

**UNIVERSITÀ POLITECNICA DELLE MARCHE**

**FACOLTÀ DI INGEGNERIA**

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**



**Fine tuning di un modello BERT finalizzato alla  
sentiment analysis di tweet in lingua italiana**

**Studenti:**

Bedetta Alessandro  
Ascani Christian

**Docenti:**

Ursino Domenico  
Marchetti Michele

# Sommario

<b>BERT .....</b>	<b>2</b>
<b>Obiettivo e Dataset utilizzati.....</b>	<b>3</b>
<b>Preprocessing dei dati etichettati e splitting.....</b>	<b>4</b>
<b>Tokenizer e fase di addestramento.....</b>	<b>6</b>
<b>Fase di testing e risultati .....</b>	<b>10</b>
<b>Conclusioni e sviluppi futuri.....</b>	<b>12</b>

## BERT

Lo scopo di questo progetto è stato quello di prendere familiarità con l'utilizzo di BERT, eseguendo un fine tuning. Infatti, BERT (Bidirectional Encoder Representations from Transformers), rilasciato alla fine del 2018, è un metodo di preaddestramento delle rappresentazioni linguistiche. Infatti, è possibile utilizzare questi modelli per estrarre caratteristiche linguistiche di alta qualità dai dati testuali, oppure è possibile perfezionare tali modelli su un compito specifico (classificazione, riconoscimento di entità, question answering, ecc.). L'uso dei transformers in questo tipo di task risulta molto comodo in quanto questi ultimi riescono, durante le fasi di apprendimento, ad assegnare pesi differenti alle varie componenti dell'input (nel caso di BERT l'input è rappresentato dalle frasi del linguaggio naturale).

Uno dei componenti più importanti dei modelli Bert è il *Tokenizer*. Si tratta di un sottocomponente, il quale scopo è quello di frammentare le frasi in input sottoforma di token, ovvero componenti atomici del linguaggio che permettono a Bert di identificare facilmente le parole aventi significati assimilabili. Di seguito, viene riportato un esempio di tokenizzazione effettuata da Bert.

Word	Token(s)
surf	['surf']
surfing	['surf', '##ing']
surfboarding	['surf', '##board', '##ing']
surfboard	['surf', '##board']
snowboard	['snow', '##board']
snowboarding	['snow', '##board', '##ing']

In quanto lo scopo del nostro progetto è stato quello di eseguire un task di sentiment analysis su un insieme di tweet in lingua italiana, è stato necessario basarsi su un tokenizer e un modello pre-addestrato su questa lingua. Dopo aver provato diverse alternative, fra cui il “***bert-base-italian-xxl-cased***” e “***gilberto-uncased-from-camembert***” è stato scelto il modello “***bert\_uncased\_L-12\_H-768\_A-12\_italian\_alb3rt0***”, poiché quest'ultimo si è rivelato il migliore in termini di accuratezza e rapidità di addestramento (numero di parametri trainabili).

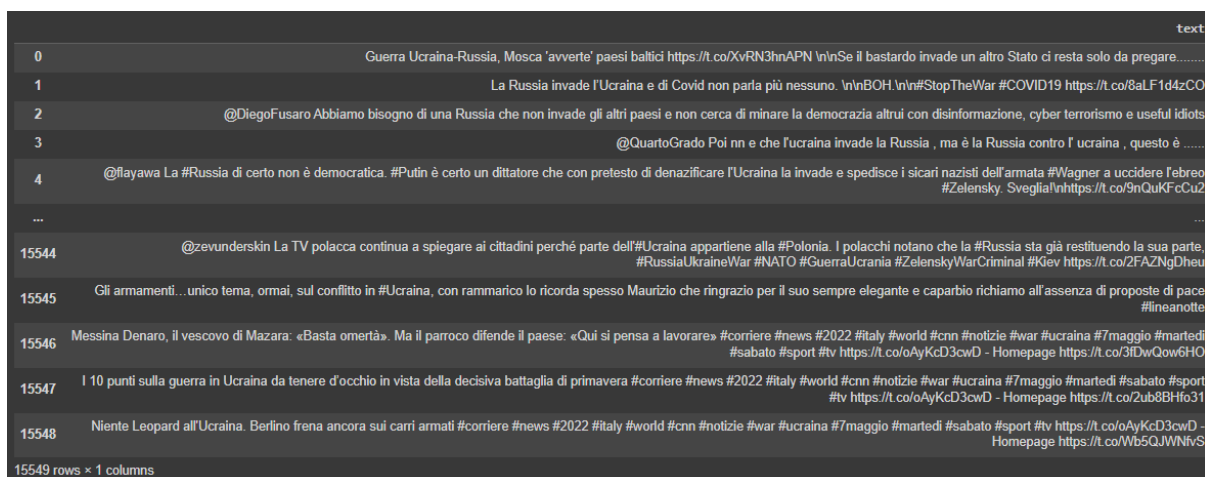
Di seguito si riportano i link alle pagine di HuggingFace e GitHub dove è possibile reperire ogni modello pre-addestrato e il corrispondente tokenizer utilizzato.

- <https://huggingface.co/dbmdz/bert-base-italian-xxl-cased>
- [https://huggingface.co/m-polignano-uniba/bert\\_uncased\\_L-12\\_H-768\\_A-12\\_italian\\_alb3rt0](https://huggingface.co/m-polignano-uniba/bert_uncased_L-12_H-768_A-12_italian_alb3rt0)
- <https://huggingface.co/idb-ita/gilberto-uncased-from-camembert>

## Obiettivo e Dataset utilizzati

Come accennato in precedenza, l'obiettivo di tale progetto è fare una sentiment analysis su una particolare tematica, ovvero il conflitto russo-ucraino, grazie all'analisi di tweet in lingua italiana.

Considerando la disponibilità di dataset corposi a riguardo, si vuole estrarre il sentiment a partire da un campione, in quanto solitamente sui social network gli utenti danno le proprie opinioni senza filtri e restrizioni (quindi, opinioni più veritiere). Il dataset è stato costruito a partire da diversi file, disponibili su Kaggle ([link 1](#) e [link 2](#)), contraddistinti da tweet in lingue differenti, contenuti con componenti non testuali (emoji, hashtag), ed informazioni inutili (come l'id dell'utente, il nome, l'ora, ecc.). Perciò, si è dovuto operare un processamento dei dati, in quanto sono stati caricati i singoli file in formato "csv", sono stati scansionati per rilevare i nomi delle colonne, e sono stati selezionati i record che presentavano "it" (codice identificativo) come lingua (colonna "language"). Come ultima fase dell'ETL, è stata selezionata solo la colonna corrispondente al tweet (colonna "text"), ottenendo un dataset ripulito ([Figura 1](#)).



	text
0	Guerra Ucraina-Russia, Mosca 'avverte' paesi baltici https://t.co/XvRN3hnAPN \n\nSe il bastardo invade un altro Stato ci resta solo da pregare.....
1	La Russia invade l'Ucraina e di Covid non parla più nessuno. \n\nBOH.\n\n#StopTheWar #COVID19 https://t.co/8aLF1d4zCO
2	@DiegoFusaro Abbiamo bisogno di una Russia che non invade gli altri paesi e non cerca di minare la democrazia altrui con disinformazione, cyber terrorismo e useful idiots
3	@QuartoGrado Poi nn e che l'ucraina invade la Russia , ma è la Russia contro l' ucraina , questo è .....
4	@flayawa La #Russia di certo non è democratica. #Putin è certo un dittatore che con pretesto di denazificare l'Ucraina la invade e spedisce i sicari nazisti dell'armata #Wagner a uccidere l'ebreo #Zelensky. Sveglia!\n\nhttps://t.co/9nQuKFcCu2
...	...
15544	@zevunderskin La TV polacca continua a spiegare ai cittadini perché parte dell'#Ucraina appartiene alla #Polonia. I polacchi notano che la #Russia sta già restituendo la sua parte, #RussiaUkraineWar #NATO #GuerraUcrania #ZelenskyWarCriminal #Kiev https://t.co/2FAZNgDheu
15545	Gli armamenti...unico tema, ormai, sul conflitto in #Ucraina, con rammarico lo ricorda spesso Maurizio che ringrazio per il suo sempre elegante e caparbio richiamo all'assenza di proposte di pace #lineanotte
15546	Messina Denaro, il vescovo di Mazara: «Basta omertà». Ma il parroco difende il paese: «Qui si pensa a lavorare» #corriere #news #2022 #italy #world #cnn #notizie #war #ucraina #7maggio #martedì #sabato #sport #tv https://t.co/oAyKcD3cwD - Homepage https://t.co/3IDwQow6HO
15547	I 10 punti sulla guerra in Ucraina da tenere d'occhio in vista della decisiva battaglia di primavera #corriere #news #2022 #italy #world #cnn #notizie #war #ucraina #7maggio #martedì #sabato #sport #tv https://t.co/oAyKcD3cwD - Homepage https://t.co/2ub8BHfo31
15548	Niente Leopard all'Ucraina. Berlino frena ancora sui carri armati #corriere #news #2022 #italy #world #cnn #notizie #war #ucraina #7maggio #martedì #sabato #sport #tv https://t.co/oAyKcD3cwD - Homepage https://t.co/Wb5QJWNivS

Figura 1: Dataset non etichettato

Tuttavia, poiché non è stato possibile trovare un dataset etichettato su questo tema specifico, è stato scelto di addestrare il modello su un [dataset diverso](#), al cui interno sono presenti anche tweet in lingua italiana, risalenti a diversi periodi temporali (2011 – 2017). Per ogni record, è associata una label: 0, se è recepito come un tweet negativo, 1 se neutrale, 2 nel caso in cui ci sia un sentiment positivo. In seguito, si è proceduto proprio con la fase di ETL.

## Preprocessing dei dati etichettati e splitting

Una volta trovati i dataset necessari alle fasi di training e di testing del nostro modello, lo step successivo è stato quello di ripulire i dati da tutte le impurità ed informazioni superflue, le quali avrebbero potuto influenzare negativamente il risultato finale.

Queste operazioni di data cleaning sono state svolte grazie al supporto della libreria “*Pandas*”; quest’ultima consente di eseguire in modo molto agile operazioni di manipolazione su dataset rappresentati attraverso il formato DataFrame.

Il primo passo è stato quello di importare la base dati grezzi, sfruttando la libreria “*datasets*” offerta proprio da “*HuggingFace*”. Quest’ultima ci ha permesso di importare i dati direttamente nell’ambiente Colab, senza dover ricorrere a lunghe operazioni di download e upload su Drive.

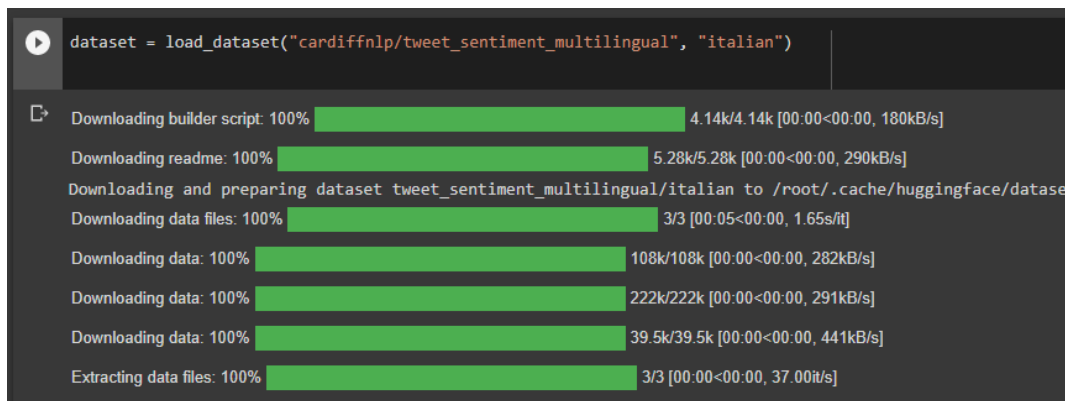


Figura 2: Caricamento del dataset etichettato

Una volta caricata la base dati integrale sono iniziate le operazioni di manipolazione vere e proprie. Il dataset di partenza si presentava suddiviso in tre partizioni: training, validation e test. Consci del fatto che avremmo dovuto svolgere operazioni sull’intero set di dati, la prima operazione da noi svolta è stata quella di concatenare le sottocomponenti del dataset convertite in Pandas DataFrame.

```
df_train = pd.DataFrame(dataset["train"])
df_test = pd.DataFrame(dataset["test"])
df_val = pd.DataFrame(dataset["validation"])

df = pd.concat([df_train, df_val, df_test])
df.head()
```

Figura 3: Merge delle partizioni

Convertiti i dati in un formato molto più maneggevole, il secondo step è stato quello di applicare le funzioni di pulizia vere e proprie. Nello specifico, la procedura da noi applicata si trova racchiusa interamente nella funzione

`clean_text()`; quest'ultima, come prima operazione applica un filtro alle frasi date in input eliminando tutti i caratteri speciali (quelli che non appartengono all'insieme a-z, A-Z, ".", "?", "!", ",", ";") e i segni di punteggiatura; fatto ciò, dalla frase vengono rimosse anche tutte le componenti che non hanno a che fare con il messaggio vero e proprio, ad esempio l'intestazione `http` e la stringa `user` (presente in tutti i tweet, indica gli autori del post).

```
def clean_text(text):  
  
    text = text.lower()  
  
    text = re.sub(r"^[a-zA-Z?!.;]+", " ", text)  
    text = re.sub("user", " ", text)  
    text = re.sub("http", " ", text)  
    text = re.sub("https", " ", text)  
  
    text = re.sub(r"http\S+", "", text)  
  
    html=re.compile(r'<.*?>')  
  
    text = html.sub(r'', text)  
  
    punctuations = '@#!?+&*[]~%.:;/()$=><|{}^' + "'`" + ' _'  
    for p in punctuations:  
        text = text.replace(p, '') #Removing punctuations
```

*Figura 4: Funzione di pulizia dei testi*

Prima di terminare, la funzione si occupa anche di rimuovere tutti i simboli speciali come emoji, simboli di mappe e flags iOS. Fatto ciò, ci troviamo con un dataset pulito da ogni tipo di impurità e contenente esclusivamente le informazioni legate ai messaggi veri e propri.

Una volta concluse queste operazioni di pulizia, l'unica cosa rimasta da fare era quella di partizionare nuovamente il dataset nelle tre partizioni training, validation e test. A causa della limitata quantità di dati a disposizione, abbiamo scelto di destinare la maggior parte di questi alla fase di training, eseguendo uno split 90/5/5. Infatti, i tweet possono essere di vario genere e trattare di tematiche molto differenti, di conseguenza il modello con pochi dati potrebbe non riuscire a cogliere le possibili sfumature (anche ironiche) che possono caratterizzare i tweet.

```
index_tr = int(len(new_df)*0.90)  
index_ts = int(len(new_df)*0.95)  
  
train_df = new_df.loc[:index_tr,:]  
val_df = new_df.loc[index_tr:index_ts,:]  
test_df = new_df.loc[index_ts:,:]
```

*Figura 5: Split in training, validation e testing set*

## Tokenizer e fase di addestramento

Eseguito lo split, il passo successivo è stato quello di preparare i dataframe che, in seguito, verranno dati in ingresso al modello Bert; per far ciò, i tweet sono stati processati dal tokenizer che li ha trasformati in gruppi di token (interpretabili dal modello Bert).

```
tokenizer = BertTokenizer.from_pretrained(checkpoint, do_lower_case=True)

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=MAX_LEN)

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
df
tokenized_datasets = tokenized_datasets.remove_columns(["text"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch")
```

Figura 6: Inizializzazione del tokenizer

Viene riportato di seguito un esempio di tokenizzazione da parte del modello scelto (Figura 7: Esempio di tokenizzazione; come è possibile osservare, nonostante i processi di pulizia, non è possibile eliminare al 100% tutti i sintagmi privi di significato; ciò è riconducibile alla natura stessa del dataset, il quale contenendo post reali, scritti da persone reali, contengono inevitabilmente delle imprecisioni ed incorrettezze lessicali. Inoltre, vengono aggiunti i token “speciali”, come [CLS] che indica l’inizio della frase [SEP] che indica la fine del tweet e [PAD] che specifica token per il padding (al fine di raggiungere la lunghezza massima impostata in fase iniziale).

gufi gelati buona scuolama anema ta mmuort  
Label: 0

Tokens	Token IDs	Attention Mask
[CLS]	2	1
gufi	7304	1
gelati	13624	1
buona	159	1
scuola	81	1
##ma	1306	1
anema	78135	1
ta	1074	1
mm	1530	1
##uor	90155	1
[SEP]	3	1

Figura 7: Esempio di tokenizzazione

Una volta pronto il dataset, si è proceduto con la definizione dei parametri fondamentali per eseguire il fine-tuning del modello Bert pre-addestrato.

Sperimentando con diversi valori, abbiamo determinato che i comportamenti (in termini di accuracy) migliori si avevano impostando:

1. **Learning rate** :  $3 \cdot 10^{-5}$
2. **Training epochs** : 5
3. **Batch size** : 32
4. **Max Length**: 11

In realtà, il numero di epoche rappresenta una limitazione superiore, in quanto è stato fin da subito chiaro che il modello raggiungeva la sua massima precisione dopo due o tre epoche al massimo. Per questo motivo, si è previsto di salvare solamente il modello migliore.

Per quanto riguarda il DataLoader, deve conoscere le dimensioni del batch per l'addestramento, dopodiché si creano i DataLoader per il training e quello per la validation, in quanto il modello richiede in ingresso dei tensori. Prima di procedere con il fine tuning vero e proprio, bisogna considerare che BERT richiede input formattati in maniera specifica. Per ogni frase di input tokenizzata precedentemente, è necessario creare:

- “input ids”: una sequenza di numeri interi che identifica ogni token di input con il suo numero di indice nel vocabolario del tokenizzatore;
- “attention mask”: una sequenza di 1 e 0, con 1 per tutti i token di input e 0 per tutti i token di padding;
- “labels”: etichetta associata alla frase tokenizzata.

Per modificare il modello pre-addestrato, si utilizza una serie di interfacce, fornite da Huggingface PyTorch ed in questo caso viene utilizzato un modello BERT con un singolo strato lineare aggiunto per la classificazione, chiamato BertForSequenceClassification. Man mano che si forniscono i dati di input, l'intero modello BERT pre-addestrato e lo strato di classificazione aggiuntivo vengono addestrati per il nostro compito specifico. Poiché gli strati pre-addestrati codificano già molte informazioni sulla lingua, l'addestramento del classificatore è relativamente poco costoso. Piuttosto che addestrare da zero tutti gli strati di un modello di grandi dimensioni, è come se dovessimo addestrare solo lo strato superiore, con un po' di modifiche ai livelli inferiori per adattarli al nostro compito, soprattutto perché è un task di SA. A questo punto si può procedere con l'addestramento che si compone di diversi passi; in dettaglio:

- Ciclo di addestramento:
  - Si indica al modello di calcolare i gradienti impostando il modello in modalità di addestramento.
  - Gli input e le etichette dei dati vengono spaccettati
  - Si caricano i dati sulla GPU per l'accelerazione



- Si cancellano i gradienti calcolati nel passaggio precedente. In pytorch i gradienti si accumulano per impostazione predefinita, a meno che non li si cancelli esplicitamente.
- Passaggio in avanti, cioè si danno i dati in ingresso alla rete
- Passaggio all'indietro (backpropagation)
- Dire alla rete di aggiornare i parametri (optimizer.step)
- Variabili di tracciamento per monitorare i progressi
- Ciclo di valutazione:
  - Si indica al modello di non calcolare i gradienti impostando il modello in modalità di valutazione.
  - Gli input e le etichette dei dati vengono spaccettati
  - Si caricano i dati sulla GPU per l'accelerazione
  - Passaggio in avanti, cioè si danno i dati in ingresso alla rete
  - Calcolo della perdita sui dati di validazione e tracciamento delle variabili per monitorare i progressi.

Al fine di valutare le performance dei modelli prodotti, è stata sfruttata la libreria “*Evaluate*”, offerta da HuggingFace. Questa presenta al suo interno decine di metriche di valutazione mirate a stimare la bontà di svariati modelli di machine learning. Nel nostro caso, l'unica metrica utilizzata è stata la *accuracy*, che consiste nel rapporto tra le predizioni corrette e il numero totale di campioni. In aggiunta, oltre a monitorare la loss di training, è stata considerata anche la loss di validazione, in modo da capire se si verificasse underfitting, cioè un modello che non riesce a generalizzare (ad esempio, a causa di rumore o di un dataset non consono), o overfitting (il modello si specializza troppo su dati in ingresso, per cui nella validazione non si ottengono risultati sufficienti). Come si evince dalla [Figura 8](#)Figura 9, la loss di training diminuisce nel tempo, mentre la loss di validazione aumenta, mostrando che il modello sta overfittando.



Figura 8: Andamento delle loss di addestramento e di validazione

L'accuratezza massima si raggiunge solamente in corrispondenza della terza epoca ed è pari al 73,78% (Figura 9).

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.925437	0.694065	0.70650	0:00:11	0:00:00
2	0.620293	0.743953	0.71100	0:00:12	0:00:00
3	0.369363	0.825390	0.73775	0:00:12	0:00:00
4	0.189254	0.967905	0.72700	0:00:16	0:00:00
5	0.103111	1.076742	0.73150	0:00:12	0:00:00

*Figura 9: Statistiche di training*

In seguito alla fase di allenamento, si può affermare che i risultati ottenuti non sono ottimali, come in altri task. Tuttavia, ciò è dovuto al fatto che il modello pre-addestrato entra in difficoltà, poiché tweet simili possono avere etichette differenti (spesso, una notizia negativa può anche essere riportata da una testata giornalistica, con una sfumatura più neutra). Inoltre, i tweet possono essere scritti in italiano non corretto.

## Fase di testing e risultati

Una volta ottenuto il modello definitivo, si esegue il test sulla partizione dedicata, eseguendo gli stessi passaggi fatti in precedenza. Perciò, considerando il medesimo tokenizer, si danno in ingresso i dati preparati al modello. I tweet processati sono 152 e si ottiene un'accuratezza pari al 73%, indicando dei risultati in linea con la fase precedente.

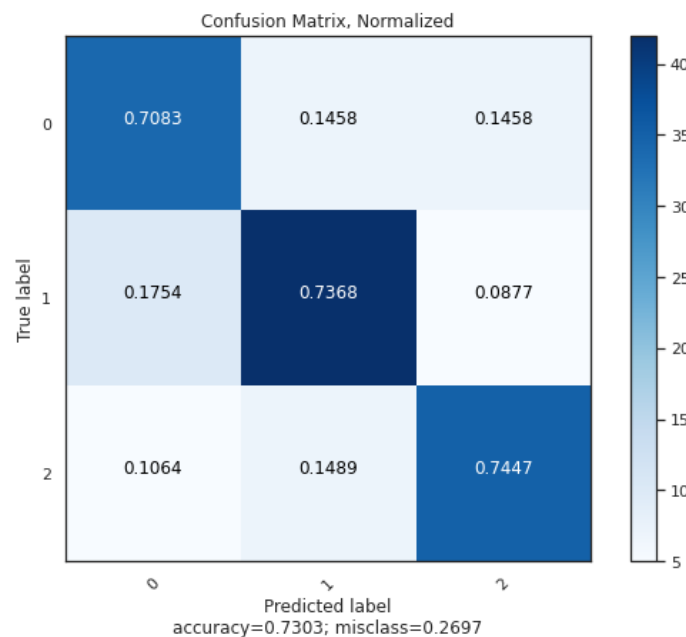


Figura 10: Matrice di confusione per il test

Dalla matrice di confusione ([Figura 10](#)), si conferma il fatto che molto spesso il modello non riesce a predire correttamente la classe nei casi negativi e neutrali. Tuttavia, bisogna comprendere che si ha a disposizione un numero ridotto di campioni di test, dunque le percentuali potrebbero essere differenti nel caso di un maggiore numero di dati.

Una volta testato il dataset etichettato, abbiamo utilizzato il modello sul dataset non etichettato, creato in fase iniziale. Come precedentemente analizzato, questo set di dati conteneva circa 15.000 tweet relativi alla guerra in Ucraina. Dato che non sono disponibili le classi per ogni tweet, non è possibile calcolare una metrica utile; però, osservando i dati, si può rilevare che in molti casi il modello predice correttamente ([Figura 11](#)). Nei casi in cui c'è ambiguità, si può notare come il modello non riesca effettivamente a discriminare le diverse casistiche; per esempio, il primo tweet, nonostante utilizzi la parola “pregare”, esprime negatività, eppure il modello lo interpreta come un testo neutrale. Invece, nel terzultimo tweet nella figura vengono utilizzate parole come “fiori” o “canzoni”, in chiave ironica, di conseguenza viene associata la label 2, anche se il tweet ha una connotazione palesemente negativa.

tweets	label
guerra ucraina russia mosca avverte paesi baltici s tco xvrn hnaphn bastardo invade altro stato resta solo pregare	1
russia invade ucraina covid parla pi nessuno boh stopthwar covid s tco alf d zco	1
diegofusaro bisogno russia invade altri paesi cerca minare democrazia altrui disinformazione cyber terrorismo useful idiots	1
quartogradio poi nn ucraina invade russia russia ucraina	0
flayawa russia certo democratica putin certo dittatore pretesto denazificare ucraina invade spedisce sicari nazisti armata wagner uccidere ebreo zelensky sveglia s tco nqukfccu	0
marcell lucafer quindi russia invade nazione nato beh sembra logico discorso	1
manolo loop ucraina russia invade commettendo crimini guerra conseguenza ucraina difende vs gaza hamas lancia missili verso aree abitate crimine guerra israele conseguenza israele difende rifate calcoli	0
sleepyjoie biden putin invade russia neanche seconda guerra mondiale successo poveri usa you are really trouble s tco cqiyylugp	1
anto doctorj vanabeau san marino russia ucraina san marino invade italia difenderemo difendere ucraina vacci posa telefono prepara zainetto	1
alberto zz guidocrosetto invece russia attacca invade stato sovrano mettiamo insegnato niente genocidio cecenia invasione georgia annessione unilaterale crimea donbass firmetta putin hitler ii	0
putin invade russia ah beh namo bene s tco cuzhvb aid	0
gt gt inizio guerra mondiale gt gt inizio ii guerra mondiale gt gt russia invade ucraina ama numeri coincidenze	1
volevo capire russia invade paese sovrano ucraina annetterlo nessuno tenta contrastarla militarmente paura guerra nucleare significa putin invadesse italia nessun paese nato interverrebbe oppure caso atomica conta	0
faggio ninabecks bella idea usciamo nato aspettiamo russia invade accoglieremo fiori canzoni persa completamente ragione popolo sfigati senza vergogna	2
gennarosenatore diesira pbiagiola gianzdav gianni elia nicolabellu alessioparodi gvergnasco posso capire senso chiedere uscita italia nato perch russia invade ucraina capisco s tco hljgzbopmt	2
blondman abbattere zar comporta azione militare suolo ucraino poi russo storicamente nessuno invade successo russia	0

Figura 11: Test su dataset non etichettato

Altre volte, si può notare che i tweet in origine presentavano molti hashtag e/o citazioni ad altri utenti, per cui il contenuto effettivo può essere compreso con maggiore difficoltà. In aggiunta, soprattutto quando si hanno frasi lunghe, il sentiment si chiarifica solo nella parte finale del tweet (Figura 12), per cui, anche a causa della lunghezza massima impostata inizialmente (11 caratteri), il modello riscontra difficoltà nell'assegnare la classi giusta, optando spesso per la neutralità.

russia ucraina intelligence tedesca stessa prima escludeva invasione poi prevedeva ucraina caduta poche settimane adesso cerca fornire scuse mancata fornitura leopard perdendo scopo mandare tank solo disprezzo s tco cn uj ajv	1
---	---

Figura 12: Esempio di ambiguità in un tweet

Per concludere, appurata la presenza di predizioni non accurate, sono stati contati gli elementi di ogni classe e sono stati ottenuti i seguenti risultati:

	tweets
label	
0	7310
1	7137
2	1102

Come prevedibile, il sentiment generale è soprattutto negativo o (7310 tweet) neutro (7137 tweet), mentre i tweet positivi sono sensibilmente di meno (1102 in totale). Ciò è dovuto soprattutto al fatto che su Twitter vengono pubblicate notizie di cronaca (recepite come neutre), mentre sono gli utenti che solitamente scrivono contenuti con note più negative.

## **Conclusioni e sviluppi futuri**

Questo progetto ha rappresentato per noi un modo per approfondire diversi aspetti di particolare rilevanza nell'ambito del Natural Language Processing, come la conoscenza della rete BERT e la Sentiment Analysis. Vedere come questo tipo di tecniche e tecnologie vengano applicate in ambiti che riguardano temi di stretta attualità, come la guerra in Ucraina, ci ha fatto veramente percepire l'importanza delle tematiche studiate.

Nonostante il comportamento non ideale dei modelli addestrati, ci riteniamo soddisfatti dei risultati raggiunti; infatti, se teniamo in considerazione il fatto che il dataset di training non era pensato per quella che è stata poi l'applicazione finale del modello prodotto, si potrebbe dire che i risultati ottenuti sono più che soddisfacenti.

In futuro, volendo andare ulteriormente a migliorare le performance dell'algoritmo, si potrebbe pensare ad allestire un dataset contenente dati più specifici e maggiormente correlati a quelli di test (tweet inerenti al conflitto russo-ucraino) e ad eseguire un preprocessing più puntuale, in modo da escludere parti non utili del contenuto.