UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

PROGETTO DI DATA SCIENCE

Utilizzo di Word2Vec per la Classificazione dei Tweet

Docenti

Ursino Domenico,
Marchetti Michele
Buratti Christopher

A cura di

De Grazia Davide,
Piergallini Enrico,
Visi Andrea

ANNO ACCADEMICO 2024-2025

Indice

Elenco delle figure

CAPITOLO 1

IL DATASET SCELTO

Il dataset utilizzato in questo progetto è il **Sentiment140 dataset**, disponibile in formato CSV chiamato `training.1600000.processed.noemoticon.csv` e può essere scaricato da Kaggle al seguente link `https://www.kaggle.com/datasets/kazanova/sentiment140` . Questo dataset è uno dei più popolari per l'analisi del sentiment su Twitter, grazie alla sua dimensione e alla sua struttura relativamente semplice.

1.1 Caratteristiche del dataset

Il dataset contiene **1.600.000 tweet** raccolti tramite l'API di Twitter. Ogni riga rappresenta un singolo tweet annotato con il relativo sentiment e diverse informazioni aggiuntive.

In particolare, i campi sono i seguenti:

- **target**: indica la polarità del tweet. I valori possibili sono:
  - **0**: sentiment negativo;
  - **2**: sentiment neutrale (non utilizzato in questo progetto);
  - **4**: sentiment positivo.
- **ids**: l'ID univoco associato al tweet.
- **date**: la data e l'orario in cui il tweet è stato pubblicato.
- **flag**: specifica una query associata al tweet; se non è presente, il valore sarà `NO_QUERY`.
- **user**: il nome utente di chi ha pubblicato il tweet.
- **text**: il testo del tweet.

Di seguito è riportata una tabella con alcune righe di esempio tratte dal dataset per mostrare la sua struttura.

Target	ID	Data	Flag	Utente	Testo
0	2087	Sat May 16 23:58:44 UTC 2009	NO_QUERY	robotickilldozr	Lyx is cool
4	3432	Fri May 15 23:47:10 UTC 2009	NO_QUERY	happyguy01	I love pizza!
0	7543	Thu May 14 10:21:05 UTC 2009	NO_QUERY	sadgirl90	Feeling so tired today...

Tabella 1.1: Esempio di righe nel dataset Sentiment140

1.2 Motivazioni della scelta del dataset

Il dataset Sentiment140 è stato scelto per le seguenti motivazioni:

- **Dimensione del dataset**: con 1.6 milioni di tweet, il dataset offre una quantità sufficiente di dati per addestrare modelli complessi, riducendo il rischio di overfitting.
- **Facilità d'uso**: essendo strutturato come un file CSV, il dataset è facilmente leggibile e manipolabile con librerie Python come `pandas`.
- **Sentiment incluso**: la presenza di annotazioni per il sentiment consente di concentrarsi sullo sviluppo del modello senza dover etichettare manualmente i dati.

CAPITOLO 2

INTRODUZIONE A Word2Vec

2.1 Cos'è?

Il Word2Vec è una tecnica che serve per trasformare le parole in numeri, rappresentandole come vettori in uno spazio matematico. È stato sviluppato da Tomas Mikolov nel 2013 ed è uno degli strumenti più usati nel campo del Natural Language Processing (NLP) per capire il significato delle parole e le relazioni tra di esse.

Tecniche più vecchie, come la One-Hot Encoding, rappresentano ogni parola come un numero univoco, ma in questo modo il computer non riesce a capire le relazioni tra parole simili. Il Word2Vec, invece, rappresenta ogni parola come un vettore numerico che tiene conto del contesto in cui viene usata. In questo modo, parole che hanno un significato simile (es. "buono" e "fantastico") finiscono vicine nello spazio numerico, mentre parole molto diverse (es. "gatto" e "spoon") saranno distanti.

2.2 Come funziona?

Ci sono due metodi principali per addestrarlo:

1. **CBOW (Continuous Bag of Words)**: cerca di prevedere una parola basandosi sul contesto, cioè sulle parole vicine.
2. **Skip-Gram**: fa il contrario, ovvero usa una parola per prevedere il contesto, cioè le parole che si trovano vicino.

2.3 Perché è importante?

Il Word2Vec ha cambiato il modo in cui il linguaggio è rappresentato nei modelli di machine learning, rendendo più facile per le macchine capire il significato delle parole in base al contesto. Rappresenta, perciò, un passo fondamentale per compiti come l'analisi del sentiment, la traduzione automatica e la classificazione del testo.

CAPITOLO 3

SVILUPPO

3.1 Importazione e analisi preliminare del dataset

Per iniziare con l'analisi del sentiment si procede innanzitutto all'importazione del dataset. Poiché la sua dimensione è significativa, per velocizzare l'elaborazione selezioniamo casualmente il 10% del dataset totale utilizzando la funzione di campionamento (`sample`). Questo riduce i tempi di esecuzione senza compromettere troppo la qualità dei dati, dato che il 10% del dataset corrisponde comunque a un numero elevato di campioni.

Dopo aver importato il dataset, si è passati ad analizzare la distribuzione dei sentiment. I risultati, che possono essere visualizzati graficamente in Fig. 3.1, sono i seguenti:

- **Sentiment positivi (1)**: 80.188;
- **Sentiment negativi (0)**: 79.812.

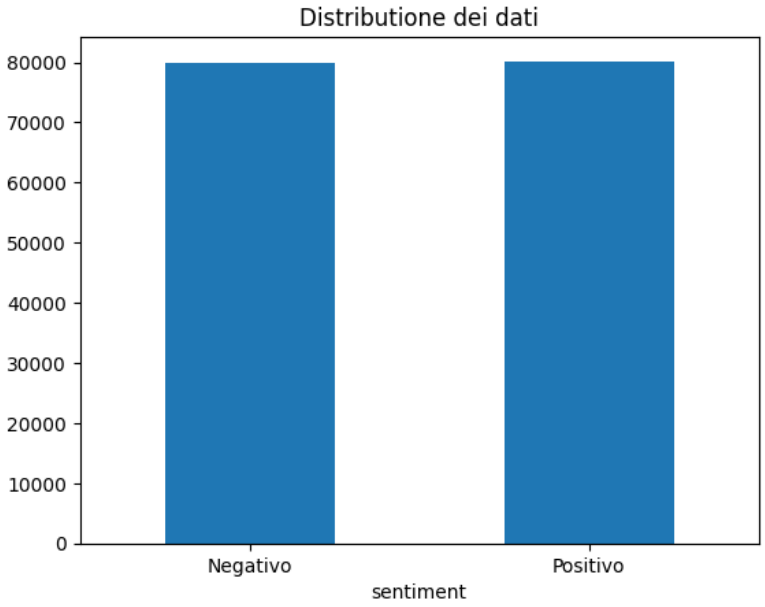

Figura 3.1: Distribuzione del sentiment nel dataset

Osserviamo, dunque, che il dataset è bilanciato, con un numero simile di tweet positivi e negativi.

3.2 Preprocessing del testo

Il preprocessing del testo è un processo che trasforma il testo in una forma più gestibile.

Nel caso dei tweet, i dati contengono spesso molte informazioni non testuali, come *mention*, *hashtag*, URL, emoji o simboli. Poiché normalmente i modelli NLP non sono in grado di elaborare questi dati, è necessario "ripulire" i tweet e sostituire tali token con informazioni significative per il modello.

I principali passaggi di preprocessing eseguiti sono i seguenti:

- **Conversione in minuscolo**: Tutto il testo viene convertito in minuscolo.
- **Sostituzione degli URL**: I link che iniziano con `http`, `https` o `www` vengono sostituiti con il token `<url>`.
- **Sostituzione dei nomi utente**: Le menzioni di utenti, come `@Mario`, vengono sostituite con il token `<user>`.
- **Sostituzione delle lettere consecutive**: Tre o più lettere consecutive vengono ridotte a due. Ad esempio, *Ciaoooo* diventa *Ciaoo*.
- **Sostituzione degli emoji**: Gli emoji vengono sostituiti utilizzando espressioni regolari. Ad esempio, `":)"` viene sostituito con `<smile>`.
- **Sostituzione delle contrazioni**: Le contrazioni vengono espanse al loro significato completo. Ad esempio, *can't* diventa *can not*.
- **Rimozione di caratteri non alfabetici**: Tutti i caratteri eccetto numeri, lettere e simboli predefiniti vengono sostituiti con uno spazio.

Il risultato del processo di pre-elaborazione del testo (per i primi 5 tweet) è mostrato nel DataFrame illustrato in Figura 3.2. Qui è possibile osservare sia il testo originale che la sua versione processata, ottenuta applicando i vari passaggi di preprocessing descritti.

	sentiment	text	processed_text
541200	0	@chrishasboobs AHHH I HOPE YOUR OK!!!	<user> ahh i hope your ok
750	0	@misstoriblack cool , i have no tweet apps fo...	<user> cool i have no tweet apps for my razr 2
766711	0	@TiannaChaos i know just family drama. its la...	<user> i know just family drama its lame hey...
285055	0	School email won't open and I have geography ...	school email will not open and i have geograp...
705995	0	upper airways problem	upper airways problem

Figura 3.2: Esempio di DataFrame contenente il testo originale e il testo processato.

3.3 Analisi dei dati

In questa sezione, analizziamo i dati pre-elaborati per comprenderne meglio le caratteristiche. Per farlo, genereremo delle **Word Cloud** per i tweet positivi (Figura 3.3) e negativi (Figura 3.4) presenti nel nostro dataset. Questi grafici ci permetteranno di visualizzare quali parole compaiono più frequentemente in ciascuna categoria di sentiment.

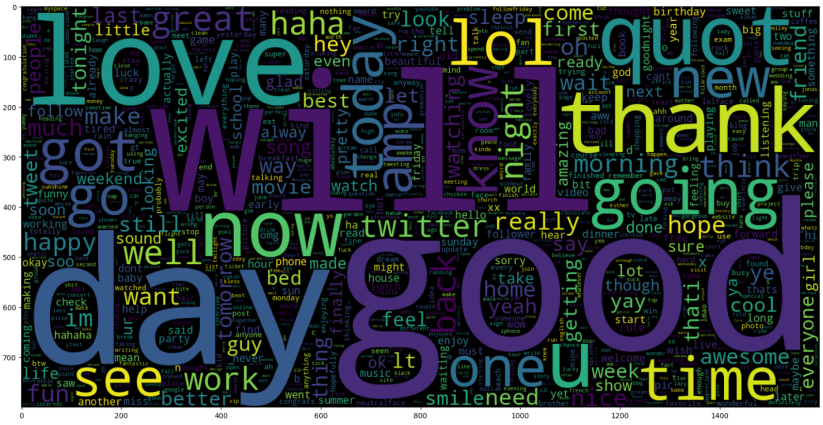

Figura 3.3: Word Cloud dei tweet positivi.

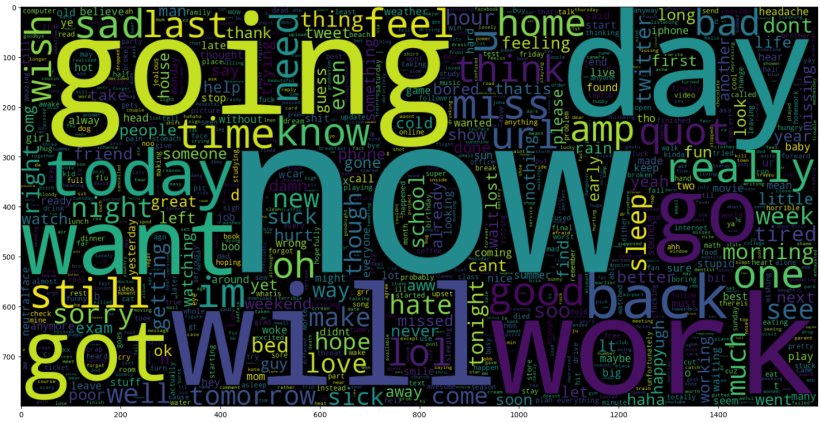

Figura 3.4: Word Cloud dei tweet negativi.

3.4 Training set e test set

Il passaggio successivo consiste nella suddivisione del dataset. Questa separazione è un passaggio cruciale per ridurre il rischio di *overfitting*, ovvero il fenomeno per cui il modello si adatta troppo bene ai dati di addestramento ma risulta inefficace quando applicato a nuovi dati.

La suddivisione del dataset è stata effettuata utilizzando la funzione `train_test_split()` della libreria *scikit-learn*, che mescola casualmente il dataset e lo divide in due insiemi distinti:

- **Training set**: Il sottoinsieme di dati utilizzato per addestrare il modello, che rappresenta l'**80%** dei dati totali.
- **Test set**: Il sottoinsieme di dati utilizzato per testare le prestazioni del modello, che rappresenta il **20%** dei dati totali.

La suddivisione 80-20 rappresenta un giusto compromesso, poiché garantisce un insieme sufficiente di dati per l'addestramento, preservando al contempo un numero adeguato di dati per una valutazione efficace del modello.

3.5 Creazione delle word embeddings con Word2Vec

Le Word Embeddings sono rappresentazioni vettoriali delle parole che catturano relazioni semantiche e contestuali. Per creare queste rappresentazioni, è stato utilizzato il modello

Word2Vec (introdotto nel Capitolo 2), che sfrutta una rete neurale per addestrare le embedding a partire dai dati forniti.

In particolare, è stata utilizzata la funzione `Word2Vec()`, che crea e addestra le word embedding utilizzando il dataset passato come input. Questo approccio consente di ottenere rappresentazioni numeriche delle parole che preservano il loro significato e la relazione con altre parole nel contesto.

Durante la configurazione e l'addestramento del modello Word2Vec, sono stati impostati i seguenti parametri:

- **`vector_size`:** Definisce il numero di dimensioni su cui il Word2Vec proietta le parole. Valori di dimensioni maggiori richiedono più dati di addestramento, ma possono generare modelli più accurati.
- **`workers`:** Specifica il numero di thread utilizzati per parallelizzare il processo di addestramento, velocizzando così l'intero processo.
- **`min_count`:** Questo parametro viene utilizzato per eliminare dal dizionario interno le parole meno frequenti. Parole che compaiono solo una o due volte in un corpus molto ampio sono spesso errori di battitura o rumore inutile. Inoltre, la loro scarsità di dati non è sufficiente per apprendere rappresentazioni significative, perciò è preferibile ignorarle.

```
# Creazione del dataset di addestramento per Word2Vec.
Word2vec_train_data=list(map(lambda x: x.split(), X_train))

# Definizione e addestramento del modello Word2Vec.
word2vec_model = Word2Vec(Word2vec_train_data,
                          vector_size=Embedding_dimension,
                          workers=8,
                          min_count=5)
```

3.6 Tokenizzazione e padding

Il passaggio successivo nel nostro approccio consiste nella tokenizzazione e nel padding del dataset.

La **tokenizzazione** è il processo attraverso cui un testo viene suddiviso in unità più piccole, chiamate *token*. Questi token possono essere definiti in base a diversi criteri, come la tokenizzazione basata sulle parole, sui caratteri o sulle sottoparole. La scelta del tipo di tokenizzazione dipende dall'applicazione specifica e dalle caratteristiche del modello utilizzato. Nel nostro caso, è stato deciso di adottare la tokenizzazione basata sulle parole, che risulta comunque particolarmente efficace.

Un ulteriore passaggio necessario è rappresentato dal **padding**. Le reti neurali infatti richiedono che gli input abbiano la stessa dimensione; tuttavia, i testi pre-elaborati, come frasi o documenti, spesso variano notevolmente in lunghezza. Il padding risolve questo problema uniformando le sequenze.

Per implementare la tokenizzazione e il padding, sono state utilizzate le seguenti funzioni:

- `Tokenizer()`: Una funzione che suddivide il dataset in una lista di token. I parametri utilizzati sono stati:
  - `filters`: Non sono stati applicati filtri poiché la pre-elaborazione è già stata effettuata.

  - `lower`: Non è stato applicato il minuscolo ai testi.
  - `oov_token`: Token di riempimento utilizzato per parole fuori dal vocabolario.

- `pad_sequences()`: Una funzione che applica il padding ai dati tokenizzati, garantendo che tutte le sequenze abbiano la stessa lunghezza.

Nel nostro caso, abbiamo impostato la lunghezza degli input (`input_length`) a **60**. Ciò significa che, dopo la tokenizzazione e il padding, ogni sequenza nel dataset avrà esattamente 60 token, indipendentemente dalla lunghezza originale delle frasi. Questa uniformità è essenziale per garantire che il modello riceva input coerenti durante l'addestramento e il testing.

3.7 Creazione della matrice di embedding

Per incorporare informazioni semantiche nel modello, è stata creata una **Embedding Matrix**. Questa matrice associa a ciascuna parola un vettore numerico rappresentativo delle sue caratteristiche. È stata costruita combinando il vocabolario prodotto dal tokenizer con le embedding generate tramite il modello *Word2Vec*. La dimensione della matrice risultante è data da:

$$\text{Vocab\_Length} \times \text{Embedding\_Dimensions}$$

3.8 Architettura del modello

Il modello è stato costruito come una rete neurale sequenziale che sfrutta un'architettura di tipo *LSTM* (*Long Short-Term Memory*) e altri layer per ottenere il contesto semantico dei testi. L'architettura è stata implementata utilizzando il framework *Keras* con il seguente codice:

```
def getModel():
    embedding_layer = Embedding(input_dim=vocab_length,
                                output_dim=Embedding_dimensions,
                                weights=[embedding_matrix],
                                input_length=input_length,
                                trainable=False)

    model = Sequential([
        embedding_layer,
        Bidirectional(LSTM(100, dropout=0.3, return_sequences=
    True)),
        Bidirectional(LSTM(100, dropout=0.3, return_sequences=
    True)),
        Conv1D(100, 5, activation='relu'),
        GlobalMaxPool1D(),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid'),
    ])
    return model
```

3.9 Compilazione del modello

Per addestrare correttamente il modello, è necessario prima compilarlo definendo tre elementi fondamentali:

- **Funzione di perdita (*Loss*)**: La funzione di perdita quantifica l'errore tra le previsioni del modello e i valori reali, fornendo una misura da minimizzare durante l'addestramento. In questo caso, utilizziamo la funzione **Binary Crossentropy**.
- **Metrica di valutazione (*Metric*)**: La metrica serve per monitorare le prestazioni del modello durante l'addestramento e il testing. Abbiamo scelto la metrica **Accuracy**, che misura la percentuale di previsioni corrette, una scelta comune per problemi di classificazione bilanciati.
- **Ottimizzatore (*Optimizer*)**: L'ottimizzatore regola i pesi del modello per minimizzare la funzione di perdita. In questo lavoro, abbiamo utilizzato **Adam**.

```
training_model.compile(loss='binary_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])
```

3.10 Addestramento del modello

L'addestramento del modello viene effettuato utilizzando il metodo `fit`, che aggiorna i pesi del modello per ridurre l'errore. I parametri generati durante l'addestramento sono salvati nella variabile `history`, utile per tracciare la curva di apprendimento. Di seguito, i principali argomenti utilizzati per l'addestramento:

- **`batch_size`**: Indica il numero di campioni utilizzati per ogni aggiornamento del gradiente. Nel nostro caso, è stato impostato a `1024`.
- **`epochs`**: Rappresenta il numero totale di epoche di addestramento. Abbiamo scelto un valore di `10`.
- **`validation_split`**: Specifica la frazione dei dati di addestramento destinati alla validazione, che non contribuiscono all'aggiornamento dei pesi, ma servono a valutare le prestazioni del modello. È stato impostato a `0.1`.
- **`callbacks`**: Include una lista di callback applicate durante il processo di addestramento. Nel nostro caso questa lista è stata configurata con i seguenti strumenti:
  - **`ReduceLROnPlateau`**: Questo callback riduce il tasso di apprendimento (*learning rate*) quando la perdita di validazione (`val_loss`) smette di migliorare.
  - **`EarlyStopping`**: Questo callback interrompe l'addestramento se una metrica monitorata smette di migliorare.

```
from tensorflow.keras.callbacks import ReduceLROnPlateau,
    EarlyStopping

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown
    =0),
```

```
    EarlyStopping(monitor='val_accuracy', min_delta=1e-4,
    patience=5)
]
```

- **`verbose`**: Controlla il livello di dettaglio dell'output durante l'addestramento. Abbiamo selezionato `1`, che corrisponde a una barra di avanzamento per una visualizzazione dettagliata.

Il codice utilizzato per l'addestramento è riportato di seguito.

```
history = training_model.fit(
    X_train, y_train,
    batch_size=64,
    epochs=10,
    validation_split=0.1,
    callbacks=callbacks,
    verbose=1,
)
```

3.11 Valutazione del modello

In questa sezione viene illustrata la valutazione del modello al fine di analizzarne le prestazioni e il comportamento.

3.11.1 Curve di apprendimento

I grafici riportati mostrano l'evoluzione delle metriche di perdita (*loss*) e accuratezza (*accuracy*) durante le diverse epoche di addestramento. Tali rappresentazioni sono fondamentali per analizzare il comportamento del modello e verificare se l'apprendimento avviene in modo corretto.

Il primo grafico (Fig. 3.5) illustra l'andamento dell'accuratezza sia per il set di addestramento che per quello di validazione. È possibile osservare che i valori di accuratezza aumentano progressivamente in entrambe le curve, fino a stabilizzarsi attorno al valore di 0.79, evidenziando un apprendimento stabile e coerente.

Il secondo grafico (Fig. 3.6) rappresenta invece l'andamento della metrica di perdita (*loss*) per i due set. Entrambe le curve mostrano un comportamento simile, con una riduzione costante della *loss* nel corso delle epoche. Questo indica che il modello sta migliorando le sue prestazioni senza segnali evidenti di *overfitting*, ossia senza adattarsi eccessivamente ai dati di addestramento a scapito della generalizzazione.

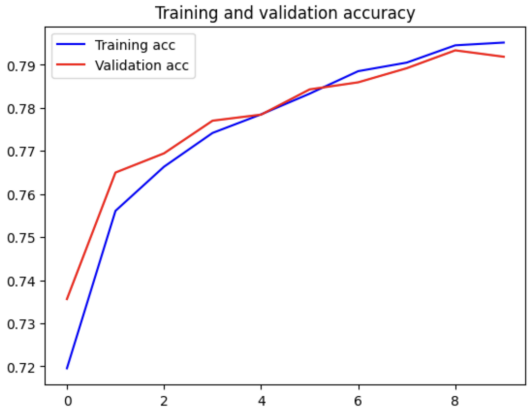

Figura 3.5: Training e Validation Accuracy

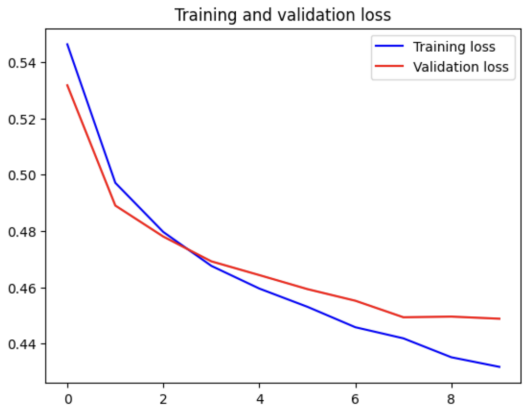

Figura 3.6: Training e Validation Loss

3.11.2 Matrice di confusione

La matrice di confusione è uno strumento fondamentale per valutare le prestazioni di un modello di classificazione. Essa fornisce una rappresentazione tabellare delle previsioni del modello confrontate con i valori reali, suddividendo i risultati in quattro categorie principali che sono True Positives (TP), True Negatives (TN), False Positives (FP) e False Negatives (FN).

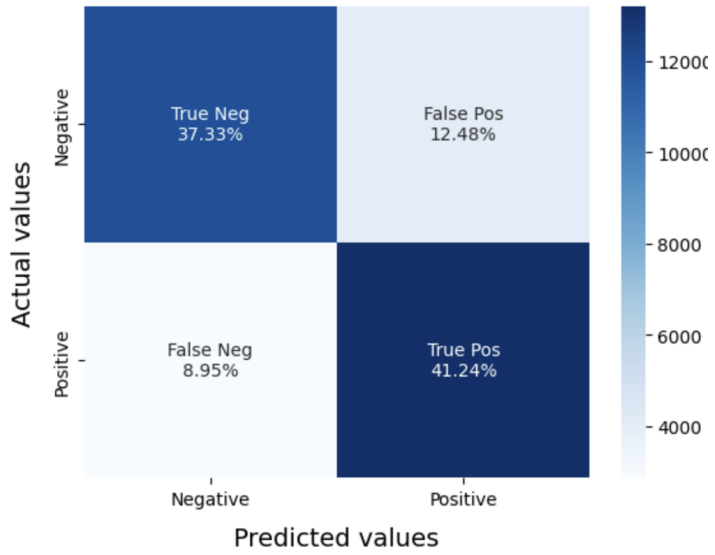

Figura 3.7: Matrice di confusione per il modello addestrato.

La matrice di confusione ottenuta, mostrata in Fig. 3.7, evidenzia che il modello riesce a classificare correttamente la maggior parte dei dati, con un leggero bias verso le previsioni positive, come evidenziato dalla proporzione di falsi positivi (ovvero il modello tende erroneamente a classificare come positivo un sentiment negativo).

Sulla base di questa matrice si possono calcolare metriche quantitative importanti come accuracy, precision, recall e F1-score, fornendo una visione chiara della capacità del modello di generalizzare sui dati di test. Le metriche ottenute, riportate di seguito (Fig. 3.8), evidenziano complessivamente delle buone prestazioni del modello.

	precision	recall	f1-score	support
0	0.81	0.75	0.78	15937
1	0.77	0.82	0.79	16063
accuracy			0.79	32000
macro avg	0.79	0.79	0.79	32000
weighted avg	0.79	0.79	0.79	32000

Figura 3.8: Pecision, Recall, F1-Score e accuracy del modello.