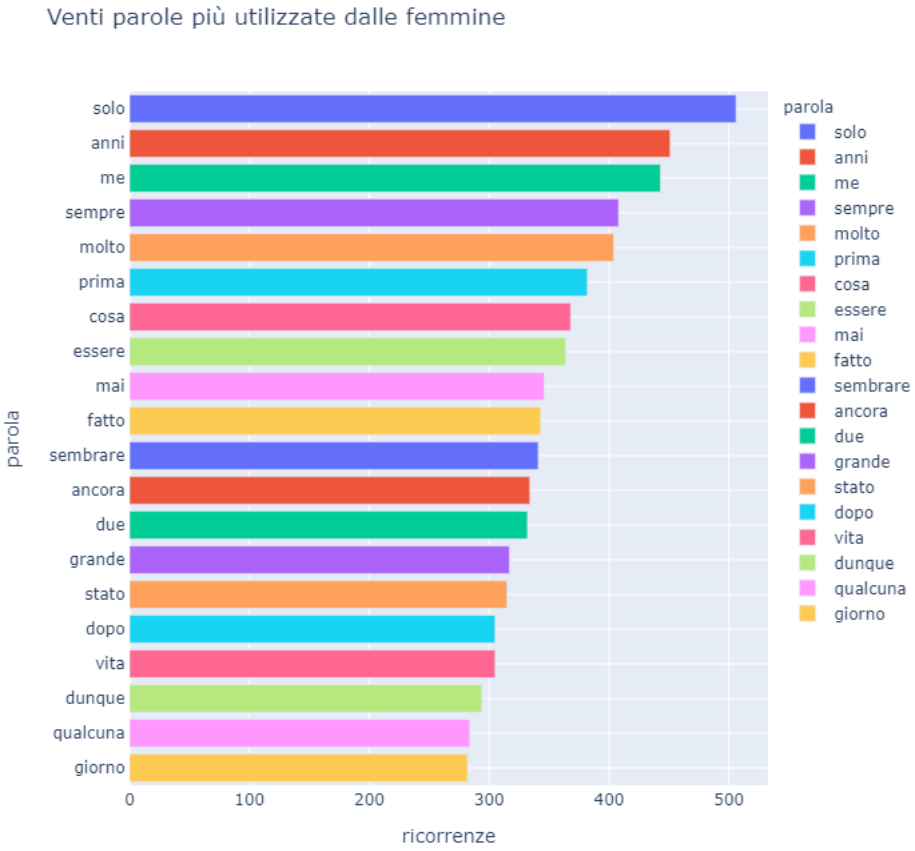

Figura 2.8: Venti parole più utilizzate dalle *femmine* nel dataset

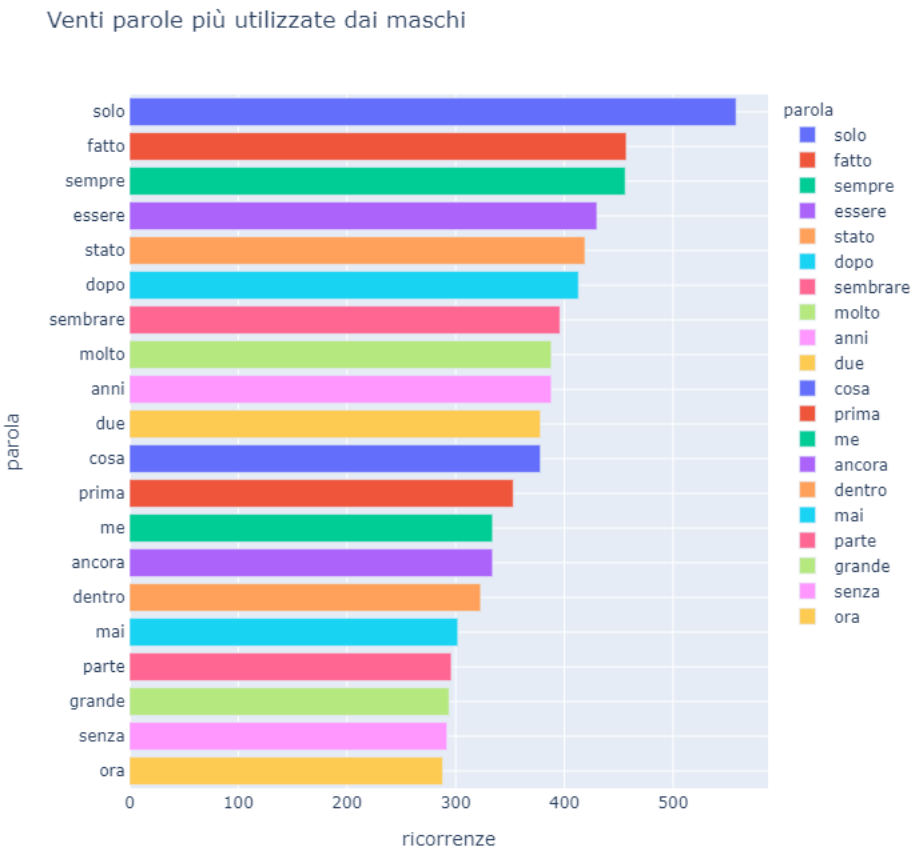

Figura 2.9: Venti parole più utilizzate dai *maschi* nel dataset

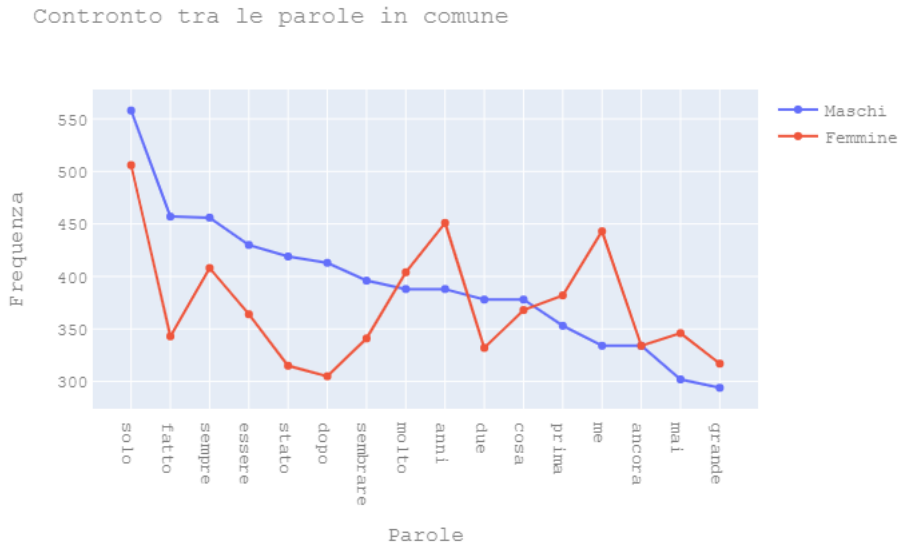

Figura 2.10: Intersezione tra le 20 parole più usate da entrambi i sessi

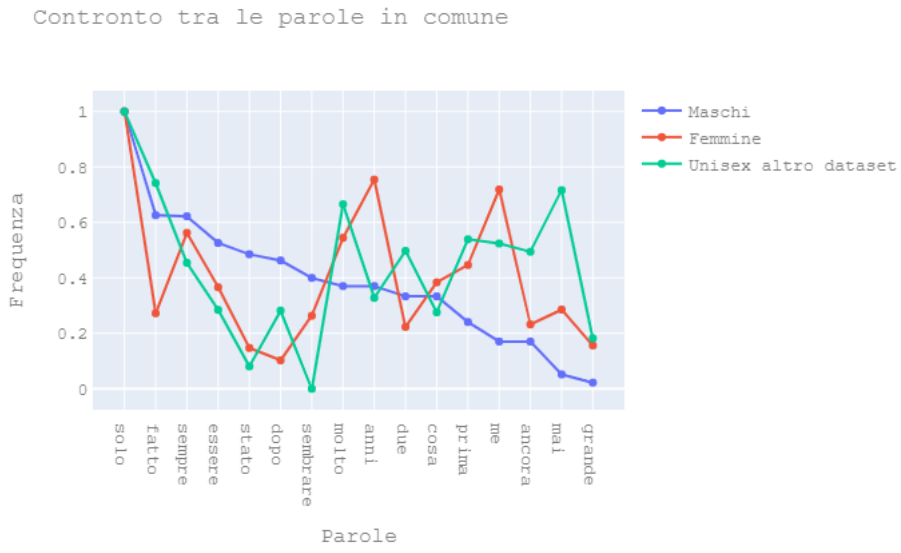

Figura 2.11: Intersezione tra le 20 parole più usate da entrambi i sessi comparate con un altro dataset

La parola più utilizzata è 'solo', con un numero di ricorrenze pari a 1064. Le femmine utilizzano più spesso le parole 'anni' e 'me' rispetto ai maschi, che utilizzano più frequentemente parole come 'fatto' e 'sembrare'. In merito all'intersezione tra le venti parole più frequenti utilizzate sia dai maschi che dalle femmine, notiamo che il numero di elementi presenti è 16, quindi più della metà delle parole vengono utilizzate da entrambe le classi, ma con frequenza diversa. In figura 2.11 viene mostrata l'intersezione tra le parole più usate dai maschi e dalle femmine in relazione alla frequenza di parole di un altro dataset, relativo ai sottotitoli di film e serie tv. Possiamo notare come le femmine utilizzino molto più frequentemente le parole 'anni' e 'me' come già precedenemente osservato, quindi queste due parole saranno prese in considerazione per il calcolo delle feature. Andiamo adesso ad analizzare il numero delle emoji utilizzate all'interno dei testi. Su circa 11000 record, solamente 376 contengono emoji, quindi il numero di occorrenze risulterà molto più basso rispetto a quello delle parole. Come vedremo, l'emoji 'face_with_tears_of_joy' sarà la più utilizzata da entrambe le classi e, come ci si aspettava, l'emoji 'Smiling Face with Heart Eyes' e 'Red Heart' saranno molto più utilizzate dalle femmine. Un'altra osservazione interessante è che, come possiamo notare dall'asse x dei grafici relativi allo studio delle emoji, le femmine utilizzano molte più emoji all'interno delle frasi rispetto ai maschi. Tutte le informazioni ottenute durante questo studio, saranno di fondamentale importanza per il calcolo delle feature.

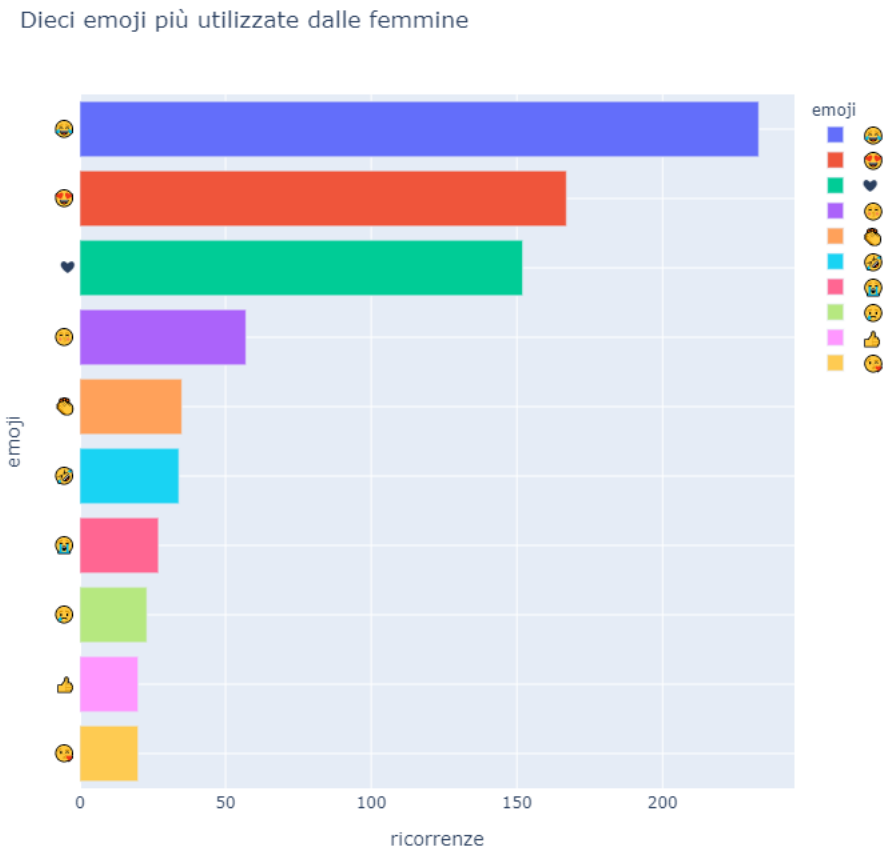

Figura 2.12: Le dieci emoji più utilizzate dalle femmine

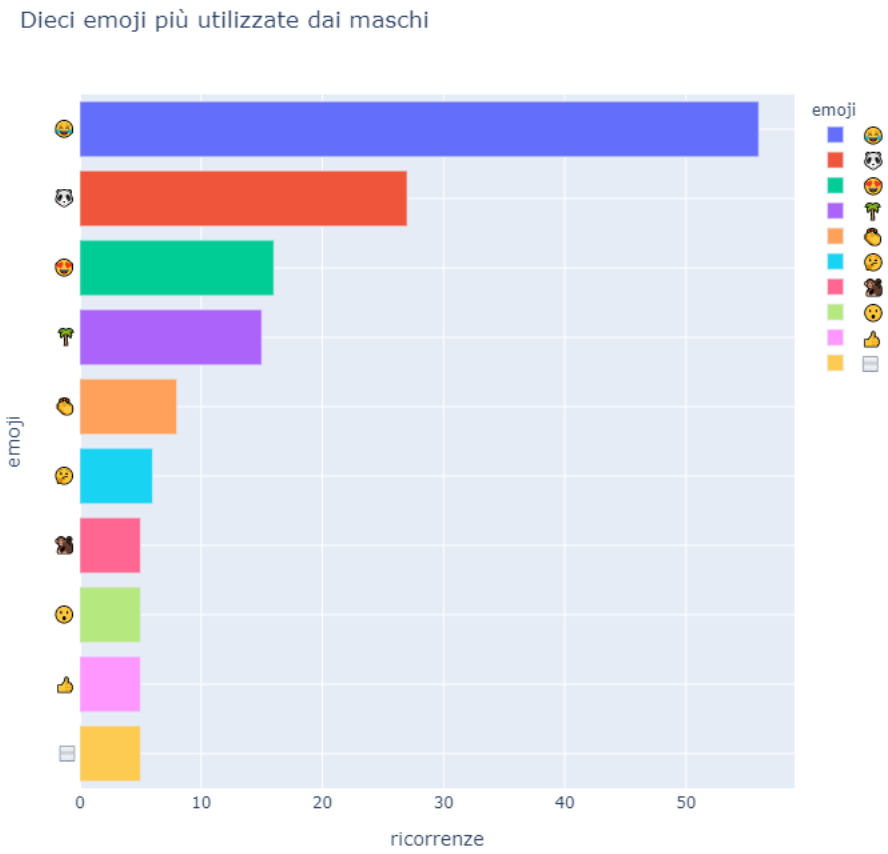

Figura 2.13: Le dieci emoji più utilizzate dai maschi

2.5 Feature Engineering

Per le creazione delle feature ho svolto un primo esperimento sulla lunghezza dei record 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 i maschi utilizzano in media 274,56 caratteri nei loro testi, mentre le femmine ne utilizzano 246,03. Come possiamo notare dai valori, i maschi sono più propensi a scrivere più caratteri rispetto alla femmine. Controlliamo adesso il numero di parole all'interno del testo. Per i maschi abbiamo in media 43,53 parole, mentre 39,65 parole per le femmine. Anche in questo caso, viene dimostrato che i maschi inseriscono più parole all'interno del testo rispetto le femmine. Sulla base di queste misure, le feature prese in considerazione per ogni record sono state:

1. Numero di caratteri
2. Numero di parole
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 20 più usate dai maschi
7. Numero di parole incluse nelle 20 più usate dalle femmine
8. Numero di punctuation
9. Numero di punti esclamativi (!)

10. Numero di punti interrogativi (?)
11. Numero di parole completamente maiuscole
12. Numero di parole che iniziano con la lettera maiuscola
13. Numero di riferimenti ad altri utenti (@user)
14. Numero di hashtag inseriti (#)
15. Numero di emoji totali all'interno della frase
16. Numero totale di emoji incluse nelle 5 più usate dai maschi
17. Numero totale di emoji incluse nelle 5 più usate dalle femmine
18. Numero di emoji di tipo 'Smiling Face with Heart Eyes' e 'Red Heart'
19. Numero di parole 'me' inserite
20. Numero di parole 'anno' inserite

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

2.6 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.

2.6.1 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 42307 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; in questo caso, la frase più lunga contiene 1138 parole. Questo valore è stato utilizzato per l'aggiunta di zero padding in post

a tutte le frasi.

```
# Mischio i record del dataset
df = shuffle(df)
df = df.drop(columns=['text_backup'])

# 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]
# Inizializzo il Tokenizer
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)
df['testo_token'] = tokenizer.texts_to_sequences(df['testo'])
print("La lunghezza prima il post padding è: ", len(df['testo_token'].iloc[1]))
#Voglio tutte le parole della stessa lunghezza quindi aggiungo zero padding
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 2.14: Creazione del vocabolario e aggiunta di zero padding

```
Frase selezionata:
[13415, 1693, 1931, 13416]
La lunghezza prima del Post Padding è:  4

[13415  1693  1931 ...     0     0     0]
La lunghezza dopo il Post Padding è:  1138
```

Figura 2.15: Esempio di applicazione di zero padding in post

Come possiamo vedere in figura 2.15, la lunghezza della frase selezionata prima del post padding è 4, mentre dopo l'applicazione è stata trasformata in un array di 1138 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 al livello Embedding di keras.

2.7 Elaborazione e Splitting dei dati

Dopo aver effettuato la pulizia del testo e creato le ventuno feature, ho notato come all'interno del dataset fossero presenti prima tutti i testi scritti

dai maschi (con classe 0) e poi tutti quelli scritti dalle femmine. 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 sesso
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',
         'sesso']]

#Ottengo i valori dal dataframe
X = df.iloc[:,0:21].values
Y = df.iloc[:,21].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 2.16: Suddivisione dei dati

Il passo successivo è stato quello di ordinare in maniera opportuna le 21 feature calcolate e la colonna 'sesso' 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 ventunesima colonna relativa alla label 'sesso') 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
- Quanti strati nascosti deve contenere la rete
- Quanti nodi devono essere presenti in ogni strato nascosto

Tutti i modelli sono 'Fully Connected Network'; 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 1138 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 20 neuroni, che rappresentano le venti 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 32 e 16 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 4 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.

```
# Adesso 1 diventa [0 1] e 0 diventa [1 0]. è il formato richiesto da Keras
from keras.utils import to_categorical
y_train = to_categorical(y_train, 2)

embedding_dim = 50
#MODELLO CON ENTRAMBI GLI INPUT
input_testo = Input(shape=(max_len,))
word_emb = Embedding(vocab_size, embedding_dim, input_length = max_len, trainable = True)(input_testo)
x = Flatten()(word_emb)
input_feature = Input(shape=(X_train_feature.shape[1],))

model_final = Concatenate()([x, input_feature])
model_final = Dense(32, activation='relu', bias_initializer='zeros')(model_final)
model_final = Dense(16, 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"])

plot_model(model_final,to_file="model_plot_due_input.png", show_shapes = True, show_layer_names = True)

history = model_final.fit(x=[X_train_embedding,X_train_feature], y = np.array(y_train),
                          batch_size = 128, epochs=4, verbose = 1,
                          validation_split=0.2)
```

Figura 3.1: Definizione del modello con doppio input

Dopo aver sperimentato con il modello sopra citato la previsione del sesso, 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 con l'aggiunta delle feature l'accuratezza rimanga sempre invariata, ma la metrica della Recall ha un miglioramento importante. Ci si aspettava questo risultato poichè gran parte delle feature sono dedicate al sesso femminile, quindi il modello diventa più performante nel riconoscere la loro classe di appartenenza.

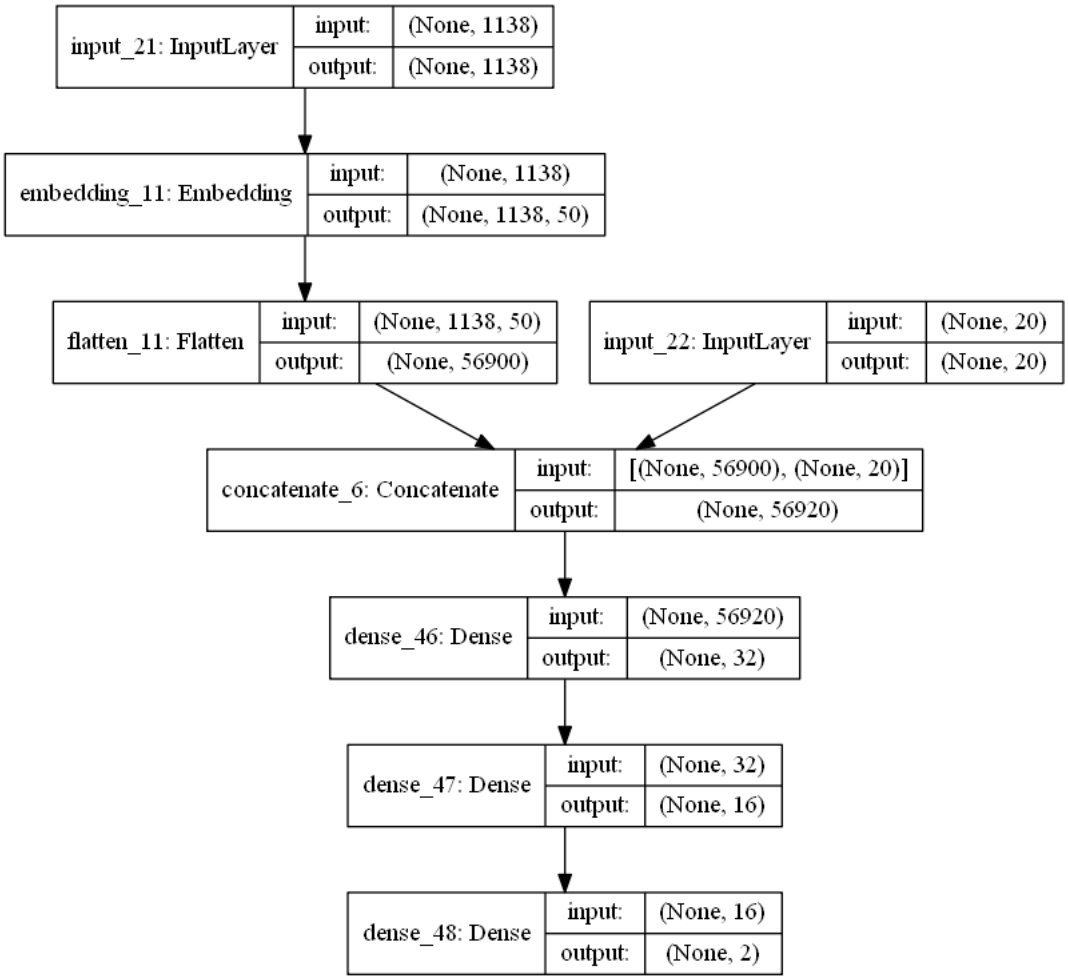

Figura 3.2: Struttura del modello ANN con doppio input per la previsione del sesso

3.2 Modelli con dizionari pre-allenati

Ho svolto poi ulteriori esperimenti con dei vettori pre allenati di parole scaricati dal sito: *https://wikipedia2vec.github.io/wikipedia2vec/pretrained/*. I file scaricati sono stati due e rappresentano rispettivamente ogni parola con 100 e 300 feature.

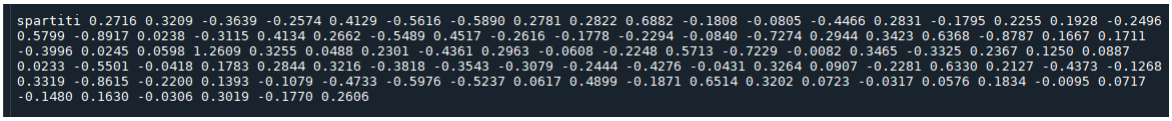

Figura 3.3: Esempio di parola rappresentata con 100 feature

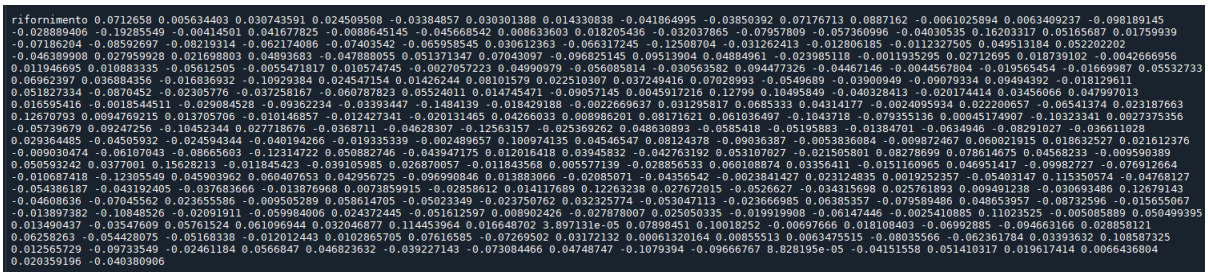

Figura 3.4: Esempio di parola rappresentata con 300 feature

3.2.1 Creazione della matrice dei pesi

Per poter utilizzare i vettori pre-allenati delle parole all'interno del modello è necessario costruire una matrice che sarà utilizzata come parametro del livello Embedding(). Il file in formato .txt scaricato è formato da due parti:

[parola,lista di valori]

che dovranno essere suddivise per creare un dizionario che associa ad ogni parola la sua rappresentazione in un vettore avente 100 (o 300) feature. La

matrice dei pesi va inizializzata tutta a zero e poi aggiornata in base alle parole del dizionario con cui si sta lavorando. Dato che i vettori erano già pre-allenati, ho impostato il parametro Trainable a False nel livello Embedding, così da non modificarli durante l'addestramento.

```
embedding_dim_modello_100 = 100
embeddings_index = dict()
f = open("../diz/modello_100_feature.txt", encoding = 'utf-8')
for line in f:
    values = line.split()
    try:
        word = values[0] #in posizione 0 c'è la parola
        coefs = np.array(values[1:], dtype = "float32") #tutto il resto sono valori
        embeddings_index[word] = coefs
        #print(embeddings_index[word])
    except ValueError:
        #print(values)
        continue
f.close()

#Creo la matrice dei pesi della dimensione adatta e la riempio di zero
embeddings_matrix = np.zeros((vocab_size, embedding_dim_modello_100))

for word, index in tokenizer.word_index.items():
  if index > vocab_size-1:
    break
  else:
    embedding_vector = embeddings_index.get(word)
    #se il mio vettore non è nullo lo metto nella matrice
    if(embedding_vector) is not None:
      #per ogni parola del vocabolario prendo la prappresentazione del dizionario già pre-allenato
      embeddings_matrix[index] = embedding_vector
```

Figura 3.5: Creazione della embeddings matrix

```
model_100 = Sequential()
model_100.add(Embedding(vocab_size, embedding_dim_modello_100,
                        input_length=maxlen_X, weights = [embeddings_matrix],
                        trainable = False))
model_100.add(Flatten())
model_100.add(Dense(5, activation="relu", bias_initializer='zeros'))
model_100.add(Dense(2, activation="softmax", bias_initializer='zeros'))
model_100.compile(optimizer='adam', loss="categorical_crossentropy", metrics=["accuracy"])


model_100.summary()
history = model_100.fit(np.array(X_train), np.array(y_train),
                        epochs=4, verbose=True,
                        batch_size=batch_size,
                        validation_data=(np.array(X_test), np.array(y_test_cat)))
```

Figura 3.6: Utilizzo della embeddings matrix

Fondamentale specificare che la dimensione dello spazio vettoriale in cui verranno incorporate le parole (variabile *embedding_dim_modello_100*) è stata impostata al valore 100 poichè in questo caso ogni parola veniva rappresentata con quel numero di feature. La stessa topologia è stata applicata al terzo modello, ma la dimensione dello spazio vettoriale in questo caso è stata impostata a 300. I modelli basati su dizionari pre-allenati sono stati addestrati per quattro epoche, poiché dopo questo numero la validation loss continuava sempre a salire. Inoltre, è stato necessario semplificare il modello, riducendo il numero di neuroni all'interno dei due livelli.

3.3 Salvataggio ed utilizzo dei modelli

Ho voluto infine inserire manualmente una frase inventata da me e controllare la previsione prodotta dal modello con doppio input che ha garantito migliori performance in assoluto. Per far questo, ho salvato prima la struttura del modello in formato *json*, e successivamente il valore dei pesi degli archi in formato *h5*. Questo ha permesso 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_30", "layers": [{"name": "input_41", "class_name": "InputLayer", "config":
{"batch_input_shape": [null, 1138], "dtype": "float32", "sparse": false, "name": "input_41"}, "inbound_nodes": []}, {"name": "embedding_22",
"class_name": "Embedding", "config": {"name": "embedding_22", "trainable": true, "batch_input_shape": [null, 1138], "dtype": "float32",
"input_dim": 42307, "output_dim": 50, "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":
1138}, "inbound_nodes": [[["input_41", 0, 0, {}]]]}, {"name": "flatten_22", "class_name": "Flatten", "config": {"name": "flatten_22",
"trainable": true, "data_format": "channels_last"}, "inbound_nodes": [[["embedding_22", 0, 0, {}]]]}, {"name": "input_42", "class_name":
"InputLayer", "config": {"batch_input_shape": [null, 20], "dtype": "float32", "sparse": false, "name": "input_42"}, "inbound_nodes": []},
{"name": "concatenate_12", "class_name": "Concatenate", "config": {"name": "concatenate_12", "trainable": true, "axis": -1}, "inbound_nodes":
[[["flatten_22", 0, 0, {}], ["input_42", 0, 0, {}]]]}, {"name": "dense_88", "class_name": "Dense", "config": {"name": "dense_88", "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":
[[["concatenate_12", 0, 0, {}]]]}, {"name": "dense_89", "class_name": "Dense", "config": {"name": "dense_89", "trainable": true, "units": 16,
"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_88", 0,
0, {}]]]}, {"name": "dense_90", "class_name": "Dense", "config": {"name": "dense_90", "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_89", 0, 0, {}]]]}],
"input_layers": [["input_41", 0, 0], ["input_42", 0, 0]], "output_layers": [["dense_90", 0, 0]]}, "keras_version": "2.2.4", "backend":
"tensorflow"}
```


Figura 3.7: Modello salvato in formato json

```
#Salvataggio del modello
model_json = model_final.to_json()
with open("Modello_salvato/modello_sesso.json", "w+") as json_file:
    json_file.write(model_json)
model_final.save_weights("Modello_salvato/modello_sesso.h5")
print("Il modello è stato salvato correttamente")
```

Figura 3.8: Salvataggio del modello

```
# Carico il modello
json_sesso = open('Modello_salvato/modello_sesso.json', 'r')
modello_json_sesso = json_sesso.read()
json_sesso.close()
loaded_model_sesso = model_from_json(modello_json_sesso)
loaded_model_sesso.load_weights("Modello_salvato/modello_sesso.h5")
print("Il modello 'Sesso' è stato caricato correttamente")
```

Figura 3.9: Caricamento del modello pre-allenato

Come possiamo notare dalle figure 3.10 e 3.11 il modello riesce a generalizzare bene quando il testo è chiaramente scritto da uno dei due sessi. Dato che il modello utilizzato per effettuare la predizione è quello composto da due input, la frase inserita è stata sia filtrata che trasformata in vettore numerico e successivamente sono state calcolate tutte le venti feature precedentemente descritte così da fornire i giusti input al modello.

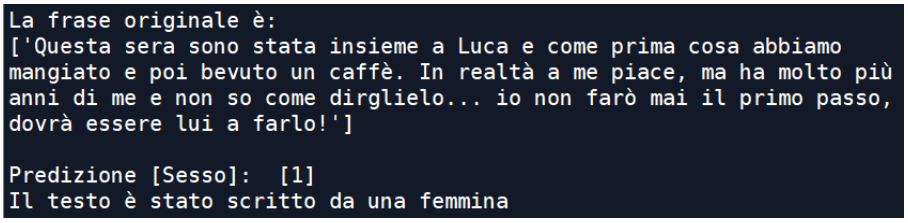

Figura 3.10: Previsione corretta della classe 1 (donna)

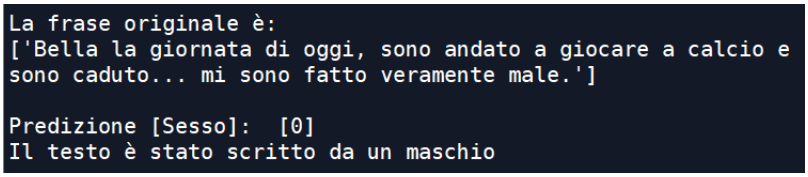

Figura 3.11: Previsione corretta della classe 0 (uomo)

3.4 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

I risultati del modello dedicato alla previsione del sesso che prende in input solamente il testo in formato numerico sono:

	Precision	Recall	F-score	Support
0	64%	79%	70%	1136
1	68%	50%	58%	1033

Accuracy: 65% su 2169 campioni.

Valori della matrice di confusione:

864	242
513	520

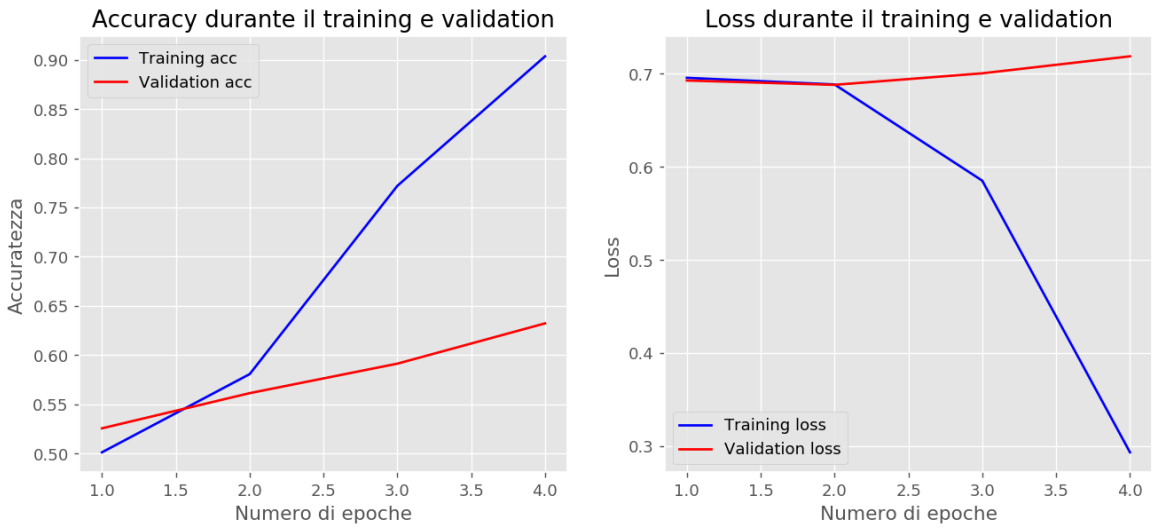

Figura 4.1: Accuracy e Loss durante l'addestramento del modello avente in input solo il testo in formato numerico

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

	Precision	Recall	F-score	Support
0	68%	61%	64%	1136
1	62%	69%	65%	1033

Accuracy: 65% su 2169 campioni.

Valori della matrice di confusione:

691	445
318	715

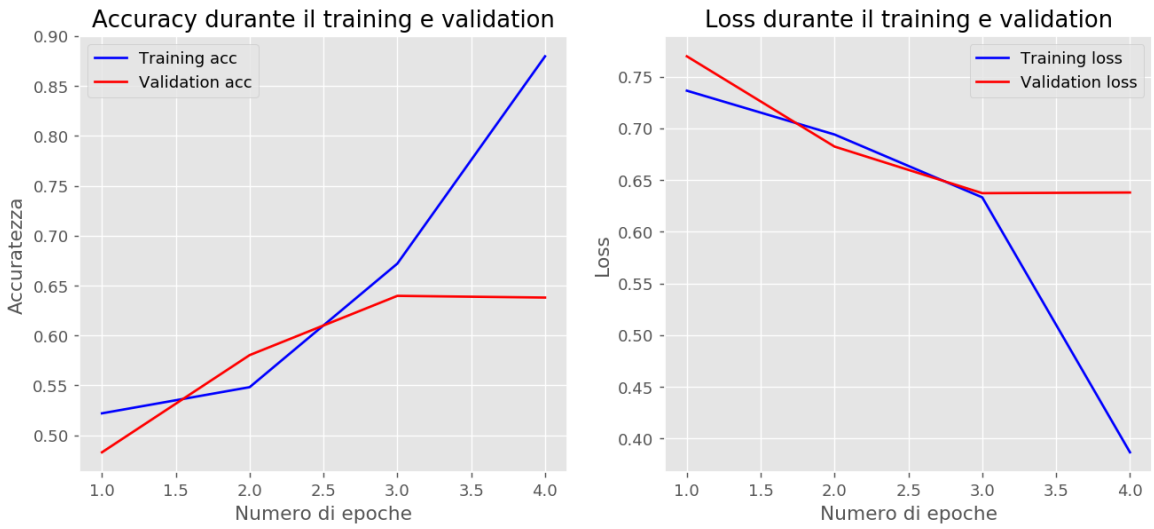

Figura 4.2: Accuracy e Loss durante l'addestramento del modello avente doppio input

I risultati del modello dedicato alla previsione del sesso che prende in input solamente le feature calcolate dal testo sono:

	Precision	Recall	F-score	Support
0	61%	21%	31%	1136
1	49%	85%	63%	1033

Accuracy: 51% su 2169 campioni.

Valori della matrice di confusione:

236	900
154	879

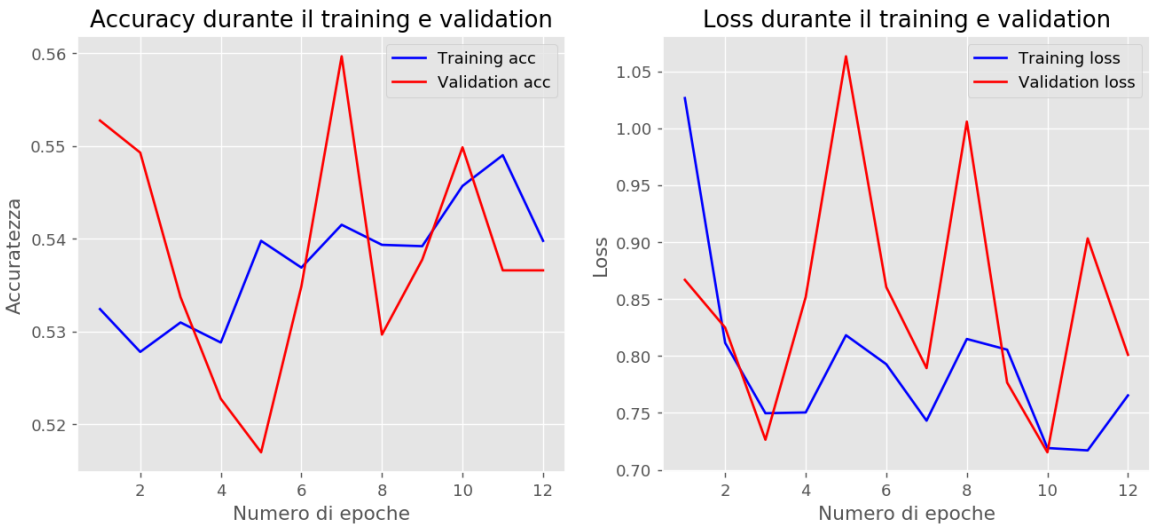

Figura 4.3: Accuracy e Loss durante l'addestramento del modello avente in input solo le feature calcolate dal testo

I risultati del modello addestrato con una rappresentazione delle parole a 100 feature sono i seguenti:

	Precision	Recall	F-score	Support
0	65%	57%	61%	1136
1	60%	69%	64%	1033

Accuracy: 63% su 2169 campioni.

Valori della matrice di confusione:

634	487
337	742

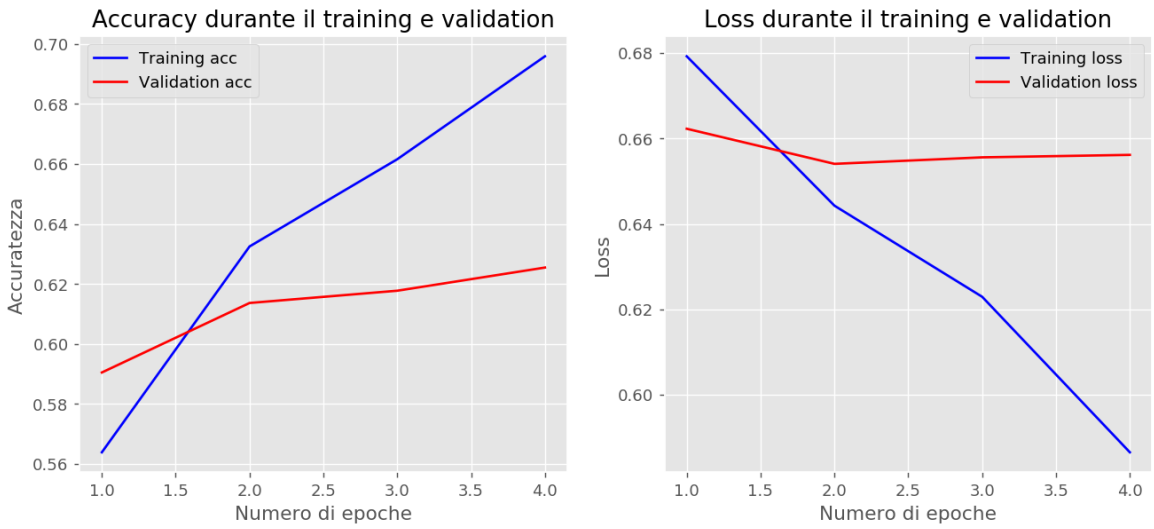

Figura 4.4: Accuracy e Loss durante l'addestramento del modello che lavora con rappresentazione delle parole a 100 feature

I risultati del modello addestrato con una rappresentazione delle parole a 300 feature sono i seguenti:

	Precision	Recall	F-score	Support
0	63%	58%	61%	1136
1	60%	65%	63%	1033

Accuracy: 62% su 2169 campioni.

Valori della matrice di confusione:

650	471
374	705

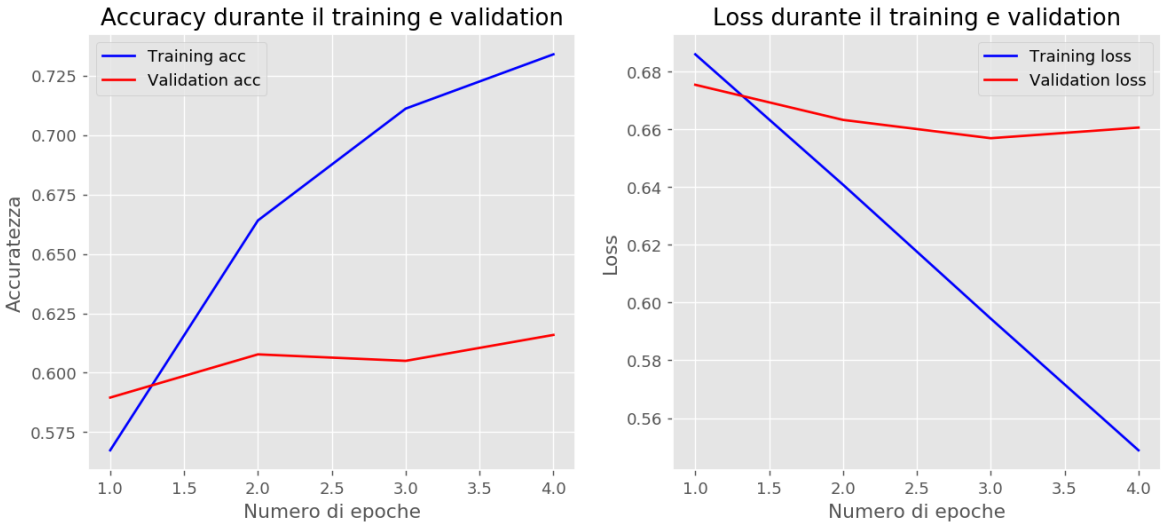

Figura 4.5: Accuracy e Loss durante l'addestramento del modello che lavora con rappresentazione delle parole a 300 feature

Conclusioni

L'obiettivo di questo progetto era mettere in relazione la sintassi ed il modo di scrittura delle persone con il loro sesso. Tutti i modelli hanno mostrato discrete prestazioni nella risoluzione del problema in oggetto, specialmente il modello avente doppio input, che grazie all'aggiunta delle feature ha subito un leggero miglioramento di tutte le performance garantendo omogeneità nelle misure. 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. Gli esperimenti sull'utilizzo effettivo dei modelli hanno dimostrato come il classificatore avente doppio input riesca a generalizzare bene quando il testo passato è chiaramente scritto da uno dei due sessi.