Big Data: Fact-Checking

Alessio Marinucci Riccardo Felici

Abstract

La disinformazione online è diventata una delle principali preoccupazioni degli ultimi anni ed è stata ulteriormente enfatizzata durante la pandemia COVID-19. Le piattaforme dei social media, come Twitter (X) o Facebook, possono essere strumenti di disinformazione online. Per comprendere meglio la diffusione di queste fake-news, si analizzano le correlazioni tra le seguenti caratteristiche testuali dei tweets: sentimento, pregiudizio politico e veridicità. Vengono addestrati diversi classificatori basati su trasformatori per rilevare queste caratteristiche testuali e identificare potenziali correlazioni utilizzando le distribuzioni condizionali delle etichette. I nostri risultati mostrano come i tweets rispecchiano l'andamento della politica mondiale e soprattutto quelli più fuorvianti potrebbero avere impatti negativi sull'opinione pubblica.

Introduzione

Con l'aumento della quantità di informazioni condivise online, siamo inclini ad affrontare una maggiore disinformazione sul web. Alcuni dei principali esempi di attività che possono compromettere la qualità delle informazioni, riguardano principalmente attività politiche o del mondo sanitario che generano un elevato traffico sui principali social-networks. Considerando che le "**fake-news**" tendono a diffondersi più velocemente e più ampiamente della verità, i ricercatori hanno iniziato a sviluppare fact-checker per migliorare la loro capacità di verificare le informazioni. La necessità di tale tecnologia è stata ancora più evidente con la recente pandemia di COVID-19, con la disinformazione condivisa abbondantemente online. Mentre alcune ricerche si concentrano sull'individuazione della disinformazione, in questo lavoro viene posta l'attenzione su una migliore comprensione del discorso online intorno alla **sentiment analysis** e al **political bias** che vanno ad influenzare la

qualità dell'informazione. Si esplorano le relazioni tra sentimento, pregiudizio politico e veridicità, sfruttando un set di dati per ogni caratteristica testuale. Il seguito dell'elaborato è diviso come segue.
Nel Capitolo 1 vengono utilizzati due set di dati per addestrare i modelli che rilevano il sentimento e la parzialità politica, e si usano questi modelli sugli altri set di dati per studiare in dettaglio le loro interazioni.
Successivamente nel Capitolo 2 viene analizzato il codice eseguito per addestrare il modello con la stampa dei parametri che caratterizzano l'esecuzione del training loop.
Nel Capitolo 3 si pone l'attenzione sulla fase di inferenza in cui un modello già addestrato viene utilizzato per fare previsioni o prendere decisioni basate su nuovi dati che non ha mai visto prima. A differenza della fase di addestramento, durante l'inferenza, i pesi del modello sono fissi e non vengono aggiornati.
Infine nel Capitolo 4 vengono mostrati i risultati più significativi dove si vede che i pregiudizi politici possono giocare un ruolo importante nella classificazione del sentimento o delle emozioni che sono utilizzati da persone che condividono contenuti potenzialmente fuorvianti.
I nostri risultati possono essere riprodotti utilizzando il codice disponibile su https://github.com/DrRicky31/Fact-Checking-on-Twitter.
Di seguito si può osservare nella Figura 1 la pipeline dell'architettura utilizzata nel progetto, con la definizione di tutte le componenti.

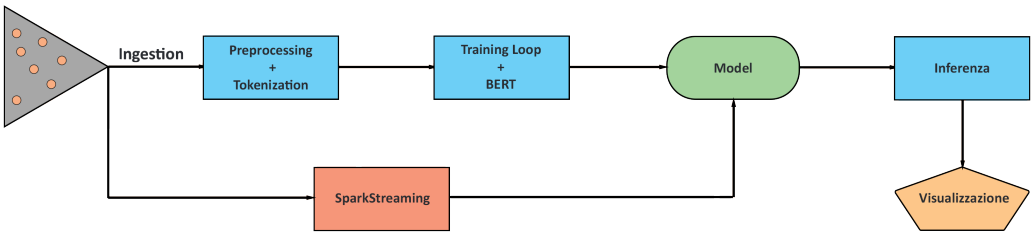


Figure 1: Pipeline dell'architettura utilizzata

1 Datasets

La disinformazione online ha spinto a studiare la diffusione nei siti web dei social media, come Twitter. Molti dataset sono stati costruiti per annotare le caratteristiche testuali dei tweets durante la pandemia. In questo lavoro, sono stati selezionati due diversi dataset di base, ognuno dei quali consente l'addestramento di un modello per il rilevamento di una caratteristica testuale: **COVIDSenti** (Sentiment) e **Russian Troll** (Political Bias).

- **COVIDSenti (Sentiment)**
  - **Descrizione**: Dataset con etichette per il sentiment nei tweet relativi a COVID.
  - **Periodo di Raccolta**: febbraio 2020 - marzo 2020
  - **Parole Chiave**: "coronavirus", "Corona Outbreak"
  - **Annotazioni Totali**: 90.000
  - **Sorgente**: https://github.com/usmaann/COVIDSenti
- **Russian Troll (Political Bias)**
  - **Descrizione**: Dataset contenente 2,9 milioni di tweet da account associati alla Internet Research Agency.
  - **Periodo di Raccolta**: febbraio 2012 - maggio 2018
  - **Scopo**: Interferenza durante le elezioni presidenziali statunitensi del 2016.
  - **Categorie di Etichettatura**:
    - "Right Troll"
    - "Left Troll"
    - "Fearmonger"
    - "HashtagGamer"
    - "NewsFeed"
  - **Focus di Questo Lavoro**: "Right Troll" e "Left Troll", con le restanti etichette raggruppate come "Other".
  - **Attributi del Dataset**:
    - external_author_id

        * author
        * content
        * region
        * language
        * publish_date
        * harvested_date
        * following
        * followers
        * updates
        * post_type
        * account_type
        * new_june_2018
        * retweet
        * account_category
    - **Sorgente**: https://www.kaggle.com/datasets/fivethirtyeight/russian-troll-tweets

I due set di dati principali vengono utilizzati anche per la valutazione della correlazione.

2 Training

2.1 Pre-processing

La **normalizzazione** del testo è un passaggio cruciale nel pre-processing dei dati testuali, specialmente quando si lavora con i social media, dove i contenuti possono essere molto variegati e contenere elementi di disturbo come link, caratteri speciali, e spazi extra. Il pre-processamento mira a pulire e standardizzare i tweets per renderli più uniformi e pronti per analisi successive. Prima di procedere con la normalizzazione, il codice filtra la lista dei tweets per assicurarsi che tutti gli elementi siano stringhe. Questo è importante per evitare errori durante la manipolazione del testo. Il codice itera su ogni tweet nella lista originale e applica una serie di trasformazioni:

- Sostituzione dei caratteri HTML.
- Rimozione di caratteri speciali e non breaking space.
- Rimozione di URL.
- Rimozione di Spazi Extra.

Alla fine, la funzione restituisce la lista dei tweets normalizzati.

```
1  import re
2  def normalize_text(tweets):
3      # Filtra solo i tweet che sono di tipo stringa
4      tweets = [text for text in tweets if isinstance(text
           , str)]
5      normalized_tweets = []
6      for text in tweets:
7          text = text.replace('&amp;', '&')
8          text = text.replace('\xa0', '')
9          text = re.sub(r'http\S+', '', text)
10         text = " ".join(text.split())
11         normalized_tweets.append(text)
12     return normalized_tweets
```

Listing 1: Pre-processing

2.2 Load Data

Si verifica inizialmente la disponibilità di una **GPU CUDA** per l'esecuzione del modello, in quanto estremamente ottimizzata, rispetto alla CPU, per l'esecuzione di elaborazioni parallele massicce e per l'ottimizzazione con l'utilizzo dei più popolari framework di DeepLearning. Viene impostato di conseguenza il dispositivo di esecuzione tra CPU e GPU. Successivamente, carica i dati dai file CSV presenti nella directory specificata, utilizzando la libreria pandas per gestire i dati in forma tabellare.
I tweets vengono estratti dai dataframe caricati e suddivisi in **Training Set** e **Test Set**. Se è necessario, il testo dei tweets viene normalizzato per uniformare la formattazione e migliorare la coerenza dei dati. Le etichette vengono assegnate alle classi di sentiment (positivo, neutro, negativo) in base alla colonna 'label'.
Viene implementato il codice utilizzato per calcolare i pesi delle classi in modo che durante l'addestramento del modello, le classi meno rappresentate ricevano un peso maggiore e viceversa. Questo aiuta a mitigare il problema del class imbalance, dove alcune classi potrebbero essere sotto-rappresentate rispetto ad altre, influenzando negativamente le prestazioni del modello.
Per processare efficacemente il testo dei tweets, il codice utilizza un **tokenizer** pre-addestrato per convertire i testi in input in sequenze di token numerici comprensibili per il modello di machine learning. Inoltre, viene impostata una lunghezza massima per le sequenze di token per gestire efficientemente la memoria durante l'addestramento.
I dati tokenizzati vengono quindi convertiti in **tensori *PyTorch***, necessari per l'addestramento dei modelli in PyTorch. Ciò include la creazione di DataLoader, che consente di iterare sui dati in batch durante l'addestramento e la valutazione del modello.
Infine, il codice gestisce i pesi delle classi per affrontare eventuali sbilanciamenti nei dati, calcolando i pesi basati sulla distribuzione delle etichette di addestramento. Il codice stampa l'output dei tweets di test per confermare la corretta lettura e tokenizzazione dei dati. Di seguito viene indicata la sezione di codice relativa al processo di tokenizzazione dei dati, eseguita per migliorare l'efficienza della memoria RAM.

```
# Tokenizza i dati in batch piu' piccoli per risparmiare memoria
def batch_tokenize(texts, tokenizer, batch_size=100, max_length=MAX_LEN):
    input_ids = []
    token_type_ids = []
    attention_masks = []

    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]
        tokenized_batch = tokenizer(batch_texts, max_length=max_length, padding='max_length', truncation=True)

        input_ids.extend(tokenized_batch['input_ids'])
        token_type_ids.extend(tokenized_batch['token_type_ids'])
        attention_masks.extend(tokenized_batch['attention_mask'])

        # Rilascia esplicitamente la memoria del batch processato
        del tokenized_batch

    return input_ids, token_type_ids, attention_masks

# Tokenizza i dati di addestramento e di test in batch
train_input_ids, train_token_type_ids, train_attention_mask = batch_tokenize(tw_train, tokenizer)
test_input_ids, test_token_type_ids, test_attention_mask = batch_tokenize(tw_test, tokenizer)
```

Listing 2: Batch Tokenizer

Il codice presentato ha l'obiettivo di tokenizzare i dati in batch più piccoli per risparmiare memoria RAM durante l'elaborazione. All'interno della funzione, i dati vengono suddivisi in batch più piccoli della dimensione specificata.
Un passaggio cruciale per risparmiare memoria è l'esplicito rilascio della memoria del batch processato, che rimuove il batch tokenizzato dalla memoria una volta che i suoi dati sono stati memorizzati nelle liste di output. Questo processo continua fino a quando tutti i testi sono stati tokenizzati.
L'approccio seguente è essenziale quando si lavora con grandi dataset, poiché consente di gestire i dati in modo più efficiente, riducendo il rischio di esaurire la memoria e migliorando le prestazioni complessive del sistema.
In conclusione le operazioni svolte in questa sezione sono estremamette adattate al dataset di riferimento, al fine di gestire, nel modo più corretto ed adatto all'obiettivo, l'etichettatura delle varie caratteristiche dei dataset.

2.3 Model

Nella sezione seguente, viene definita la classe *CovidTwitterBertClassifier* che eredita da *nn.Module*. Nel suo costruttore, il modello **BERT** pre-addestrato viene caricato utilizzando il modello **digitalepidemiologylab/covid-twitter-bert-v2**. Il modello è una versione migliorata del modello BERT (Bidirectional Encoder Representations from Transformers) adattato specificamente per analizzare il linguaggio relativo al COVID-19 su Twitter. Questo modello è stato sviluppato per comprendere e classificare meglio i tweets riguardanti il COVID-19, grazie a un fine-tuning su un corpus di tweets a tema pandemico. Il layer finale del BERT viene modificato per adattarsi al numero di classi di output specificato. Viene anche inizializzato un layer *Sigmoid*, ma attualmente non è utilizzato nel metodo forward.
Il metodo forward accetta input tokenizzati, li passa attraverso il modello BERT e restituisce i **logits** dell'output del modello, che rappresentano i valori grezzi che indicano la forza relativa di ciascuna classe predetta dal modello per un dato input.
Un'istanza del modello *CovidTwitterBertClassifier* viene creata con il numero di classi impostato su tre, per classificare il sentiment o il bias politico dei tweets.
Il modello viene trasferito sulla GPU per sfruttare la sua potenza di calcolo parallelo. In aggiunta, viene utilizzato un ottimizzatore chiamato **AdamW** per aggiornare i parametri del modello durante l'addestramento. **AdamW** è una variante dell'ottimizzatore Adam, ottimizzato per il DeepLearning. Utilizza un tasso di apprendimento di 1e-5 e applica un peso di decadimento di 0.01 per controllare la regolarizzazione dei parametri.
Per monitorare e adattare il tasso di apprendimento durante l'addestramento, è configurato uno scheduler chiamato *ReduceLROnPlateau*. Questo scheduler riduce il tasso di apprendimento se la loss non migliora dopo un certo numero di epoche, definito come "*patience=4*". Il fattore di riduzione è impostato su 0.3, il che significa che il tasso di apprendimento viene moltiplicato per 0.3 ogni volta che il criterio di pazienza non viene soddisfatto, permettendo al modello di convergere più stabilmente durante l'addestramento.
La loss function criterion è impostata su *nn.CrossEntropyLoss* con un peso specificato (weights), che viene utilizzato per affrontare eventuali sbilanci di classe nei dati.
I modelli basati sui trasformatori hanno ampiamente contribuito al progresso di molte attività di elaborazione del linguaggio naturale (NLP), tra cui la

traduzione automatica, la risposta alle domande e la classificazione dei testi. In particolare, BERT ha superato altri metodi, come TF-IDF o le reti neurali ricorrenti, fornendo un modello pre-addestrato che può essere sintonizzato per compiti specifici. Di seguito è indicata la sezione relativa di codice che svolge questo processo di addestramento.

```
class CovidTwitterBertClassifier(nn.Module):
    def __init__(self, n_classes):
        super().__init__()
        self.n_classes = n_classes
        self.bert = BertForPreTraining.from_pretrained('
           digitalepidemiologylab/covid-twitter-bert-v2'
           )
        self.bert.cls.seq_relationship = nn.Linear(1024,
            n_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_ids, token_type_ids,
       input_mask):
        outputs = self.bert(input_ids = input_ids,
           token_type_ids = token_type_ids,
           attention_mask = input_mask)
        logits = outputs[1]
        return logits

model = CovidTwitterBertClassifier(3) # 5 for emotion
   and 3 for sentiment and political bias
model.to(device)
#optimizer_grouped_parameters
optimizer = AdamW(model.parameters(),
                  lr=1e-5,
                  #lr=3e-5,
                  weight_decay = 0.01)

scheduler = ReduceLROnPlateau(optimizer, patience=4,
   factor=0.3)
criterion = nn.CrossEntropyLoss(weight = weights)
```

Listing 3: BertClassifier

2.4 Training Loop

Questa sezione descrive il processo di addestramento e valutazione di un modello di machine learning utilizzando PyTorch. Il modello viene addestrato su un set di dati di training e valutato su un set di dati di test per un numero specifico di epoche. Durante questo processo, vengono monitorate metriche di performance come la **loss, l'accuracy** e il punteggio **F1 Score**. Il miglior stato del modello viene salvato in base alla migliore performance ottenuta.
Inizialmente, vengono impostate alcune variabili di controllo, come il numero totale di epoche e *best_loss* inizializzato a un valore molto alto per tracciare la loss minima.
Il ciclo di addestramento parte con un loop su ciascuna epoca, durante il quale il modello è impostato in modalità di addestramento (*model.train()*). I dati di addestramento sono iterati per batch, con gli input del modello trasferiti sulla GPU per l'elaborazione. Ogni batch aggiorna i gradienti azzerati con *optimizer.zero_grad()*, esegue il forward pass per ottenere i logits e il calcolo della loss con criterion.
Terminato il passaggio di addestramento, il modello passa in modalità di valutazione (*model.eval()*). Qui, i dati di test sono iterati per batch, i logits calcolati senza aggiornare i gradienti tramite *torch.no_grad()*, e la loss di valutazione è calcolata e accumulata.
Alla fine di ogni epoca, lo scheduler è aggiornato in base alla media della loss di valutazione. Vengono calcolate metriche di performance come l'accuratezza e il punteggio F1 utilizzando i logits predetti e le etichette vere.
Se il punteggio F1 corrente supera quello precedentemente migliorato, i migliori valori di loss, F1, accuratezza e lo stato del modello (*state_dict*) sono aggiornati. I risultati dell'epoca corrente, inclusi loss media, punteggio F1 e accuratezza, vengono stampati a schermo. Nella Figura 2 sono mostrati i risultati di un primo addestramento del modello usando il dataset *Russian Troll Tweets*.

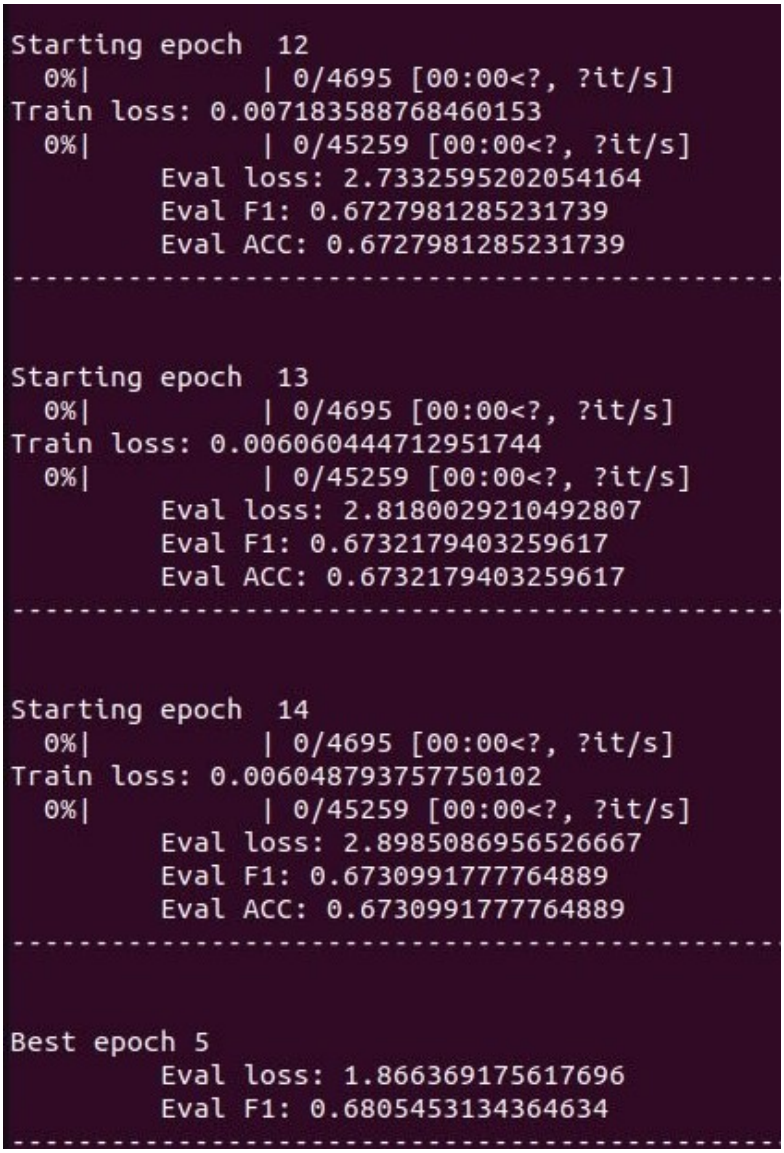


Figure 2: Risultati di Training ottenuti su 778K tweets

3 Inference

Inizialmente, il codice carica il modello precedentemente addestrato nel Capitolo 2.4 da file locale. Il modello viene quindi impostato in modalità valutazione (*model.eval()*), stabilendo che il modello sarà utilizzato esclusivamente per eseguire **inferenza**, ovvero per fare predizioni su dati di input senza apportare modifiche ai suoi parametri tramite il backpropagation.
Successivamente, il codice itera attraverso i batch di dati di test. All'interno del ciclo, ogni batch viene elaborato utilizzando *torch.no_grad()*, il che significa che non vengono calcolati né utilizzati i gradienti per ottimizzare i parametri del modello. Il modello predice gli output desiderati (logits) dai dati di input del batch. Questi logits rappresentano i punteggi di classe non normalizzati per ciascuna possibile classe di output.
Dopo aver ottenuto i logits per un batch, vengono estratti i risultati e memorizzati in tre liste: *logits_full, ground_truth_full, e ids_full*. Queste liste conservano rispettivamente i logits predetti dal modello, le etichette reali (ground truth) e gli identificatori univoci associati a ciascun dato di test.
Una volta completata l'iterazione attraverso tutti i batch di dati di test, il codice calcola una serie di metriche per valutare le prestazioni del modello. Infine, i risultati vengono stampati a schermo per fornire una panoramica delle prestazioni del modello durante la valutazione e vengono salvati su disco in un file CSV.
Nel resto del Capitolo è presentata l'implementazione tramite **SparkStreaming** per poter raccogliere i dati non solo in maniera batch offline, ma tramite un flusso di dati streaming in tempo reale.

3.1 SparkStreaming

Per la raccolta dei dati in input durante la fase di inferenza si è fatto uso del framework SparkStreaming per simulare lo streaming in tempo reale di un flusso di dati proveniente da fonti esterne. Nel nostro lavoro lo streaming è simulato da dati presenti in una directory locale, ma in un applicativo reale viene fatto uso del framework **Kafka** per costruire pipeline di dati in tempo reale.
I dati sono inviati al client attraverso una connessione TCP, simulando così un ambiente di streaming dati, come quello utilizzato da SparkStreaming.
Viene inizialmente creata una **SparkSession**, questa sessione è fondamentale per utilizzare le funzionalità di Spark, incluso Spark Streaming.

Un DataFrame di streaming viene creato leggendo i dati dal server socket, simulando così un flusso di dati in tempo reale che verranno poi elaborati da Spark. I dati ricevuti dal flusso vengono convertiti dal formato JSON, permettendo di lavorare con dati strutturati. La funzione *process_data* viene applicata a ciascun batch di dati ricevuti attraverso una query di scrittura di streaming. Questa parte del codice è cruciale per l'elaborazione in tempo reale dei dati in arrivo, poiché *foreachBatch* consente di applicare la funzione *process_data* a ogni batch di dati ricevuti dal flusso di streaming.
Infine, *query.awaitTermination()* mantiene l'applicazione in esecuzione finché il flusso di dati non viene terminato.
Nel seguito è presentato il codice che implementa la SparkSession per raccogliere i dati streaming da una socket locale e lanciare il processing dei dati per ogni batch.

```
# Creazione della sessione Spark
spark = SparkSession.builder.appName("COVIDTweetStream")
    .getOrCreate()
# Schema per il JSON in arrivo
schema = StructType([
    StructField("content", StringType(), True),
    StructField("account_category", StringType(), True)
])
def process_data(df):
    # Funzione per il processing dei dati
# Creazione di un DataFrame di streaming da una socket
df = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()
# Applica la funzione di processazione ai dati in
    streaming
query = json_df.writeStream.foreachBatch(lambda batch_df
    , batch_id: process_data(batch_df)).start()
query.awaitTermination()
```

Listing 4: Data Ingestion tramite SparkStreaming

Infine vengono descritti i risultati ottenuti lanciando l'esecuzione di Spark-Streaming per ottenere i dati in tempo reale. I parametri che vengono registrati sono gli stessi delle esecuzioni tramite raccolta dei dati offline in locale, ovvero Eval Loss e Eval F1. Anche i tempi di esecuzione sono comparabili, in quanto SparkStreaming si occupa di convertire il fusso dati streaming in microbatch (come si vede in Figura 3) che sono processati nel medesimo modo dal modello di inferenza, producendo stesso output in tempi equiparabili.

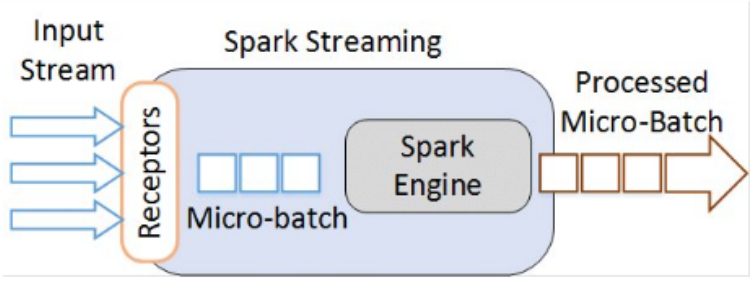


Figure 3: Fasi di processamento di Spark Streaming

4 Risultati Ottenuti

In questo capitolo vengono mostrati i risultati ottenuti dall'inferenza sul modello, prendendo i tweets sia in tempo reale, usando spark streaming, sia da repository locale, raccogliendo i dati da entrambi i dataset considerati.

4.1 Strumenti utilizzati

Per svolgere questo progetto è stato necessario l'utilizzo di una macchina remota fornita dalla piattaforma di ricerca e sperimentazione **CloudLab**. La macchina utilizzata possiede le seguenti caratteristiche:

- CPU: Two 16-core AMD 7302 3.00 GHz
- RAM: 128GB 3200MT/s
- GPU: NVIDIA 24GB Ampere A30

4.2 Analisi e Prestazioni

In questo paragrafo vengono illustrati i risultati migliori ottenuti dall'esecuzione, con l'ottimizzazione dei parametri, utilizzando i due dataset citati nel Capitolo 1. Gli addestramenti sono stati effettuati eseguendo un Training Loop su **15 epoche** e registrando la migliore epoca registrata. Si riportano i seguenti addestramenti effettuati:

- Addestramento su dataset ***Russian Troll Tweets***: l'addestramento del modello su un dataset di 743.900 tweets ha prodotto i seguenti parametri di valutazione:
  - Eval loss: 0.623
  - Eval F1: 0.873
- Addestramento su dataset ***COVIDSenti***: l'addestramento del modello su un dataset di 90.000 tweets ha prodotto i seguenti parametri di valutazione:
  - Eval loss: 0.517
  - Eval F1: 0.945

Nel seguito sono presentati i LineChart che mostrano i risultati ottenuti dall'addestramento dei modelli.

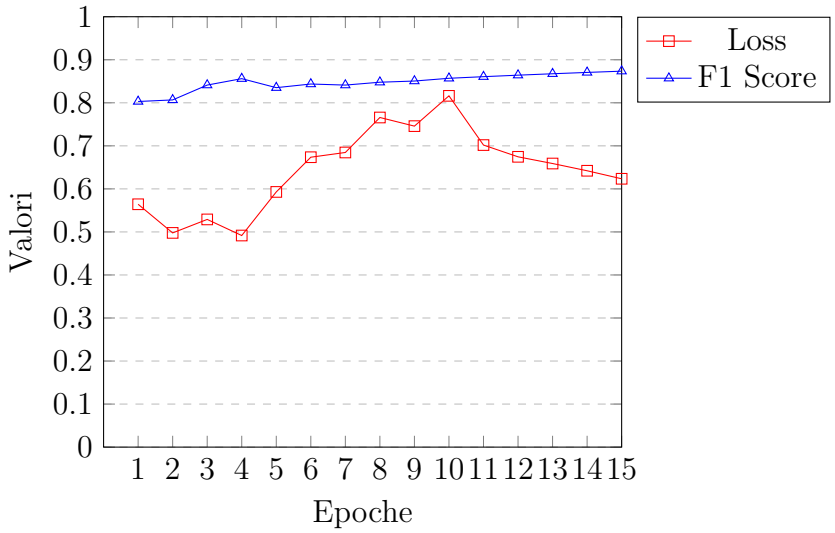


Figure 4: Russian Troll. Grafico dei valori di Loss e F1 Score per 15 epoche

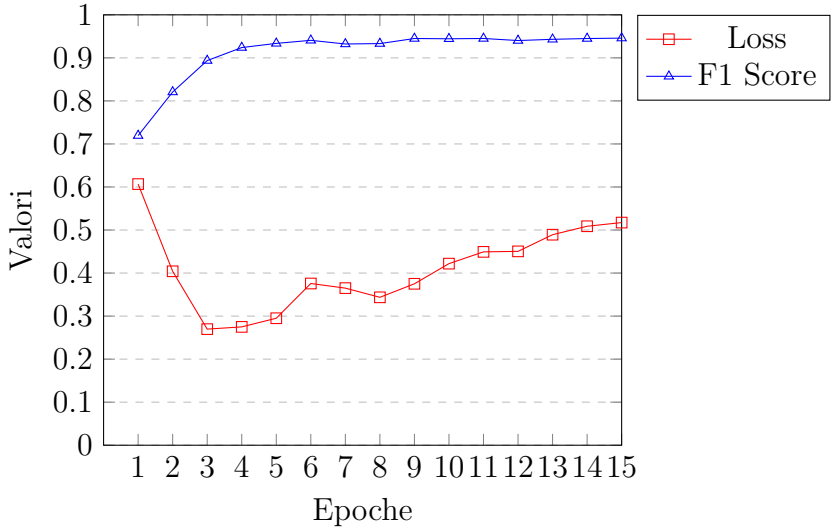
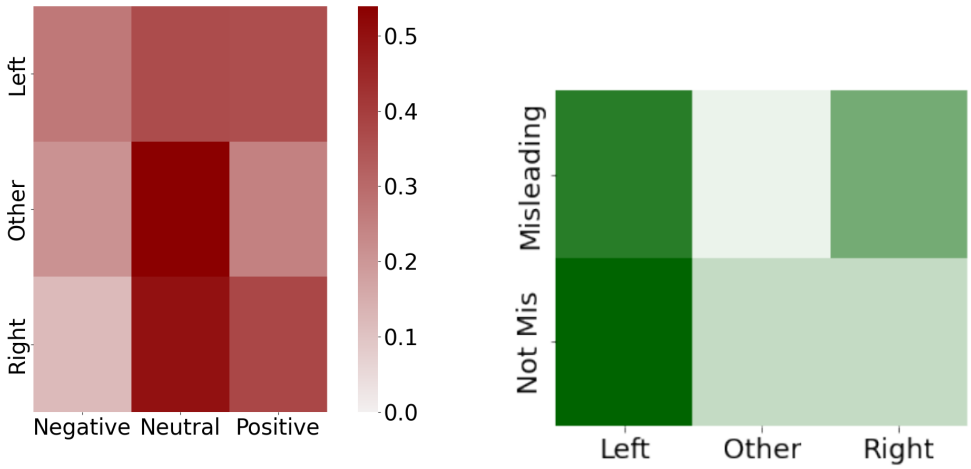


Figure 5: COVIDsenti. Grafico dei valori di Loss e F1 Score per 15 epoche

I risultati ottenuti mostrano come sentimenti ed emozioni specifiche non sono fortemente correlati a un pregiudizio politico o all'altro, come mostrato in Figura 6a. Tuttavia, come riportato in Figura 6b, i tweet potenzialmente fuorvianti non risultano essere legati a un particolare pregiudizio politico, mentre i tweet non fuorvianti hanno maggiore probabilità di essere condivisi con pregiudizi politici di sinistra.
Utilizzando altri tipi di dataset supervisionati, che permettono di etichettare ciascun tweet in relazione a determinati argomenti controversi relativi alle disposizioni COVID-19, come ad esempio l'utilizzo delle mascherine, disposizioni sulla chiusura scuole o sul restare a casa, è possibile analizzare il riscontro a determinati temi in base al political bias di ciasun utente.
Analizzando la correlazione tra la caratteristica sentiment e le altre caratteristiche è possibile notare come le persone contrarie agli argomenti citati tendono a condividere un sentiment più negativo. Tuttavia, la distribuzione del sentiment è la stessa sia per le tendenze di destra che di sinistra.
Nel complesso, pochissimi tweet condividono un sentiment positivo in questo dataset, suggerendo che le persone su Twitter sono più interessate a etichettare i tweet negativi.
La Tabella 1 offre una panoramica sulle reazioni degli utenti di Twitter a eventi e dibattiti politici, nonché sulle loro inclinazioni politiche. Mostra i risultati ottenuti dal processo di classificazione sul dataset *Russian Troll Tweets*. Questi dati evidenziano la varietà di opinioni espresse online e l'importanza di comprendere le dinamiche della polarizzazione politica e dell'influenza sociale nel contesto delle piattaforme di social media.

ids	content	political_bias
8991	Japan: Thousands protest military legislation changes in Tokyo	1
300492	Will you stop collecting my personal data? #DemnDebate	2
572593	OMG All people should become vegetarian now! #GoVegan #KochFarms #Turkey #foodpoisoning	0
22579	OMG Capitalism is #failed #KochFarms#Turkey #NYC #USDA	0
2532	#DemnDebate Sanders with most to prove as Dems gather for 2nd debate	2
98508	I think Hillary should tell us about her E-mails, and I mean the truth. But #DemnDebate and truth are opposites, so...	0
15506	even if the person doesn't understand politics at all, he knows there is some BS on TV when #DemnDebate	0
78467	#DemnDebate Giant Bernie Sanders costume draws onlookers at debate site	1
45448	Paris attacks likely to dominate #DemnDebate	0
9445	#DemnDebate #DemDebate Bernie is too weak to end corruption	2

Table 1: Tabella con output della classificazione



(a) HeatMap del sentiment in base al Political Bias

(b) HeatMap della Veracity

Figure 6: Confronto tra HeatMap

Conclusioni

In questo lavoro, vengono analizzate le correlazioni tra tre caratteristiche testuali: sentimento, pregiudizio politico e veridicità. Si sfruttano set di dati rilevanti per addestrare due modelli per prevedere sentimenti e pregiudizi politici sui tweet relativi a COVID-19 e alle Elezioni Statunitensi. Questi modelli consentono di analizzare la distribuzione condizionale delle diverse etichette per comprendere meglio il comportamento online. I risultati principali includono che gli argomenti relativi alle normative relative al COVID-19, che, essendo molto controversi, generano molti sentimenti negativi. Il pregiudizio politico degli utenti su questi argomenti ha inoltre delineato la posizione politica nel dibattito.