Classificazione Binaria del Sentiment in italiano
Confronto Multi-Seed di BERT-base Italian uncased e XLM-RoBERTa-base

Virginia Vento
virginia.vento@studio.unibo.it

Bianca De Crecchio
bianca.decrecchio@studio.unibo.it

Abstract

Questo progetto esplora il task di classificazione binaria del sentiment in italiano mediante il fine-tuning di modelli transformer ("BERT-base Italian uncased" e "XLM-RoBERTa-base" multilingue) su un dataset combinato di risorse esistenti (FEEL-IT, MultiEmotions-It, SENTIPOLC16). L'adozione di un approccio multi-seed ha permesso di valutarne la robustezza, mostrando prestazioni ampiamente superiori rispetto alla baseline casuale stratificata. I risultati evidenziano una lieve superiorità del modello BERT-base in Accuracy e Macro-F1, mentre il modello XLM-RoBERTa si distingue nelle metriche probabilistiche, come ROC-AUC. Valutato su uno split stratificato del dataset combinato, BERT-base supera UmBERTo – addestrato per lo stesso task su FEEL-IT e valutato sul test set di SENTIPOLC16 – nelle metriche Macro-F1 e Accuracy.

1. Introduzione

In questo lavoro abbiamo adattato lo script *run_glue_no_trainer.py* per il fine-tuning dei modelli transformer **BERT-base Italian uncased** (*dbmdz/bert-base-italian-uncased*) e **XLM-RoBERTa-base** (*xlm-roberta-base*) su un task di classificazione binaria del sentiment in lingua italiana, utilizzando diversi dataset contenenti testi tratti da social network, quali Twitter, YouTube e Facebook.

Abbiamo, inoltre, condotto gli esperimenti utilizzando diversi valori di *random seed* (42, 123, 999), al fine di verificare la stabilità e la riproducibilità dei risultati.

2. Dati e preparazione

2.1 Modelli

Come anticipato, per il nostro task di classificazione binaria del sentiment in lingua italiana, abbiamo utilizzato due modelli transformer distinti:

- ***BERT-base Italian uncased*** (d'ora in poi BERT): un modello pre-addestrato su testi italiani, quindi particolarmente adatto a catturare le sfumature linguistiche specifiche dell'italiano.

- ***XLM-RoBERTa-base*** (d'ora in poi RoBERTa): un modello multilingue della famiglia BERT, pre-addestrato su grandi corpora in oltre cento lingue, più robusto su testi eterogenei e social media multilingue, ma meno specializzato sull'italiano puro.

2.2 Descrizione dei dataset

Per addestrare i due modelli su un ampio numero di esempi e catturare la varietà linguistica dei testi italiani sui social media, abbiamo selezionato dataset già disponibili, creati e annotati appositamente per task di emotion e/o sentiment analysis, contenenti tweet e commenti pubblicati sulle piattaforme YouTube e Facebook.

Di seguito, riportiamo per ciascun dataset una breve descrizione:

- **FEEL-IT**: dataset contenente 2037 tweet in italiano, annotati con quattro emozioni di base: rabbia, paura, gioia e tristezza. L'annotazione è stata effettuata da alcuni ricercatori del MilaNLP Group[1] nell'ambito del loro lavoro *FEEL-IT: Emotion and Sentiment Classification for the Italian Language*[2] per fornire un benchmark per l'analisi delle emozioni nei testi italiani. Nel nostro codice questo dataset è stato rinominato ***data1***.

- **MultiEmotions-It:** dataset contenente 3240 commenti a video musicali e pubblicità su YouTube e Facebook, annotati manualmente su quattro dimensioni: rilevanza (commento pertinente o meno rispetto al contenuto), polarità di opinione (positiva vs. negativa), emozioni[3] e sarcasmo. L'annotazione è stata effettuata durante il seminario "Sentiment Analysis"[4], organizzato nell'ambito della Laurea magistrale 'Comunicazione per l'impresa, i media e le organizzazioni complesse' presso l'Università Cattolica del Sacro Cuore di Milano. Nel nostro codice questo dataset è stato rinominato ***data2***.

- **SENTIPOLC 2016:** dataset contenente 9410 tweet in italiano, annotati per soggettività, polarità generale (orientamento emotivo complessivo del testo), polarità letterale (basata sulle singole parole, senza considerare il contesto), e ironia. Creato per il task SENTIPOLC – SENTIment POLarity Classification 2016[5] nell'ambito della campagna EVALITA 2016, era già suddiviso in due sottoinsiemi: "train" (*training_set_sentipolc16_anon_rev.csv*) e "test" (*test_set_sentipolc16_gold2000_anon_rev.csv*), contenenti rispettivamente 7410 e 2000 tweet. Tali sottoinsiemi, nel nostro codice, sono stati rinominati rispettivamente ***data3*** e ***data4***.

2.3 Pre-processing dei dati

Per rendere omogenei i dati destinati al fine-tuning dei modelli transformer, abbiamo applicato una serie di operazioni di pre-processing a tutti i dataset selezionati.

2.3.1 Selezione delle etichette

Per ciascun dataset, abbiamo uniformato le etichette al formato binario richiesto dal task di classificazione del sentiment (positivo vs. negativo):

[1] https://milanlproc.github.io/
[2] https://aclanthology.org/2021.wassa-1.8/ (Bianchi *et al.,* WASSA 2021).
[3] Per l'annotazione delle emozioni è stato adottato il modello di Plutchik, tenendo conto sia delle otto emozioni di base (*gioia, tristezza, paura, rabbia, fiducia, disgusto, sorpresa, trepidazione/attesa*) che di quelle complesse, ovvero le diadi (per esempio l'emozione *amore* è considerata come l'unione di *gioia* e *fiducia*).
[4] https://github.com/RacheleSprugnoli/Esercitazioni_SA
[5] http://www.di.unito.it/~tutreeb/sentipolc-evalita16/index.html

- **FEEL-IT (*data1*)**: le emozioni originariamente presenti nel dataset sono state mappate in etichette binarie: *joy* → 1 (sentiment positivo), mentre *sadness*, *fear* ed *anger* → 0 (negativo). Sono stati scartati eventuali esempi con etichette ambigue.
- **MultiEmotions-It (*data2*)**: i tweet e i commenti sono stati filtrati per considerare solo quelli con polarità d'opinione univoca, ossia con valori *POS* = 1 e *NEG* = 0 (sentiment positivo) oppure POS = 0 e NEG = 1 (sentiment negativo). In seguito, la colonna contenente i commenti è stata rinominata *text*, mentre la colonna della polarità positiva (*POS*) è stata trasformata in *labels*, con valore 1 per i commenti positivi e 0 per quelli negativi. Gli esempi ambigui sono stati scartati.
- **SENTIPOLC 2016 (*data3* e *data4*)**: prima di tutto, sono stati esclusi i tweet ambigui o con etichette inconsistenti, che rappresentavano sentiment neutro o misto (dove, per esempio, *opos* = *oneg*). I due sottoinsiemi, originariamente denominati "train" e "test", presentavano strutture leggermente diverse: le colonne del secondo sono state rinominate e adattate a quelle del primo (*opos, oneg* e *text*), garantendo così coerenza tra i dataset. Successivamente, entrambi i sottoinsiemi sono stati processati per creare una colonna binaria *labels*, derivata dalla polarità negativa (oneg) secondo la logica: se *oneg* = 1 → *labels* = 0 (sentiment negativo); se *oneg* = 0 → *labels* = 1 (sentiment positivo).

2.3.2 Merge e pulizia dei dataset

Dopo aver uniformato le etichette e adattato le colonne, i dataset (*data1* = FEEL-IT, *data2* = MultiEmotions-It, *data3* = SENTIPOLC train, *data4* = SENTIPOLC test) sono stati concatenati in un unico DataFrame. In questa fase, sono state rimosse eventuali righe con valori mancanti o testo vuoto, e il dataset risultante – rinominato ***data*** - è stato mescolato casualmente mediante uno *shuffle* controllato da un *seed*, al fine di evitare qualsiasi ordinamento preesistente e garantire la riproducibilità degli esperimenti. Infine, la colonna *labels* è stata uniformata a valori numerici interi (0 = sentiment negativo, 1 = sentiment positivo), rendendo i dati pronti per il *fine-tuning* dei modelli *transformer*.

2.3.3 Suddivisione di *data* in training, validation e test

Il nuovo dataset ***data*** è stato suddiviso in sottoinsiemi di training, validation e test tramite una procedura stratificata sulla colonna *labels*, in modo da mantenere la stessa proporzione di esempi positivi e negativi in ciascun sottoinsieme. In particolare:

- ***data*** è stato prima suddiviso in due parti: 80% come *training set* e 20% temporaneo;
- Successivamente, il 20% temporaneo è stato ulteriormente suddiviso in parti uguali, ottenendo un 10% come *validation set* e un 10% come *test set*.

I tre sottoinsiemi così ottenuti sono stati denominati ***train_df*** (80% di *data*), ***val_df*** (10%) e ***test_df*** (10%). Questo approccio ha permesso di costruire tre insiemi coerenti e bilanciati, idonei per l'addestramento, la validazione e la valutazione dei modelli.

3. Fasi di *training* e *validation*

3.1 *Training set up*

3.1.1 *Baseline* proporzionale

A questo punto, a partire dai dati ottenuti dall'unione dei dataset selezionati, è stato introdotto il calcolo di una ***baseline* di riferimento** basata sulla proporzione reale delle classi presenti. Una *baseline* rappresenta la performance dell'algoritmo più semplice possibile ed è essenziale per valutare l'efficacia di sistemi più complessi: costituisce un punto di partenza necessario per il confronto con i risultati dell'addestramento[6].

Il codice calcola la distribuzione delle classi e stima la probabilità relativa di ciascuna. Attraverso un generatore casuale, vengono poi create etichette di previsione rispettando queste probabilità: la *baseline*, quindi, non produce previsioni del tutto casuali, ma rispetta le proporzioni reali dei dati. In questo modo fornisce un termine di paragone minimo e realistico che un modello di deep learning deve superare nettamente per essere considerato efficace.

3.1.2 Iperparametri

Per l'addestramento dei modelli sono stati scelti specifici **iperparametri**, cioè parametri impostati prima del *training* che influenzano il comportamento dell'ottimizzazione ma che non vengono aggiornati dal modello:

- **Numero di epoche:** 6 (il modello vede tutti i dati 6 volte)
- **Batch size:** 16 (numero di esempi elaborati insieme in un passo del *training*)
- **Learning rate:** 1e-5 (velocità con cui il modello aggiorna i pesi)
- **Weight decay:** 0.01 (piccolo "freno" per evitare l'*overfitting*)
- **Early stopping con pazienza:** 4 epoche (il training si interrompe se il modello non migliora per 4 epoche consecutive)

3.1.3 Tokenizzazione

I dati testuali sono stati elaborati utilizzando i tokenizer specifici dei due modelli impiegati. Ogni testo è stato convertito in una sequenza numerica compatibile con l'input dei transformer, con lunghezza massima di **160 token** (*max_length = 160*). Le sequenze che eccedono questo limite vengono troncate, mentre quelle più corte sono uniformate all'interno di ciascun batch (BATCH_SIZE = 16) mediante la classe *DataCollatorWithPadding* di HuggingFace, che applica un padding dinamico alle sequenze. Questo approccio, integrato con *Accelerate,* garantisce un uso ottimizzato della memoria GPU.

3.1.4 Gestione dello sbilanciamento: Focal Loss

Analizzando la distribuzione delle classi nei set di *training* e *validation*, si è osservato uno sbilanciamento tra esempi positivi e negativi (circa 46% vs. 54%). Per gestire questo squilibrio, è stata definita la **Focal Loss** (poi applicata nel training loop, cfr. § 3.2.2), una funzione di perdita che

[6] *Manning & Schütze,*1999.

riduce il peso degli esempi facilmente classificabili e aumenta l'attenzione su quelli più difficili o meno rappresentati.

3.2 Training loop

3.2.1 *Multi-seed*

Gli esperimenti sono stati ripetuti con **tre** valori distinti di ***random seed*: 42, 123, 999**. Questa strategia, chiamata ***Multi-Seed***, riduce la variabilità dovuta all'inizializzazione casuale dei pesi e all'ordine dei dati durante il *training*, permettendo di ottenere metriche più stabili e affidabili. Per ciascun *seed*, tutti i generatori di numeri casuali sono fissati al valore corrispondente e le librerie cuDNN configurate in modalità deterministica, permettendo la replicabilità dei risultati.

All'interno del loop di training, il dataset ***data*** viene suddiviso in modo controllato attraverso la funzione *stratified_split* (cfr. § 2.3.3), richiamata separatamente per ciascun seed. I dati risultanti sono, poi, organizzati in *DataLoader* di HuggingFace, che gestiscono automaticamente ***batching*** e ***padding* dinamico** (cfr. § 3.1.3), garantendo una preparazione coerente dei batch per addestramento e validazione dei modelli.

3.2.2 *Training* e *Validation* per ciascun *seed*

Per ciascun *seed*, il modello viene addestrato sul corrispondente training set (*train_df)*, calcolando il pos_weight (che attribuisce maggiore peso alla classe dei positivi) e la Focal Loss (cfr. § 3.1.4) su ogni batch: questa combinazione permette di gestire il leggero sbilanciamento tra le classi. L'aggiornamento dei pesi avviene tramite ***backpropagation***, regolato dall'ottimizzatore AdamW e dal *learning rate scheduler* lineare.

Dopo ogni epoca, **il modello viene valutato** sul *validation set* (*val_df*): viene posto in modalità di valutazione (*eval*), disattivando tutti i comportamenti casuali come il dropout[7], in modo da ottenere stime affidabili delle prestazioni. Le probabilità predette dal modello vengono confrontate con le etichette reali per tracciare la **curva *precision-recall*** e individuare la soglia che massimizza l'F1 score macro, bilanciando precision e recall tra le due classi.

Se il valore Macro-F1ottenuto in quell'epoca supera il valore migliore registrato fino a quel momento, il modello viene salvato; altrimenti, viene incrementato un contatore di pazienza. Quando il contatore raggiunge il numero massimo di epoche senza miglioramenti (patience = 4), l'***early stopping*** interrompe l'addestramento, preservando il modello con le migliori prestazioni.

Al termine del ***loop training-validation*** per ciascun *seed*, i pesi del modello vengono ricaricati dalla versione salvata con la migliore Macro-F1, garantendo che le fasi successive utilizzino sempre il modello ottimale.

3.2.3 Test finale e metriche

Dopo il *training* e la selezione del modello migliore per ciascun *seed* (basata sulle prestazioni sul *validation set*), viene eseguita la **fase di *testing* sui dati mai visti** (*test_df*). Anche in questo caso, il

[7] La strategia di dropout prevede che alcuni neuroni vengono "spenti" casualmente durante il training per evitare overfitting.

modello viene posto in modalità di valutazione (*eval*), quindi nessun peso viene aggiornato e i comportamenti casuali sono disattivati per garantire valutazioni affidabili.

L'analisi viene ripetuta separatamente per ciascun *seed*, così da verificare la robustezza del modello rispetto alla variabilità introdotta dai diversi valori di *random seed*. Questo permette di osservare come cambiano le prestazioni al variare della randomizzazione dei pesi iniziali e dell'ordine dei dati nei batch.

Per ogni *seed*, vengono calcolate **metriche globali di performance**, quali:

- **Accuracy** – proporzione di predizioni corrette;
- **Macro-F1** – media non pesata dell'F1 score tra le classi, utile per dataset sbilanciati;
- **ROC-AUC e PR-AUC** – capacità del modello di distinguere tra classi positive e negative;
- **Matthews Correlation Coefficient (MCC)** – indicatore bilanciato della qualità della classificazione;
- **Balanced Accuracy** – media dell'Accuracy calcolata separatamente per ciascuna classe.

Sempre per ogni seed, vengono calcolate anche **metriche specifiche per ciascuna classe**:

- **Precision** – proporzione di predizioni corrette rispetto al totale delle predizioni assegnate a quella classe;
- **Recall** – proporzione di esempi della classe correttamente identificati;
- **Support** – numero di esempi effettivamente presenti in ciascuna classe.

Viene, inoltre, effettuata un'**analisi degli errori**, identificando gli esempi classificati in modo errato e, utile per comprendere eventuali pattern di difficoltà o casi particolarmente complessi per il modello.

Infine, i risultati dei tre *seed* vengono aggregati per calcolare medie e deviazioni standard delle principali metriche. Vengono quindi generati due grafici: uno per **le curve di Loss** e uno per **le curve ROC**, ciascuno contenente le tre curve corrispondenti ai diversi *seed*,[8] permettendo di visualizzare l'andamento del *training* e la capacità discriminativa dei modelli.

4. Risultati[9]

4.1 Confronto BERT vs RoBERTa

Sulla base delle metriche descritte nel paragrafo precedente, abbiamo confrontato le prestazioni dei due modelli presi in esame, **BERT** e **RoBERTa**, ciascuno valutato su tre diversi *seed*. Le **Tabelle 1 e 2** riportano i valori medi e le deviazioni standard delle principali metriche di performance, sia globali che per ciascuna classe.

[8] I due grafici sono caricati nella cartella **results\.**

[9] Il codice fornito (code.py) esegue l'esperimento principale con BERT. Per completezza, nella cartella **results** sono presenti anche i risultati di RoBERTa, computati offline, così da consentire un confronto diretto tra i due modelli.

Tabella 1 – Metriche globali dei due modelli

Metriche	BERT	RoBERTa
Accuracy	0.8751 ± 0.0103	0.8728 ± 0.0062
Macro-F1	0.8738 ± 0.0104	0.8719 ± 0.0056
ROC-AUC	0.9427 ± 0.0054	0.9480 ± 0.0021
PR-AUC	0.9410 ± 0.0047	0.9478 ± 0.0014
MCC	0.7487 ± 0.0206	0.7459 ± 0.0110
Balanced Accuracy	0.8727 ± 0.0104	0.8723 ± 0.0036

Dal confronto delle metriche globali emerge una leggera differenza tra i due modelli. **BERT** ottiene valori medi leggermente più alti in **Accuracy, Macro-F1 e MCC**, indicando che è più preciso nella classificazione effettiva delle etichette, cioè nel decidere correttamente se un esempio è positivo o negativo. **RoBERTa**, invece, mostra un leggero vantaggio nelle metriche **ROC-AUC** e **PR-AUC**, che misurano quanto bene il modello assegna probabilità coerenti agli esempi, anche se questo non si traduce necessariamente in una classificazione più corretta.

Questi valori sono, comunque, di gran lunga superiori a quelli definiti dalla *baseline* (cfr. § 3.1.1), che presenta un'**Accuracy** e una **Macro-F1** intorno a **0.51**. Ciò conferma che i modelli superano una strategia casuale proporzionata alle classi e riescono a individuare pattern linguistici utili per classificare correttamente il sentiment.

Tabella 2 – Metriche per classi dei due modelli

Classe	Metriche	BERT	RoBERTa
0	Precision	0.8702 ± 0.0120	0.8859 ± 0.0182
	Recall	0.9036 ± 0.0137	0.8787 ± 0.0367
	Support avg	536	536
1	Precision	0.8818 ± 0.0146	0.8613 ± 0.0331
	Recall	0.8417 ± 0.0170	0.8658 ± 0.0297
	Support avg	457	457

Sulla base della Tabella 2, si evince che:

- per la **classe 0 (negativa):** RoBERTa ha precision più alta, quindi commette meno errori quando predice la classe negativa, mentre BERT ha recall più alto, individuando un maggior numero di esempi negativi reali.
- per la **classe 1 (positiva):** BERT ha precision più alta, quindi commette meno errori quando predice la classe positiva, mentre RoBERTa ha recall più alto, individuando un maggior numero di esempi positivi reali.

Dunque, ciascun modello eccelle in aspetti diversi. In termini di equilibrio tra precision e recall per entrambe le classi, RoBERTa appare leggermente più bilanciato, specialmente per la classe 1. Tuttavia, BERT sembra restituire prestazioni complessivamente maggiori sia nelle metriche globali che in quelle per le singole classi, rendendolo più efficace nella corretta classificazione degli esempi positivi e negativi.

Per questo motivo, abbiamo deciso di adottare **BERT come modello di riferimento** per le analisi successive. Il fatto che sia addestrato specificamente su testi in italiano potrebbe aver contribuito alle sue migliori performance rispetto a un modello multilingue come RoBERTa. Dal punto di vista del costo computazionale, BERT si è dimostrato **più efficiente**: gestisce meglio la memoria, richiede **meno risorse GPU** e si addestra **più velocemente**, caratteristiche particolarmente utili per dataset di dimensioni medio-piccole come quello impiegato in questo progetto. I dati[10] confermano questa differenza: con BERT i tempi di computazione quasi si dimezzano e viene utilizzata meno della metà della memoria rispetto a RoBERTa.

4.2 Confronto fra i *seed* di BERT

Analizzando le performance di BERT sui tre diversi *seed*, si osserva che le metriche globali variano leggermente tra le ripetizioni, ma restano tutte elevate (Tabella 3). In particolare, il ***seed* 42** ottiene i valori più alti in Accuracy (0.8892), Macro-F1 (0.8882), PR-AUC (0.9471) e MCC (0.7769), evidenziando le **migliori prestazioni complessive sul test set**. I *seed* 123 e 999 registrano risultati leggermente inferiori ma comunque solidi, con Accuracy rispettivamente di 0.8651 e 0.8711, e Macro-F1 di 0.8640 e 0.8693.

Tabella 3 – BERT: Metriche globali per seed

SEED	Accuracy	Macro-F1	PR-AUC	MCC	Balanced Accuracy
42	**0.8892**	**0.8882**	**0.9471**	**0.7769**	**0.8872**
123	0.8651	0.8640	0.9358	0.7281	0.8634
999	0.8711	0.8693	0.9400	0.7410	0.8674

Per quanto riguarda le metriche di classe (Tabella 4):

- **Classe 0**: il recall è sempre elevato (0.8843-0.9142), indicando che BERT identifica correttamente la maggior parte degli esempi negativi. Anche la precision è alta, anche se di poco inferiore (0.8566-0.8859), mostrando una buona affidabilità del modello nel classificare gli esempi negativi.

- **Classe 1**: la precision risulta alta in tutti i *seed* (0.8613-0.8934), indicando che, quando il modello predice un positivo, sbaglia raramente. Il recall è leggermente più basso (0.8206-0.8621), evidenziando una lieve difficoltà nel catturare tutti gli esempi positivi.

[10] Nella cartella **compute_log**\ del repository.

Tabella 4 – BERT: Metriche di classe per seed

SEED	Class	Precision	Recall	Support
42	0	0.8859	0.9123	536
	1	0.8934	0.8621	457
123	0	0.8681	0.8843	536
	1	0.8613	0.8425	457
999	0	0.8566	0.9142	536
	1	0.8907	0.8206	457

In sintesi, le metriche globali mostrano solo lievi variazioni tra i *seed*, confermando la stabilità e le buone prestazioni di BERT. A livello di classe, la negativa presenta recall costantemente elevato e precision leggermente inferiore e più variabile, mentre la positiva mantiene alta precision a fronte di un recall di poco inferiore. Il **seed 42**, complessivamente il più performante, viene assunto come riferimento per le analisi successive.

4.3 Confronto fra le epoche di *training* di BERT

Stabilito che il miglior *seed* è il 42, analizziamo nel dettaglio l'andamento del training e della validation per ciascuna epoca.

Tabella 5 – Epoche BERT (SEED 42)

Epoca	Train Loss	Val Loss	Val F1
1	0.1965	0.1326	0.8797
2	**0.1147**	**0.1221**	**0.8941**
3	0.0721	0.1345	0.8936
4	0.0442	0.1705	0.8849
5	0.0253	0.1865	0.8901
6	0.0190	0.1962	0.8936

Durante le prime due epoche si osserva una rapida diminuzione della Train Loss, che misura quanto il modello "sbaglia" sui dati di *training*. A questa tendenza, si associano:

- un incremento della Val F1, che misura quanto il modello classifica correttamente sia esempi positivi sia negativi sui dati di validazione, raggiungendo il valore massimo (**0.8941**) **alla seconda epoca**;
- un abbassamento della Val Loss, che misura quanto il modello sbaglia sui dati di validazione, cioè su esempi non visti durante il *training*.

Dalla terza epoca in poi, la **Val F1** rimane sostanzialmente stabile, mentre la **Val Loss** cresce leggermente, indicando un inizio di ***overfitting***. L'**early stopping** seleziona quindi come modello finale quello della seconda epoca, quando il modello aveva raggiunto il miglior equilibrio tra accuratezza sui dati di *training* e capacità di generalizzazione sui dati di *validation*.

4.4 Analisi degli errori di BERT: l'impatto dell'ironia sui falsi positivi

Durante l'analisi dei risultati, abbiamo posto l'attenzione su quei testi che non sono stati classificati correttamente da BERT, ovvero i **falsi positivi** e i **falsi negativi**.

Dall'esame delle metriche **FNR** (*False Negative Rate*) e **FPR** (*False Positive Rate*), calcolate dalle matrici di confusione per ciascun *seed*, è emerso che il modello ha più difficoltà nel riconoscimento degli esempi positivi: in tutte le matrici, la FNR risulta più elevata della FPR. Questo comportamento potrebbe derivare dallo sbilanciamento delle classi nel dataset (cfr. § 3.1.4), che vede più rappresentata la classe dei negativi.

Da un punto di vista qualitativo, l'analisi dei falsi positivi ha fatto emergere un fenomeno interessante:

Sentence	**True label**	**Predicted**
È confortante sapere che il PD appoggerà il governo Monti nella realizzazione del programma del PdL! #portaaporta #orabasta #acasa	0	1
La nuova sala stampa di Palazzo Chigi by Mario Monti è degna della sala funerali del Cimitero Verano!	0	1
Vorrei conoscere il genio che ha partorito questo spot. "Noi di Mediolanum siamo al vostro fianco" e invece di andare incontro alle esigenze del cliente e ad assisterlo in questo momento particolare, magari pensando a qualche agevolazione finanziaria, loro invece che fanno? Te chiedono i soldi! Complimenti, proprio un bel modo di "coccolare" il cliente.	0	1

Da questi esempi emerge che alcuni falsi positivi derivino dalla presenza di **ironia** o **sarcasmo** nei testi. Infatti, quando viene impiegata l'ironia, si scelgono tipicamente parole dal significato letterale positivo per comunicare un'opinione negativa, e viceversa[11]. Gli esempi mostrano esattamente come espressioni positive ("confortante", "degna", "genio") possano veicolare **un sentiment negativo implicito**. Come sostenuto da Van Hee *et al.* (2018), la componente ironica – molto diffusa nei social media – porterebbe i classificatori a **invertire la polarità reale del sentiment** per la mancanza di "common sense" e conoscenza pragmatica. Poiché i modelli computazionali si basano esclusivamente sui dati di addestramento e non possiedono conoscenza del mondo, interpretano con difficoltà un significato implicito.

Per ridurre gli errori dovuti all'ironia e migliorare l'accuratezza del modello, si potrebbero considerare alcune strategie. Una prima soluzione consiste nell'integrare un **sistema di Irony Detection**, capace di rilevare segnali tipici dell'ironia – come contrasti lessicali, punteggiatura marcata, virgolette o espressioni enfatiche – così da supportare il modello nell'assegnazione corretta della polarità[12].

[11] Hernańdez Farías *et al.,* 2016.
[12] Hernańdez Farías *et al.,* 2015.

Un'altra possibilità è definire un **sentiment prototipico** per determinate situazioni, così da individuare rapidamente i testi il cui sentiment si discosta da quello atteso[13].

Infine, si potrebbe considerare un **approccio multi-prompt**, come proposto recentemente da Magnini *et al.* (2025) con il benchmark **Evalita-LLM**: per l'analisi del *sentiment* su tweet italiani, vengono utilizzate sei diverse formulazioni di prompt – domande dirette, descrizioni seguite da quesiti o affermazioni che richiedono risposta – per testare la coerenza del modello di fronte a diverse modalità di interrogazione[14].

Conclusioni

In questo progetto abbiamo affrontato il task di classificazione binaria del sentiment su testi italiani provenienti dai social media (Twitter, Facebook e YouTube), confrontando due modelli pre-addestrati su più seed per verificarne la robustezza. Dall'analisi delle metriche globali e delle performance per ciascuna classe, abbiamo deciso di adottare come riferimento **BERT**, che ha mostrato prestazioni complessivamente superiori, richiedendo al contempo un minor consumo di GPU e beneficiando del fatto di essere stato pre-addestrato specificamente su testi in lingua italiana.

BERT ha restituito una **Macro-F1** e un'**Accuracy** medie di circa **0.87**, valori decisamente superiori alla *baseline* casuale (circa 0.51), dimostrando una buona capacità di generalizzazione su dati mai visti.

Un confronto interessante emerge dai risultati ottenuti dal **MilaNLP Group**[15]: il modello **UmBERTo**, addestrato in tre configurazioni diverse (su FEEL-IT[16], sul train set di SENTIPOLC16 e su entrambi) è stato poi valutato sul test set di SENTIPOLC16, ottenendo la performance migliore con Macro-F1 di 0.81 e Accuracy di 0.84. Il nostro test set è stato costruito tramite uno split stratificato del dataset ***data*** combinato (FEEL-IT + SENTIPOLC16 + MultiEmotions-IT), includendo quindi anche esempi parzialmente sovrapponibili a quelli su cui UmBERTo era stato valutato. Le performance di BERT (**Macro-F1 e Accuracy ≈ 0.87**), superiori a quelle di UmBERTo, confermano la sua efficacia nel gestire testi italiani recenti e diversificati, provenienti dai social media. Anche **ROC-AUC e PR-AUC (≈ 0.94)** indicano buone capacità predittive su dataset leggermente sbilanciati.

I risultati ottenuti hanno confermato quanto il modello sia solido anche su dati nuovi e vari. Speriamo che possano essere in futuro un utile spunto per **ulteriori studi ed esperimenti sulla Sentiment Analysis in lingua italiana.**

[13] Van Hee *et al.*, 2018.
[14] Magnini et al., 2025.
[15] https://huggingface.co/MilaNLProc/feel-it-italian-sentiment
[16] Come fatto da noi in fase di pre-processing dei dati, le emozioni del dataset FEEL-IT sono state ricondotte dal MilaNLP Group a due categorie: la gioia è stata mappata nella classe positiva, mentre rabbia, paura e tristezza sono state aggregate nella classe negativa.

Riferimenti

Documentazione:

- https://docs.python.org/3/library/ – Python Standard Library (os, random) documentation
- https://numpy.org/doc/ – NumPy documentation
- https://pandas.pydata.org/docs/ – Pandas documentation
- https://pytorch.org/docs/stable/index.html – PyTorch documentation (torch, torch.nn, torch.utils.data, torch.optim)
- https://torchmetrics.readthedocs.io/ – TorchMetrics documentation
- https://huggingface.co/docs/datasets – Hugging Face Datasets documentation
- https://huggingface.co/docs/transformers – Hugging Face Transformers documentation
- https://huggingface.co/docs/accelerate – Hugging Face Accelerate documentation
- https://scikit-learn.org/stable/documentation.html – scikit-learn documentation
- https://matplotlib.org/stable/ – Matplotlib documentation
- https://seaborn.pydata.org/ – Seaborn documentation
- https://research.google.com/colaboratory/ – Google Colab documentation

Dataset e articoli correlati:

Feel-IT:

- Federico Bianchi, Debora Nozza, and Dirk Hovy. 2021. FEEL-IT: Emotion and Sentiment Classification for the Italian Language. In *Proceedings of the Eleventh Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 76–83, Online. Association for Computational Linguistics. https://aclanthology.org/2021.wassa-1.8/
- https://github.com/MilaNLProc/feel-it - dataset ("data 1"). Importante: per avere accesso al dataset, completare questo form e arriverà una notifica via e-mail contenente il file .csv.
- https://milanlproc.github.io/ MilaNLP Group.
- https://huggingface.co/MilaNLProc/feel-it-italian-sentiment

MultiEmotions-It:

- Sprugnoli, R., *MultiEmotions-it: un nuovo dataset per la polarità dell'opinione e l'analisi delle emozioni per l'italiano*, in Atti della settima conferenza italiana di linguistica computazionale (CLiC-it 2020), (online, 01-03 marzo 2021), Accademia University Press, Torino 2020: 402-408. https://books.openedition.org/aaccademia/8910
- https://github.com/RacheleSprugnoli/Esercitazioni_SA/tree/master/dataset - dataset ("data 2")

Sentipolc 2016 (train e test):

- Barbieri, F., Basile, V., Croce, D., Nissim, M., Novielli, N., Patti, V., et al. (2016). *Overview of the Evalita 2016 SENTIment POLarity Classification Task.* CEUR Workshop Proceedings, 1749. https://huggingface.co/datasets/evalitahf/sentiment_analysis - dataset splittato (“data 3” e “data 4”)
- https://www.evalita.it/campaigns/evalita-2016/tasks-challenge/sentipolc/ - EVALITA 2016

Altro

- Hernández Farías, D.I., Benedí Ruiz, J.M., Rosso, P. (2015). *Applying basic features from sentiment analysis on automatic irony detection.* In: Pattern Recognition and Image Analysis: 7th Iberian Conference, IbPRIA 2015, Santiago de Compostela, Spain, June 17- 19, 2015, Proceedings. Springer International Publishing. 337-344. https://doi.org/10.1007/978-3-319-19390-8_38
- Hernańdez Farías D.I., Patti V., Rosso P. (2016). *Irony Detection in Twitter: The Role of Affective Content.* ACM Trans. Internet Technology 16 (3), Article 19 (August 2016), 24 pages. https://doi.org/10.1145/2930663
- Magnini B., Zanoli R., Resta M., Cimmino M., Albano P., Madeddu M., Patti V. (2025). *Evalita-LLM: Benchmarking large language models on Italian.* https://doi.org/10.48550/arXiv.2502.02289
- Manning C. D., Schütze H. (1999). *Foundations of statistical natural language processing*. MIT Press.
- Van Hee C., Lefever E., Hoste V. (2018). *We Usually Don’t Like Going to the Dentist: Using Common Sense to Detect Irony on Twitter.* Computational Linguistics 44 (4): 793–832. https://doi.org/10.1162/coli_a_00337