Progetto per Tirocinio formativo Curriculare

Gmail Sentiment Analysis

Autore	Matricola	E-mail
Andrea Ruggiero	0512114732	a.ruggiero@datagrafservizi.com a.ruggiero150@studenti.unisa.it

Sommario

Introduzione

Scopo del progetto

Con il presente progetto si intende sviluppare un **modello di predizione del sentimento delle e-mail** recapitate presso caselle di posta elettronica con dominio **Gmail** o domini ad esso associati (ad esempio *studenti.unisa.it*).
L'obiettivo principale è analizzare in maniera automatica il contenuto dei messaggi e classificarne il **sentimento** in base al grado di criticità che la comunicazione può rappresentare per l'ecosistema aziendale. Non tutte le e-mail, infatti, hanno lo stesso livello di rilevanza: una semplice comunicazione informativa può essere considerata neutra o a basso impatto, mentre una mail di **avviso di disservizio** richiede un'attenzione maggiore, poiché può segnalare problematiche che influiscono direttamente sulla continuità operativa. Allo stesso modo, anche un messaggio che notifica l'**avvenuto cambio password** deve essere trattato come potenzialmente critico: nel caso in cui si tratti di un'azione malevola condotta da soggetti esterni non autorizzati, tale e-mail può rappresentare un serio rischio per la sicurezza dei sistemi informativi aziendali. Per raggiungere questo scopo, il progetto prevede non solo la costruzione del modello di **sentiment analysis**, ma anche la realizzazione di una **piattaforma applicativa** semplice e intuitiva. L'utente potrà accedere tramite il proprio account personale e visualizzare la casella di posta direttamente dall'interfaccia. Le e-mail verranno evidenziate in base al livello di criticità individuato dal modello: i messaggi con maggiore impatto negativo o sospetto saranno mostrati in **rosso**, così da catturare immediatamente l'attenzione. Inoltre, ogni e-mail sarà accompagnata da un **indicatore del grado di criticità**, che permetterà di comprendere rapidamente l'importanza e la priorità dell'intervento. In questo modo, il progetto intende fornire uno strumento innovativo per la **gestione proattiva delle comunicazioni** in ambito aziendale, supportando gli utenti nell'individuazione tempestiva di potenziali rischi e contribuendo a rafforzare la sicurezza complessiva dell'infrastruttura digitale.

Definizione del problema

Contesto del problema

Negli ultimi anni la posta elettronica è diventata uno degli strumenti di comunicazione più diffusi, sia in ambito privato che aziendale. Tuttavia, l'aumento esponenziale del numero di e-mail ricevute quotidianamente comporta nuove sfide, legate non solo alla gestione del volume delle informazioni, ma anche alla capacità di identificare in modo tempestivo i messaggi che rappresentano un rischio per la sicurezza informatica o che richiedono un'azione immediata. In particolare, le organizzazioni si trovano a dover fronteggiare problematiche come:

- **Phishing e social engineering**, ossia e-mail apparentemente legittime che cercano di carpire credenziali o dati sensibili;
- **Comunicazioni malevole o sospette**, che possono includere notifiche di cambio password non autorizzate, tentativi di accesso fraudolento o messaggi di disservizi provenienti da fonti non attendibili;
- **Sovraccarico informativo**, che riduce la capacità degli utenti di distinguere velocemente i messaggi rilevanti da quelli secondari.

Queste criticità hanno un impatto diretto sulla **sicurezza aziendale**, in quanto un singolo messaggio non gestito correttamente può aprire la strada a violazioni gravi, con conseguenze economiche e reputazionali. Inoltre, la classificazione manuale dei messaggi da parte degli utenti non è sostenibile, né efficiente, soprattutto in realtà complesse con un flusso costante e massivo di comunicazioni. In questo scenario, l'applicazione di tecniche di **analisi del sentimento (sentiment analysis)** ai messaggi e-mail rappresenta una possibile soluzione innovativa: permette di assegnare un livello di criticità ai testi ricevuti e di supportare l'utente nell'individuazione rapida delle comunicazioni più delicate.

Specifiche PEAS

Performance

Accuratezza delle previsioni errore medio basso (MAE e MSE ridotti).

Capacità di spiegare la varianza R^2 vicino a 1.

Robustezza mantenere performance simili su dati di test reali non visti.

Efficienza tempi di inferenza ridotti per l'analisi in tempo reale delle e-mail.

Stabilità il modello dopo il training iniziale rimane uguale fino ad un re-train periodico

Environment

Parzialmente osservabile: il modello non vede l'"intenzione reale" dietro al testo, ma solo la sequenza di parole che compongono oggetto e testo della mail.

Agente singolo: c'è solo il modello che apprende e produce output, non ci sono altri agenti con cui deve interagire.

Stocastico: l'output non è perfettamente deterministico, poiché testi simili possono avere sfumature diverse ed essere trattati in maniera differente dal modello.

Continuo: gli output possibili appartengono a un intervallo numerico infinito ovvero una scala di criticità del sentiment che va da 0 a 1.

Statico: il testo non cambia mentre il modello elabora. Non ci sono eventi esterni che modificano l'osservazione durante l'analisi.

Episodico: ogni e-mail (o testo) viene analizzato indipendentemente dalle altre. Non serve tener conto delle precedenti per valutare il sentiment corrente

Actuators

L'agente dovrà mostrare in output le ultime 25 e-mail della casella postale dell'utente (che avrà la possibilità di aggiornare) ogni e-mail dovrà mostrare la percentuale di negatività, le e-mail che superano un dato livello di criticità saranno contrassegnate in rosso e l'utente avrà la possibilità di ordinare le e-mail in base al livello di criticità

Sensors

L'agente recupererà le e-mail dalla casella postale Gmail dell'utente tramite un servizio di Google dedicato

Definizione della soluzione

Valutazione delle soluzioni

In una fase iniziale di brainstorming era stata valutata la possibilità di adottare un **modello di classificazione** con tre classi principali: *positivo*, *neutro* e *negativo*. Tale approccio, tuttavia, per sua natura restituisce come risultato la **classe predetta** alla quale la frase viene assegnata, accompagnata da una percentuale di confidenza relativa alla decisione presa. Questo tipo di output non risulta perfettamente in linea con l'obiettivo del progetto: ciò che ci interessa, infatti, non è semplicemente etichettare un'e-mail come positiva o negativa, ma ottenere un **valore numerico che quantifichi il grado di negatività** del messaggio, così da poter determinare il livello di criticità e portare all'attenzione dell'utente le comunicazioni più delicate.

Per questo motivo, un approccio basato sulla **regressione** risulta maggiormente indicato: il modello fornisce un punteggio continuo che consente di ordinare i messaggi in base al loro livello di severità. A questo punteggio viene poi associata una **classe discreta**, derivata dal valore numerico (critico ≤ 0.40, mediamente critico ≤ 0.60, positivo > 0.60), così da suddividere i gradi di criticità in categorie facilmente interpretabili e utilizzabili anche in un futuro sviluppo grafico.
In questo modo si mantiene non solo la categorizzazione chiara dei messaggi, ma anche una **gerarchia interna alla classe discreta**, determinata dal punteggio continuo, che permette di distinguere i casi più o meno gravi all'interno della stessa fascia di criticità. Ad esempio, una mail di avviso di disservizio potrà ricevere un valore molto basso (critico), mentre una comunicazione neutra come una newsletter informativa sarà valutata positivamente (non rappresentando una criticità), lasciando all'utente un chiaro indicatore della priorità da assegnare alla lettura. Per quanto riguarda la costruzione del modello, si è optato per una strategia di **fine-tuning** a partire da un modello linguistico preesistente piuttosto che svilupparne uno da zero o utilizzare un modello generico senza alcuna personalizzazione. Le motivazioni principali sono le seguenti:

1. **Sviluppare un modello da zero** richiederebbe enormi quantità di dati annotati, risorse computazionali significative e tempi di addestramento molto elevati. Nel caso specifico delle e-mail, la disponibilità di dataset etichettati di grandi dimensioni è limitata, il che renderebbe questa strada poco praticabile.
2. **Utilizzare un modello già pronto senza fine-tuning** (ad esempio un modello di sentiment generico addestrato su recensioni di film o social media) potrebbe portare a prestazioni non ottimali. Il linguaggio usato nelle e-mail aziendali presenta caratteristiche particolari (terminologia tecnica, stile formale, lessico legato a procedure e sistemi), che differiscono notevolmente dai testi presenti nei dataset comunemente usati per l'analisi del sentiment.
3. **Il fine-tuning** rappresenta un compromesso ideale: sfrutta la conoscenza linguistica già appresa dal modello di base (che ha compreso strutture sintattiche e semantiche generali), ma la adatta al dominio specifico delle e-mail aziendali. In questo modo, con un numero relativamente ridotto di esempi annotati, è possibile ottenere un modello altamente performante e capace di cogliere le sfumature di criticità che emergono nei testi.

In conclusione, l'adozione di un **modello di regressione sottoposto a fine-tuning** appare la scelta più adeguata, in quanto permette di bilanciare efficienza, accuratezza e adattabilità al contesto, offrendo uno strumento realmente utile per la classificazione critica dei messaggi di posta elettronica.

Modello di base

Il modello di base utilizzato per il fine-tuning è **dbmdz/bert-base-italian-xxl-cased**, una variante di BERT specializzata nella lingua italiana, sviluppata dal gruppo dbmdz (Digital Humanities Group del Munich Center for Machine Learning). Si tratta di un modello Transformer Encoder pre-addestrato su un ampio corpus testuale in italiano (Wikipedia, OpenSubtitles, Gazzetta Ufficiale, giornali, ecc.), che permette di catturare le caratteristiche sintattiche e semantiche della lingua.

Le caratteristiche principali sono:

- Architettura: basata su BERT addestrato in due compiti principali: **Masked Language Modeling (MLM)** per predire la **parola mancante** in una frase e **Next Sentence Prediction (NSP)** per predire se una frase B segue logicamente una frase A. Grazie a questo sviluppa un'ottima capacità di predire la semantica delle parole, la relazioni tra frasi e parole e produrre **embedding contestuali** ad esempio la parola "positivo" in "il test è positivo" (medico implica negatività) non ha lo stesso significato di "positivo" in "il feedback è positivo". Caratteristiche fondamentali per distinguere correttamente il sentiment.
- Dimensione: XXL (24 strati, 1024 hidden size, 16 teste di attenzione, ~550M parametri)
- Cased: il modello distingue tra maiuscole e minuscole (utile per nomi propri, acronimi, ecc.)

Tokenizer

Insieme al modello, viene utilizzato il **tokenizer ad esso associato**.
Il tokenizer è responsabile della trasformazione del testo grezzo in una sequenza di **token numerici** che il modello può elaborare. Caratteristiche principali del tokenizer:

- **WordPiece Tokenizer**: suddivide le parole in sottoparole (subword) quando non presenti nel vocabolario. Questo riduce il problema delle *Out-of-Vocabulary words* (OOV). Esempio: la parola *"informatizzazione"* → ["informat", "##izzazione"]
- **Cased**: mantiene le differenze tra maiuscole e minuscole → "Roma" ≠ "roma".
- **Vocabolario**: circa **32000 token** (pre-addestrato sull'italiano).
- **Special token**:
    - [CLS]: indica l'inizio della sequenza (usato per la classificazione/regressione).
    - [SEP]: separa due frasi o indica la fine della sequenza.
    - [PAD]: padding per uniformare la lunghezza delle sequenze.
    - [MASK]: usato in pre-training per il compito di Masked Language Modeling.

Nel nostro caso specifico

```
MODEL = "dbmdz/bert-base-italian-xxl-cased"

tokenizer = AutoTokenizer.from_pretrained(MODEL)


def tokenize(batch):

    return tokenizer(batch['review_text'], padding='max_length', truncation=True,
max_length=512)


dataset = dataset.map(tokenize, batched=True)
```

Dopo aver caricato il tokenizer pre-addestrato associato al modello **dbmdz/bert-base-italian-xxl-cased**, applichiamo la funzione tokenize a tutto il dataset mediante il metodo map. Questo processo restituisce un nuovo dataset in cui, oltre alle colonne originali (il testo della recensione e l'etichetta), vengono aggiunte le colonne prodotte dalla tokenizzazione. In particolare, la funzione tokenize applica il tokenizer alla colonna contenente le recensioni degli utenti, trasformando ogni testo in una sequenza di token. Le operazioni svolte sono le seguenti:

- **Segmentazione in token**: ogni recensione viene suddivisa in parole e sottoparole secondo l'algoritmo WordPiece.
- **Inserimento di special token**: vengono aggiunti token come [CLS] (inizio sequenza) e [SEP] (separatore/fine sequenza).
- **Generazione di input_ids**: il testo tokenizzato viene convertito in un vettore di interi, dove a ciascun token (inclusi gli special token) corrisponde un identificativo numerico unico.
- **Generazione di attention_mask**: viene prodotto un vettore parallelo di 0 e 1 che indica quali token devono essere considerati (1 = token reale, 0 = padding). Dato che la lunghezza massima è fissata a **512 token** se una sequenza è più corta, il tokenizer aggiunge token di padding [PAD] fino a raggiungere 512. Viceversa, se una sequenza supera la lunghezza massima, i token eccedenti vengono troncati.

Flusso di predizione

La fase di predizione è suddivisa in vari step:

Lookup + somma dove ogni valore di **input_ids** viene trasformato tramite la **embedding matrix di BERT** in un **vettore denso** (di dimensione 1024 nel modello utilizzato). Per ogni token ottieni quindi tre vettori della stessa dimensione:

- **Token Embedding** che rappresenta il significato base del token.
- **Position Embedding** che rappresenta la posizione del token nella sequenza (0, 1, 2, … fino a 511).

- **Segment Embedding** che distingue tra frase A e frase B (aiutando a comprendere frasi complesse suddivise in più periodi)

I tre vettori vengono sommati elemento per elemento formando un vettore finale dove ogni elemento risulterà:

$$Embedding\ finale\ del\ Token = E_{Token} + E_{Posizione} + E_{Segmento}$$

Successivamente si passa agli strati del **Transformers Encoder** che ricevono in input per ogni token il vettore denso di dimensione d_{model} (1024 nel caso del modello utilizzato). Quindi se la frase ha *L token*, l'input al Transformer è una **matrice** di dimensione:

$$X \in R^{L \times d_{model}}$$

Multi-Head Self-Attention dove BERT applica **self-attention** a questi vettori attraverso determinati passi.

1. **Proiezioni Q, K, V**
   Ogni embedding x viene moltiplicato per tre matrici apprese W_Q, W_K, W_V:
   $$Q = x\,W_Q, \qquad K = x\,W_K, \qquad V = x\,W_V$$
   Quindi da ogni token ottieni tre nuovi vettori (Q, K, V) più piccoli di d_{model} perché ogni testa lavora in uno spazio ridotto (d_k).
   $$d_k = \frac{d_{model}}{n_{teste}}$$
2. **Prodotto scalare tra Q e K**
   Per ogni token, verrà confrontato il suo Q con i K di tutti gli altri token nella sequenza:
   $$score(i,j) = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}$$
   Questo calcola quanto il token i deve prestare attenzione al token j.
3. **Softmax sui pesi**
   I punteggi vengono normalizzati con una softmax, così diventano probabilità che sommano a 1.
   $$\alpha_{ij} = \mathrm{softmax}_j\big(score(i,j)\big)$$
4. **Combinazione con i valori V**
   Per ogni token, calcoli una combinazione pesata dei vettori V:
   $$z_i = \sum_j \alpha_{ij} V_j$$

Questo significa che il nuovo embedding di un dato token è una media pesata delle informazioni di tutti gli altri token.

Questo processo viene ripetuto in parallelo con più teste (Multi-Head). Ogni testa nel suo spazio ridotto d_k cattura relazioni diverse (es. sintattiche, semantiche, distanza a lungo raggio) e i risultati vengono concatenati e proiettati di nuovo in uno spazio di dimensione d_{model}.

$$n_{\text{heads}} \cdot d_k = d_{\text{model}}$$

Feed-Forward Network (FFN), dopo che i token hanno "guardato" gli altri tramite **self-attention**, ognuno di essi passa per una piccola rete neurale **indipendente** applicata allo stesso modo a tutti i token.

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2$$

Significa:

1. **Proiezione in uno spazio più grande**

   Il vettore x (dimensione d_{model}) viene moltiplicato per una matrice W_1 di dimensione più alta tipicamente 4 volte d_{model}. Questo serve a far apprendere **rappresentazioni più ricche**.

2. **Non linearità con GELU**

   GELU ("Gaussian Error Linear Unit") è una funzione di attivazione simile a ReLU ma più "morbida" che decide in modo probabilistico quanto dell'input x lasciar passare, in base a quanto è probabile che sia utile secondo una distribuzione gaussiana. In questo modo il modello introduce non linearità permettendo di rappresentare relazioni più complesse, non solo trasformazioni lineari.

3. **Proiezione di ritorno**

   Il risultato passa in W_2, che riporta il vettore alla dimensione originale d_{model} restituendo ogni token embedding **rimappato e arricchito** ma restando delle stesse dimensioni

5. Residual Connections + LayerNorm

Sia il blocco di Self-Attention che quello di FFN è avvolto da questa struttura:

$$output = LayerNorm\big(x + Sublayer(x)\big)$$

- **Residual Connection**: somma l'input x del blocco con l'output dello stesso. Questo permette di preservare l'informazione originale anche se il sublayer impara poco o in modo sbagliato ed aiuta il flusso dei gradienti durante il backpropagation, evitando il problema del vanishing gradient.
- **Layer Normalization**: normalizza i valori della somma per stabilizzare gli embedding e velocizzare il training.

Questo passaggio è presente dopo ogni blocco di Self-Attention e di FFN , garantendo che l'informazione del token e del contesto rimanga stabile e coerente durante le iterazioni nei vari strati del modello.

La Regression Head è un **layer fully connected** che rappresenta l'ultimo step del flusso di predizione. Dopo che la sequenza dei precedenti step di Multi-Head Self-Attention, Feed-Forward Network e Residual Connections + LayerNorm viene iterata per **N volte** (N strati del modello BERT), arriveremo in una situazione in cui ogni embedding dei token conterrà informazioni non solo sul proprio token, ma anche sul contesto dell'intera frase, grazie al meccanismo di self-attention. In particolare, il token speciale [CLS], aggiunto automaticamente all'inizio della sequenza dal tokenizer, è progettato per riassumere le informazioni dell'intera frase. Questo avviene perché, durante l'attenzione, il token [CLS] "guarda" tutti gli altri token e incorpora progressivamente le loro informazioni nei suoi embedding. L'ultimo step del flusso di predizione utilizza proprio l'embedding finale del token [CLS] come input alla Regression Head, che tramite un layer lineare produce un unico valore continuo che rappresenta lo score di sentiment della frase:

$$y_{\text{pred}} = W \cdot h_{[\text{CLS}]} + b$$

Dove:

- $h_{[CLS]}$ = embedding finale del token [CLS] (dimensione d_{model})
- W = vettore di pesi del layer lineare (dimensione $1 \times d_{model}$)
- b = bias scalare
- y_{pred} = score continuo di regressione

Dataset

Il modello è stato progettato per la **predizione della criticità nelle e-mail aziendali**. Tuttavia, non esistono dataset pubblici (né disponibili internamente) che contengano e-mail aziendali etichettate con un valore di sentiment. Inoltre, qualora tali dataset fossero reperibili, sarebbe necessario un oneroso lavoro manuale di annotazione, in quanto ogni e-mail dovrebbe essere classificata con un valore di sentiment. Per superare questo limite, è stato scelto di utilizzare un **dataset di recensioni TripAdvisor**, che presenta due vantaggi principali:

1. **Disponibilità di etichette già pronte** ogni recensione contiene una valutazione numerica che rappresenta implicitamente il sentiment.
2. **Dominio testuale vicino alle e-mail aziendali** le recensioni sono testi mediamente lunghi, articolati e orientati a esprimere un giudizio soggettivo (positivo, negativo o neutro), analogamente a quanto avviene nelle comunicazioni via e-mail.

Il dataset in questione è stato recuperato da:
https://www.kaggle.com/datasets/alessandrolobello/italian-tripadvisor

Pre-processing dei dati

Il dataset di recensioni presenta alcune **criticità tipiche dei testi non strutturati**, come la presenza di **emoji, righe vuote e link condivisi**. Questi elementi non contribuiscono in modo significativo alla

```
df = pd.read_csv("dataItalian.csv")

def clean_text(text):
    if pd.isnull(text):
        return ""
    text = str(text)
    text = re.sub(r"http\S+|www\.\S+", "", text)
    text = emoji.replace_emoji(text, replace='')
    text = "\n".join([line for line in text.splitlines() if line.strip() != ""])
    return text.strip()
df['review_text'] = df['review_text'].apply(clean_text)
df.to_csv("dataItalian.csv", index=False, encoding="utf-8")
print(f"Dataset pulito e sovrascritto' ({len(df)} righe).")
```

comprensione semantica del testo e rischiano di introdurre **rumore** durante il training del modello. Per questo motivo, tutte le recensioni sono state sottoposte a una fase di **pre-processing** che ha previsto:

- rimozione di **URL e link esterni**,
- eliminazione di **emoji e simboli non testuali**,
- filtraggio delle **righe vuote o superflue**.

Questa pulizia ha permesso di ottenere un corpus più coerente e privo di distrazioni, migliorando la qualità dell'input fornito al modello durante l'addestramento.

Mapping delle labels

Il dataset di recensioni fornisce come etichetta un valore intero compreso tra **1 e 5 stelle**, che rappresenta il giudizio associato al testo. Tuttavia, il modello è stato progettato per affrontare il problema come **task di regressione**, con output continuo compreso nell'intervallo 0,1. Per rendere le etichette compatibili con questo approccio, è stato introdotto un **mapping lineare delle stelle**:

- 1 stella → 0.00
- 2 stelle → 0.25
- 3 stelle → 0.50
- 4 stelle → 0.75
- 5 stelle → 1.00

Questo processo consente al modello di **apprendere una scala continua di sentiment**, mantenendo la coerenza con la natura ordinali delle valutazioni a stelle e permettendo una maggiore flessibilità in fase di predizione.

Downsample

Dall'analisi preliminare del dataset è emerso un forte sbilanciamento delle etichette:

Valore mappato	Stelle	Numero recensioni
1.00	5	108.016
0.00	1	32.119
0.75	4	18.558
0.50	3	4.763
0.25	2	3.512

Come si osserva, la classe 1.00 (5 stelle) è di gran lunga la più rappresentata, mentre le classi intermedie e soprattutto 0.25 (2 stelle) risultano fortemente sottorappresentate. Questo squilibrio avrebbe compromesso l'addestramento, inducendo il modello a privilegiare le predizioni sulle classi maggiori. Sono state considerate quindi due possibili strategie di bilanciamento:

- **Oversampling** delle classi minoritarie, questo avrebbe richiesto la creazione artificiale di nuove recensioni, ad esempio tramite tecniche di **data augmentation** basate sulla scomposizione e rimescolamento dei periodi. Tuttavia, l'enorme divario (oltre 100.000 recensioni tra la classe maggiore e la minore) avrebbe generato un corpus ridondante con numerose frasi ripetute, aumentando il rumore e riducendo la capacità di generalizzazione del modello.
- **Downsampling** delle classi maggioritarie ovvero la riduzione del numero di esempi per le classi più frequenti fino ad allinearle alla numerosità della classe minore.

Si è optato per il downsampling, portando tutte le classi a 3.512 recensioni ciascuna e di conseguenza il dataset finale utilizzato per il fine-tuning è composto da 17.560 recensioni bilanciate.

Suddivisione del dataset

Per addestrare e valutare correttamente il modello predisponiamo il dataset in tre sottoinsiemi distinti:

- **Training set (70%)** usato per aggiornare i pesi del modello durante il training.
- **Test set (15%)** usato durante il training per monitorare le prestazioni del modello la qualità dell'apprendimento effettuato durante il training.
- **Validation set (15%)** riservato a una valutazione finale, condotta **solo dopo** la fase di sviluppo; non è mai visibile al modello durante il training né durante il tuning degli iperparametri.

Questa separazione evita **data leakage** e permette di stimare in modo realistico la capacità di generalizzazione del modello. La proporzione 70/15/15 è stata scelta per mantenere un buon bilanciamento tra quantità di dati disponibili per l'addestramento e affidabilità delle valutazioni. La suddivisione sarà effettuata **in modo casuale ma riproducibile** (usando un seed fisso) per consentire replicabilità degli esperimenti e verrà preservata la distribuzione delle classi tra i set attraverso uno **split stratificato**.

Addestramento

Il fine-tuning del modello *dbmdz/bert-base-italian-xxl-cased* per la predizione del sentiment tramite regressione è stato realizzato utilizzando la classe **Trainer** della libreria Hugging Face Transformers, che automatizza gran parte del processo di addestramento.

Funzionamento del Trainer

1. **Preparazione dei batch**
   Prende i dati dal train_dataset ed usa il **data_collator** impostato per costruire i batch nel nostro caso tensori con input_ids, attention_mask, labels per poi caricare ogni batch in GPU (se disponibile).
2. **Forward Pass**
   Per ogni batch i **token** entrano nel modello. BERT tramite il flusso di predizione sopra illustrato produce le rappresentazioni e la **regression head** (il layer lineare finale) trasforma la rappresentazione [CLS] in un valore numerico ovvero la predizione.
3. **Calcolo della Loss**
   Confronta la predizione con la label reale usando la **loss function**. Nel nostro caso optiamo per la Mean Squared Error, una **metrica di regressione** che misura quanto le predizioni di un modello si discostano dai valori reali.

4. **Backpropagation**
   Viene calcolato il **gradiente** della loss rispetto a tutti i pesi del modello tramite

$$\frac{\partial \mathcal{L}oss}{\partial W}$$

   Questo gradiente indica la direzione in cui i pesi devono essere cambiati per **ridurre l'errore**.

5. **Gradient Accumulation**
   Hugging Face prima di aggiornare i pesi accumula i gradienti su più batch per simulare un batch più grande senza saturare la GPU.

6. **Ottimizzazione (AdamW)**
   Hugging Face usa di default **AdamW**, una variante di Adam che gestisce meglio la regolarizzazione aggiornando i pesi del modello usando i gradienti calcolati.

$$W_{\text{nuovo}} = W_{\text{vecchio}} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W}$$

   η Indica il learning rate che tramite lo **scheduler del learning rate** può scalare il passo di apprendimento durante il training.
7. **Logging ed Evaluation**
   Alla fine di ogni epoca (in base a logging_strategy ed eval_strategy):
   - calcola le metriche definite in compute_metrics (MSE, MAE, R^2) sul test set.
   - salva i log in logging_dir.
   - Grazie a load_best_model_at_end=True, tiene traccia del modello con le performance migliori.
   - La callback controlla le metriche a fine epoca. Se non ci sono miglioramenti per un determinato numero di epoche interrompe il training, altrimenti ripete il ciclo fino al raggiungimento delle epoche prefissate.

Argomenti del Trainer

L'oggetto **Trainer** prende in input:

- il **modello** (model) che elaborerà i dati,
- i **parametri di addestramento** (args)
- il **dataset di addestramento** (train_dataset) e il **dataset di valutazione** (eval_dataset),
- il **tokenizer** (processing_class), utilizzato per processare i dati testuali,
- il **data collator**, che trasforma i batch di dati in tensori pronti per il modello,
- la funzione **compute_metrics**, che calcola le metriche di valutazione alla fine di ogni epoca,
- e infine il **callback di early stopping**, che interrompe automaticamente l'addestramento se la metrica monitorata non migliora per un certo numero di epoche.

Gli argomenti del Trainer che massimizzeranno l'apprendimento del Trainer sono:

- `output_dir="./sentiment_regression"`
  Determina la cartella in cui verranno salvati i **checkpoint del modello** e i file di configurazione. Alla fine di ogni epoca il Trainer creerà una sottocartella qui.
- `eval_strategy="epoch"`
  Determina **quando fare la valutazione sul dataset di validazione**. In questo caso alla fine di ogni epoca ovvero **un passaggio completo del modello su tutto il dataset di addestramento**.
- `save_strategy="epoch"`
  Determina **quando salvare i checkpoint** del modello. In questo caso salverà alla fine di ogni epoca all'interno di "./sentiment_regression".
- `fp16=True`
  Abilita **il mixed precision training** (16-bit floating point). Riduce l'uso di memoria GPU e velocizza l'addestramento mantenendo quasi la stessa precisione. Normalmente il deep learning usa i numeri in **32-bit floating point (FP32)** più precisi, ma che occupano più memoria e richiedono più tempo di calcolo. Con **mixed precision training** alcune parti del calcolo usano **16-bit floating point (FP16)** occupando la metà della memoria e calcolando più velocemente sulla GPU mentre le operazioni sensibili (es. accumulo dei gradienti, aggiornamento dei pesi) restano in **32-bit** per evitare perdita di precisione.
- `per_device_train_batch_size=4`
  Numero di recensioni inclusi in ogni batch durante l'addestramento.
- `per_device_eval_batch_size=4`
  Numero di recensioni per batch durante la valutazione sul test set. Influisce solo sulla velocità di valutazione, non sull'addestramento dei pesi.
- `logging_strategy="epoch"`
  Frequenza con cui vengono salvati i log di addestramento: in questo caso una volta alla fine di ogni epoca.
- `logging_dir='./logs'`
  Determina la cartella in cui vengono salvati i log di addestramento.
- `load_best_model_at_end=True`
  Dopo l'addestramento, viene caricato automaticamente il modello con le migliori prestazioni in base alla metrica di valutazione scelta (MSE).
- `num_train_epochs=10`
  Numero di epoche totali: quanti passaggi completi il modello farà sull'intero dataset di addestramento (salvo **callback di early stopping)**.
- `learning_rate=2e-5`
  Tasso di apprendimento iniziale dell'ottimizzatore η. Impostato a 0.00002, tramite lo scheduler del learning rate diminuisce gradualmente durante il training delle varie epoche fino a un valore vicino a zero
- `gradient_accumulation_steps=8`

Numero di batch raggruppati nella fase di gradient accumulation per un calcolo dei gradienti simulando un batch più grande senza saturare la GPU

- `weight_decay=0.01`
  Con l'aggiunta del weight decay, l'aggiornamento dei pesi diventa:
  $$W_{\text{nuovo}} = W_{\text{vecchio}} - \eta \cdot \left(\frac{\partial \mathcal{L}}{\partial W} + \lambda W_{\text{vecchio}}\right)$$
  Dove λ è il parametro di weight decay, che controlla l'intensità della penalizzazione. Questo agisce come una forza che spinge i pesi verso zero, riducendo la loro magnitudine. Aiutando a prevenire che i pesi crescano troppo grandi, il che potrebbe portare a un modello che si adatta troppo ai dati di addestramento (overfitting).

Valutazione

Valutazione dei risultati

Dall'esecuzione del training si può osservare l'andamento delle metriche per epoca. Già alla **terza epoca** il modello raggiunge le sue migliori performance, con un **Validation Loss pari a 0.0279**, un **MAE di 0.1283** e un **R^2 di circa 0.77**. A partire dalla **quarta epoca**, tuttavia, il Validation Loss peggiora nonostante il Training Loss continui a diminuire. Questo è un chiaro segnale di **overfitting**: il modello impara troppo bene le specificità del set di training, perdendo capacità di generalizzazione sui dati non visti. Questa dinamica è attesa, considerando che il dataset, dopo il bilanciamento, conta circa **15.000 recensioni**: una dimensione utile per il fine-tuning, ma comunque ridotta rispetto alla complessità di un modello di tipo BERT. Difatti, grazie al meccanismo di **Early Stopping**, il training viene interrotto automaticamente quando per due epoche consecutive non si registra un miglioramento. Inoltre, l'opzione **load_best_model_at_end** garantisce che, al termine del processo, venga salvato e restituito il modello corrispondente alla miglior epoca, in questo caso la terza.

Epoch	Training Loss	Validation Loss	Mse	Mae	R2
1	0.045600	0.031330	0.031330	0.134902	0.747506
2	0.027300	0.029241	0.029241	0.128033	0.764342
3	0.021100	0.027957	0.027957	0.128308	0.774691
4	0.016300	0.031332	0.031332	0.132238	0.747483
5	0.012700	0.029596	0.029596	0.129294	0.761478

I valori ricavati ci dicono che:

- **Validation Loss (0.0279)**: essendo la loss definita come **MSE**, significa che in media lo scarto quadratico tra predizione e valore reale è molto basso (≈0.028). Più la loss è vicina a 0, più le predizioni si avvicinano ai valori target.
- **MAE (0.1283)**: indica l'errore medio assoluto tra predizione e valore reale. Qui è circa **0.13** su una scala che va da 0 a 1 vuol dire che, in media, il modello sbaglia di circa **13 punti percentuali**. Ad esempio: se una recensione ha come valore target 0.75, il modello in media predirà valori compresi tra 0.62 e 0.88.
- **R^2 ≈ 0.77**: significa che il modello spiega circa il **77% della varianza** dei dati di validaziorene. È un buon valore, soprattutto considerando che il dataset è relativamente piccolo e che la regressione sul sentiment è un problema complesso.

Pertanto possiamo dedurre che il modello, nonostante la disponibilità limitata di dati (≈17.500 recensioni), riesce a catturare in maniera efficace la relazione tra testo e livello di criticità. Riuscendo a spiegare oltre i tre quarti della variabilità presente nel dataset, con un margine di errore contenuto. La presenza di un leggero overfitting dopo la terza epoca evidenzia come l'aumento del numero di dati o in un successivo re-training aggiungendo e-mail inerenti alla problematica porterebbe a migliorare ulteriormente le prestazioni. Nel complesso, il modello si dimostra adatto al compito di regressione del sentiment e fornisce predizioni affidabili, costituendo una buona base per applicazioni reali di analisi delle email aziendali.

Valutazione del Validation Set

Il **validation set** ha prodotto i seguenti risultati finali:

- **MSE (Mean Squared Error): 0.0260**
- **MAE (Mean Absolute Error): 0.1234**
- **R^2: 0.7810**

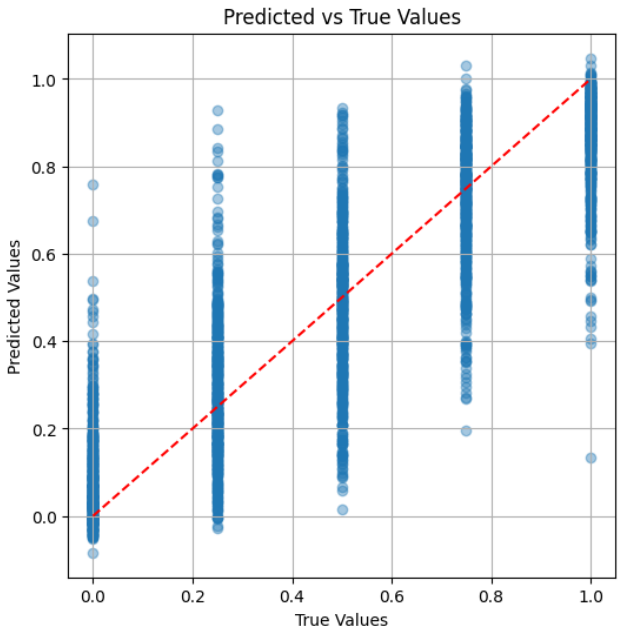
Questi valori indicano che il modello è in grado di generalizzare bene anche su dati **mai visti prima**, mantenendo prestazioni molto simili a quelle osservate durante il training. In particolare:

- L'**MAE basso** dimostra che l'errore medio nelle predizioni è contenuto e vicino al valore reale.
- Il **MSE ridotto** conferma che il modello riesce a limitare anche gli errori più grandi.
- Un **R^2 di 0.78** implica che circa il 78% della variabilità del sentiment è spiegata dal modello, un risultato solido considerando la complessità del linguaggio naturale e la dimensione ridotta del dataset.

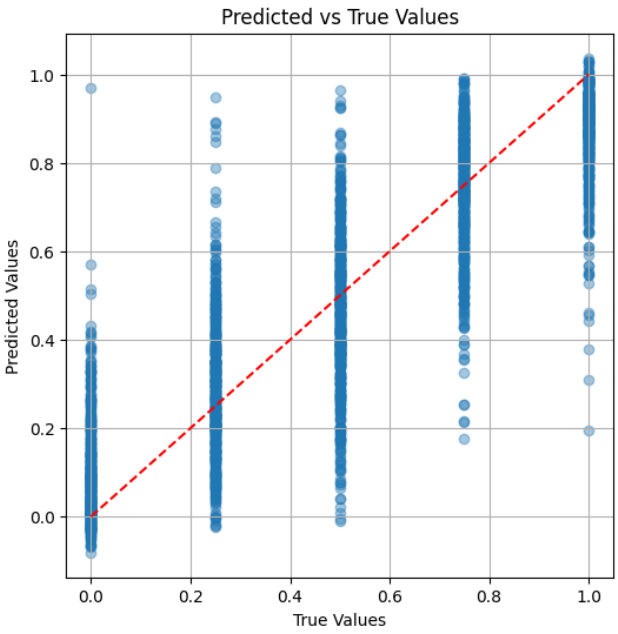
Questi dati dimostrano che il modello non si è limitato a memorizzare i dati di training, ma ha appreso schemi utili e trasferibili, riuscendo a **mantenere una buona capacità predittiva anche su esempi mai incontrati**. Nel complesso, ciò conferma l'efficacia del fine-tuning e la robustezza del modello come base per un sistema di analisi del sentiment applicabile in contesti reali.

Comparazione grafica

Test Set

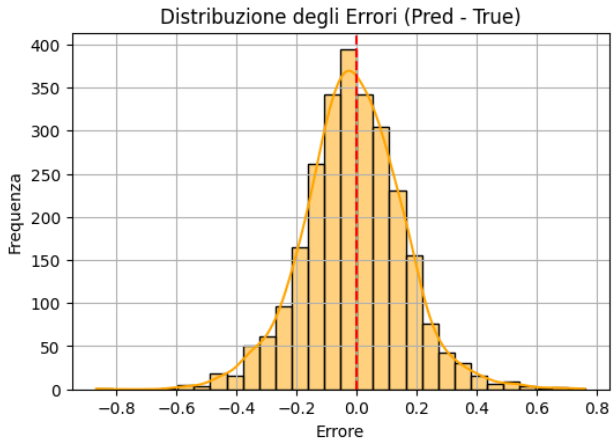


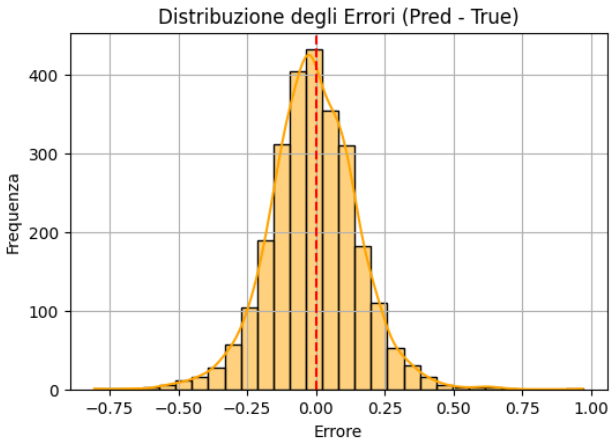
Valutation Set



Predizioni vs Valori reali

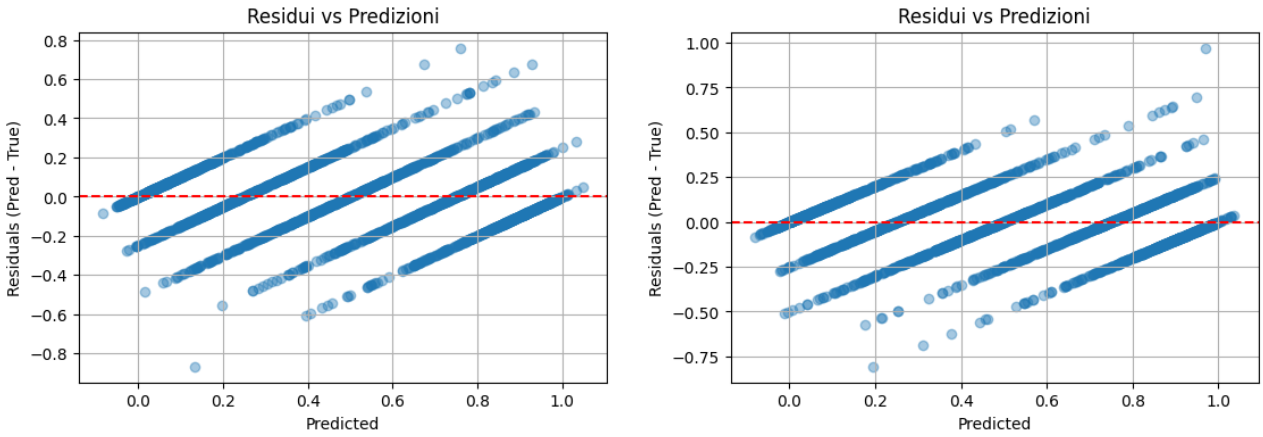
I punti si dispongono lungo colonne verticali, a causa del **mapping discreto** delle stelle in valori reali che hanno solo 5 possibili livelli. La maggior densità di punti è registrata vicino alla linea rossa ciò implica buona accuratezza: il modello predice valori vicini al reale.
C'è un po' di dispersione, soprattutto nelle classi intermedie (0.25–0.75), segno che il modello è meno sicuro in situazioni in cui la recensione è mite rispetto agli estremi.





Distribuzione degli errori

La distribuzione è **simmetrica e centrata intorno allo zero**, quindi il modello non ha bias marcato. La forma a campana è vicina a una normale questo implica buona proprietà statistica, gli errori sono distribuiti in modo regolare. La maggior parte degli errori è contenuta tra **−0.15 e +0.15**, in linea con la MAE risultata, quindi lo scostamento medio è ridotto.



Residui vs Predizioni

I residui sono distribuiti abbastanza simmetricamente intorno allo zero quindi il modello non ha bias sistematico (non sovrastima o sottostima sempre).
Si notano bande diagonali, dovute al fatto che le etichette originali derivano da un **mapping discreto** (stelle in valori 0, 0.25, 0.5, 0.75, 1). Il modello predice valori continui, ma tende comunque a raggrupparsi attorno a quelle soglie.

Comparazione con modello preesistente

In conclusione, è stata effettuata una comparazione con un modello presente in azienda per lo svolgimento del medesimo task. Il modello aziendale LLaMA funziona **come un LLM generico**: fondamentalmente, riceve in input il testo della e-mail e viene istruito a **predire il sentimento** tramite un prompt strutturato, in maniera simile a come si interagirebbe con ChatGPT.
Tra le maggiori differenze:

- Il testo dell'e-mail viene "dato in pasto" al modello senza fine-tuning sul task specifico. Il modello genera quindi una **classificazione del sentiment** e una spiegazione, basandosi sulle sue conoscenze generali apprese durante il pre-training.
- L'approccio è quindi **prompt-driven**, e la precisione dipende dal linguaggio e dal contesto della e-mail, senza adattamento diretto ad un dataset specifico.
- La Predizione è **categorica**, senza possibilità di distinzione granulare inoltre non è possibile una distinzione tra **oggetto e corpo della mail** senza possibilità di calcolare medie pesate tra i due.
- Tempo di risposta più elevato rispetto a modelli ottimizzati per task specifici essendo un LLM generico, deve elaborare l'intero testo della e-mail e generare una spiegazione testuale oltre che la sua predizione
- **Sicurezza e privacy**: LLaMA non è un modello proprietario locale, quindi l'elaborazione delle email sensibili aziendali comporta rischi di privacy. Non è consigliabile inviare dati riservati a modelli esterni o in cloud, mentre il modello fine-tunato è **salvato e gestito internamente**, garantendo piena sicurezza dei dati.

I modelli sono stati dunque confrontati su un dataset di 25 mail aziendali.

	Sentiment Modello LLaMA	Sentiment Modello Progettato	Valutazione umana
1	Positivo	Positivo (0.706313)	Positivo
2	Neutro	Critico (0.378834)	Negativo/Critico
3	Positivo	Positivo (0.695130)	Positivo
4	Negativo	Critico (0.178528)	Negativo/Critico
5	Positivo	Positivo (0.610413)	Positivo
6	Positivo	Positivo (0.753420)	Positivo
7	Negativo	Critico (0.321840)	Negativo/Critico
8	Positivo	Positivo (0.653510)	Positivo
9	Negativo	Critico (0.337634)	Negativo/Critico
10	Positivo	Medio (0.496337)	Positivo
11	Negativo	Critico (0.169815)	Negativo/Critico
12	Positivo	Positivo (0.707988)	Positivo
13	Negativo	Critico (0.356359)	Negativo/Critico
14	Positivo	Positivo (0.812338)	Positivo
15	Negativo	Critico (0.229822)	Negativo/Critico
16	Positivo	Medio (0.502515)	Medio
17	Negativo	Medio (0.416985)	Tra Medio e Negativo
18	Positivo	Positivo (0.712428)	Positivo
19	Negativo	Critico (0.188929)	Negativo/Critico
20	Positivo	Positivo (0.689351)	Positivo
21	Negativo	Medio (0.513048)	Medio
22	Negativo	Critico (0.195180)	Negativo/Critico
23	Negativo	Medio (0.512196)	Medio
24	Negativo	Medio (0.479015)	Medio
25	Positivo	Medio (0.534046)	Medio

Dalla comparazione emerge che il modello progettato non solo **è più veloce** (4.6 secondi vs 1.39 minuti del modello LLaMa) ma riesce anche ad eseguire le predizioni in modo nel complesso accurato e in linea con la valutazione umana delle e-mail, con soltanto un errore evidente di classificazione. Rispetto al modello LLaMA, che restituisce una classificazione più dicotomica (positivo/negativo) e tende a trascurare le sfumature intermedie, il nuovo modello mostra una **maggiore granularità**, riuscendo a distinguere meglio i casi di criticità o ambiguità e capendo il grado della criticità rilevata.
Si può dunque dedurre che il modello progettato rappresenta una soluzione valida, capace di sostituire efficacemente la precedente implementazione, migliorandone la precisione e l'aderenza al giudizio umano.

How to use

Alla fine del processo di fine-tuning il modello verrà salvato tramite:

```
model.save_pretrained("./sentiment_model")
tokenizer.save_pretrained("./sentiment_model")
```

Il modello potrà quindi essere caricato e utilizzato seguendo questi passaggi:

1. Caricamento del modello e del tokenizer dalla directory di salvataggio.

```
MODEL_DIR = "./sentiment_model"
tokenizer = AutoTokenizer.from_pretrained(MODEL_DIR)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_DIR)
```

2. Settaggio del modello in modalità valutazione, poiché vogliamo solo effettuare predizioni e non training.

```
model.eval()
```

3. Tokenizzazione del testo tramite il tokenizer e conversione in tensori PyTorch

```
text = "ciao bel modello"
inputs = tokenizer(text, return_tensors="pt")
```

4. Elaborazione del testo e produzione dell'output tramite il modello.

```
outputs = model(**inputs)
score = outputs.logits.squeeze().item()
print("Valore predetto (regressione):", score)
```

Possibili sviluppi futuri

Uno sviluppo fondamentale per l'utilizzo delle e-mail negli scambi con la Pubblica Amministrazione è l'**analisi documentale degli allegati**, in particolare dei PDF, che spesso contengono le informazioni più rilevanti e a valore legale. Lo sviluppo di un aggiornamento di questo tipo può avvenire secondo i seguenti step:

1. **Riconoscimento del tipo di documento**
   Il modello identifica se il file allegato è un PDF testuale o una scansione.
   a. Per PDF testuali si utilizzano librerie come pdfminer.six, PyPDF2 o Apache Tika per estrarre il testo.
   b. Se il PDF è una scansione, viene applicato un layer OCR (es. Tesseract, docTR o API cloud) per convertire l'immagine in testo analizzabile.
2. **Segmentazione del documento**
   Il testo estratto viene suddiviso in sezioni significative.
   Si eliminano intestazioni, piè di pagina, informazioni di mittente/destinatario e altre parti superflue, concentrandosi sul corpo del documento.
3. **Analisi semantica e scoring**
   Il testo viene pulito da caratteri speciali, link, virgolette e altri elementi di disturbo, quindi passato al modello di regressione, che assegna un punteggio continuo di criticità.
   Il punteggio derivato dal documento viene combinato con quello dell'oggetto e del corpo della e-mail tramite una media ponderata, ottenendo un punteggio finale per la e-mail.
   Un'alternativa evoluta consiste nel passare il testo del documento a un **LLM**, in grado di restituire una spiegazione del contenuto. Questa spiegazione può essere poi utilizzata da un secondo modello per valutare la criticità, il cui risultato verrà combinato con quello dell'oggetto e del corpo della e-mail tramite la stessa media ponderata, migliorando la comprensione semantica e la trasparenza del processo.