# Human vs Machine Text Classifier

**Marta Cozzini**

marta.cozzini@studio.unibo.it

**Abstract**

Il progetto implementa un classificatore in grado di distinguere tra testi scritti da esseri umani e testi generati da modelli di linguaggio automatizzati (come GPT). La prima parte del progetto è dedicata all'estrazione di caratteristiche rilevanti, utilizzate dal modello per prendere le sue decisioni. Per questa estrazione viene impiegata la libreria di NLP SpaCy. Nella fase finale del progetto, viene utilizzata la libreria LIME per esplorare e interpretare il comportamento del modello, migliorando la comprensione delle decisioni prese dal classificatore.

**Introduzione**

Il progetto nasce dall'idea di mettere in pratica le tecnologie apprese durante il corso e di combinarle con le conoscenze di machine learning acquisite in un corso dell'ultimo semestre.

L'idea di sviluppare un classificatore capace di distinguere tra testi scritti da esseri umani e testi generati da macchine nasce invece da un seminario a cui ho partecipato, durante il quale si cercava di spiegare quali fossero alcune possibili caratteristiche linguistiche rilevanti per distinguere tra le due tipologie di testo.

Per addestrare il mio classificatore ho usato solo alcune delle features presentate durante il seminario, come il numero di token e la varietà lessicale. Inizialmente, avevo pianificato di includere anche altre caratteristiche, come la complessità sintattica, analizzando il numero di frasi per testo, il numero di verbi e connettivi e la struttura sintattica delle singole frasi. Tuttavia, il mio dataset non conteneva testi abbastanza lunghi da consentire un'analisi di questo tipo. Il dataset analizzato è stato reperito sulla piattaforma Kaggle e contiene 50000 testi, di cui 25000 scritti da esseri umani e 25000 generati da macchine.

Per motivi di memoria e di spazio ho deciso di considerare solo 15mila testi del dataset, 7500 umani e 7500 prodotti da macchine. Come già evidenziato purtroppo la lunghezza dei testi e la qualità del dataset non mi hanno permesso di considerare alcune features rilevanti. Le features che, invece, ho deciso di considerare sono risultate rilevanti, ma con una differenza di valori minima tra le due tipologie di testi. Pertanto, ritengo che sarebbe interessante replicare l'analisi utilizzando un dataset più completo, con testi più lunghi e appartenenti a diverse categorie testuali.

Il progetto è suddiviso in tre parti facilmente riconoscibili: una prima parte dedicata all'estrazione delle features rilevanti, una seconda parte di addestramento del modello e una terza parte di analisi delle decisioni prese dal classificatore.

**Pre processing e Feature extraction**

Dopo una prima parte di pre processing e pulizia del testo, si è proceduto all'estrazione delle caratteristiche rilevanti. A priori, sono state selezionate le caratteristiche linguistiche che si riteneva fossero rilevanti per distinguere tra testi scritti da esseri umani e testi generati da macchine. Le features considerate sono state: il numero di token per testo, il numero di frasi per testo, la ricchezza lessicale di ogni testo calcolata attraverso la metrica *type token ratio* (il rapporto tra il numero di parole uniche *tipi* e il numero totale di parole *token* nel testo), il numero di verbi e la loro ricchezza lessicale, anche questa calcolata attraverso il *type token ratio,* e infine il numero di connettivi per frase. Si è poi deciso di ignorare quest'ultima feature, poiché il valore dei connettivi risultava pari a zero in quasi tutti i testi.

Questo aspetto era evidente già dal numero medio di frasi, che per entrambi i tipi di testo (umani e generati da macchine) superava di poco l'1. Per confrontare le features sono stati creati due dataset distinti: uno contenente esclusivamente i testi umani e l'altro con i testi generati da macchina. Una volta estratte tutte le features rilevanti, i dataset sono stati uniti per la parte di addestramento del classificatore.

Per confrontare visivamente i valori delle features rilevanti tra i testi umani e quelli generati da macchine, sono stati creati dei grafici (barplot) basati sulla media delle features calcolate. I grafici includono il confronto tra il numero medio di token, il numero medio di frasi, il type-token ratio (TTR) medio, la frequenza media dei verbi e il TTR medio dei verbi.

I risultati di queste analisi sono archiviati nella cartella *results* del progetto, suddivisi in due sottocartelle: una contenente i grafici e l'altra con i valori medi delle features, organizzati in dataframe o semplici file .txt.

I risultati mostrano che le features considerate sono state effettivamente rilevanti per distinguere tra le due tipologie di testo, sebbene i valori ottenuti siano risultati molto simili. Questo può essere attribuito alla qualità del dataset utilizzato. Per esempio, il confronto tra il valore medio del TTR dei verbi ha rivelato una media di 0.825 per i testi generati da macchine e di 0.875 per i testi scritti da esseri umani. Un TTR medio più alto indica una maggiore varietà lessicale, il che conferma la nostra aspettativa che i testi umani presentino una varietà lessicale superiore, anche per quanto riguarda i verbi. Tuttavia, i valori sono molto vicini, un fatto che probabilmente dipende dalle caratteristiche del dataset.

Il dataset finale, che è stato utilizzato per l'addestramento del classificatore, conteneva le seguenti features: il numero di frasi per testo, il numero di token, il type-token ratio (TTR), il numero di verbi e il TTR dei verbi.

**Addestramento del modello**

Per addestrare il classificatore, è stato utilizzato il dataset iniziale di 15.000 testi, arricchito con le colonne contenenti le features numeriche calcolate nella prima parte del progetto. Il nostro dataset era quindi suddiviso in due categorie principali: testo e features numeriche. Per prima cosa, si è separato il testo (X_train_text) dalle features numeriche (X_train_metrics) e dalle etichette target (y), che rappresentano la variabile da predire, ossia la colonna "source" che indica la provenienza dei testi (umano o AI). Successivamente, i dati sono stati divisi in training set, test set e validation set.

Si è proceduto quindi al pre-processing delle caratteristiche numeriche, che sono state normalizzate tramite la libreria StandardScaler di scikit-learn. La normalizzazione è importante perché permette di portare tutte le variabili numeriche sulla stessa scala, evitando che il modello venga influenzato da variabili con valori più grandi o piccoli. Per il pre-processing del testo, è stata invece utilizzata la libreria TensorFlow, che consente di trasformare il testo in sequenze numeriche. La libreria Tokenizer di Keras è stata impiegata per convertire le parole in indici numerici, assegnando un numero unico a ciascuna parola nel vocabolario. Le sequenze ottenute sono state quindi uniformate alla stessa lunghezza (80), operazione essenziale per poter alimentare i dati nel modello. Le sequenze di testo sono state poi convertite in array NumPy per facilitarne l'elaborazione successiva.

Una volta completato il pre-processing, si è passati alla creazione del modello. Il modello è un modello multi-input che prende in input sia il testo che le caratteristiche numeriche. Come abbiamo già visto, il modello riceve in input sequenze di testo in cui ogni parola è rappresentata tramite un indice numerico, che corrisponde a una parola nel vocabolario del modello. Questi indici numerici vengono utilizzati per ottenere un embedding della parola tramite un Embedding Layer. Questo layer è una componente chiave del modello, poiché mappa ogni indice di parola a un vettore di dimensione embedding_dim, che rappresenta la parola in uno spazio continuo e denso, catturando la sua semantica e il contesto. Il flusso del testo viene gestito da un LSTM (Long Short-Term Memory), un tipo di rete neurale particolarmente adatto per l'elaborazione di sequenze temporali

come il testo. Dopo l'LSTM, viene applicato un layer di Dropout (0.5) per ridurre il rischio di overfitting durante l'addestramento. Come abbiamo già visto il modello prende in input anche un insieme di features numeriche. I due flussi (testo e numerico) vengono concatenati in un unico flusso, che combina le informazioni estratte dalle sequenze di testo e dalle features numeriche. Un layer denso con 64 unità e funzione di attivazione ReLU è utilizzato per apprendere le rappresentazioni combinate di testo e features numeriche. Infine, viene applicato un altro layer di Dropout (0.5) per evitare ulteriori rischi di overfitting.

L'output del modello è costituito da un singolo neurone con attivazione *sigmoid*, che restituisce una probabilità di classificazione per determinare se il testo è stato scritto da un essere umano o è stato generato da una macchina.

Il modello viene compilato utilizzando l'ottimizzatore *Adam* e la funzione di perdita *binary_crossentropy* (poiché si tratta di un problema di classificazione binaria). Inoltre, viene calcolato il peso delle classi per gestire il problema di sbilanciamento delle classi (*class imbalance*), qualora le due categorie non siano equamente distribuite. Il modello viene quindi allenato utilizzando i dati di training. Durante l'allenamento, il modello esegue 30 epoche con un batch size di 32, monitorando le prestazioni sul validation set. Per ottimizzare l'addestramento, vengono utilizzate le callback *EarlyStopping* e *ReduceLROnPlateau*. La prima interrompe l'allenamento se la perdita non migliora, evitando l'overfitting. La seconda riduce il learning rate nel caso in cui le prestazioni non migliorino per un certo numero di epoche, contribuendo a un processo di addestramento più efficace.

Una volta completato l'addestramento, si procede alla valutazione del modello. La prima operazione consiste nell'eseguire il metodo *.evaluate()* sui dati di test, che restituisce il valore della perdita (loss) e dell'accuratezza (accuracy) del modello. In questo caso, il modello ha ottenuto un'accuratezza del 0,8030 e una perdita di 0,42. La perdita misura quanto il modello si discosti dalla previsione corretta, mentre l'accuratezza rappresenta la percentuale di predizioni corrette rispetto al totale delle predizioni. Viene poi generata la matrice di confusione, che visualizza le performance del classificatore confrontando le etichette predette con quelle reali. Un altro strumento utile per analizzare le prestazioni del modello è il classification report, che fornisce metriche come precisione, recall, F1-score e supporto per ciascuna classe. La precisione indica la percentuale di previsioni corrette per una determinata classe tra tutte le previsioni fatte per quella classe. Il Recall (Sensibilità) misura la percentuale di veri positivi che sono stati correttamente identificati dal modello. Il supporto indica il numero di istanze effettive di ogni classe nel dataset di test, aiutando a comprendere la distribuzione delle classi nel dataset, mentre F1-score è la media armonica della precisione e del recall, una misura complessiva delle performance del modello. Tutti i risultati di queste metriche e, in generale, delle performance del modello sono salvati nella cartella *results* del progetto, nella sottocartella *performance*. In generale, possiamo osservare che i valori di quasi tutte le metriche si aggirano intorno a 0,80 indicando che il modello classifica in modo abbastanza accurato sia i testi umani che quelli generati dalla macchina.

**Analisi con LIME**

Nell'ultima parte del progetto si utilizza la libreria LIME (Local Interpretable Model-agnostic Explanations) per spiegare le predizioni del modello di classificazione sviluppato, basato su input testuali e numerici. L'obiettivo principale è fornire interpretazioni locali delle decisioni del modello, evidenziando le caratteristiche più influenti nelle predizioni. La funzione *preprocess* trasforma i testi in sequenze numeriche utilizzando un tokenizer pre-addestrato e normalizza le metriche numeriche usando uno scaler, restituendo due array pronti per il modello. La funzione *predict_proba_lime* utilizza questi dati preprocessati per calcolare e restituire le probabilità delle classi (ai e human) in formato comprensibile per LIME. Possiamo quindi procedere alla spiegazione delle predizioni del nostro modello. Per prima cosa, viene inizializzato un oggetto *LimeTextExplainer* per interpretare le

predizioni relative ai testi, con le classi AI e human. Successivamente, viene selezionato un esempio di testo dal dataset, e la funzione *explain_instance* identifica le 3 caratteristiche più influenti per la predizione su questo testo. Questo processo viene eseguito per due esempi, e i risultati vengono salvati in un dataframe nella sottocartella *lime analysis* della cartella *results*.

Si passa poi alla spiegazione delle feature numeriche. Un altro oggetto, LimeTabularExplainer, viene inizializzato per analizzare l'influenza delle metriche numeriche. Un esempio del dataset di test (*X_test_metrics*) viene selezionato e analizzato tramite *explain_instance*. La funzione restituisce le 5 feature numeriche più rilevanti per la predizione del modello insieme ai loro pesi. Anche in questo caso, eseguiamo lo stesso procedimento per due esempi distinti. Analizzando i dataframe dei risultati ottenuti con LIME, osserviamo che le feature numeriche hanno un peso molto basso nel determinare le decisioni del modello. Questo risultato potrebbe essere attribuito alla qualità del dataset, che potrebbe aver prodotto valori di feature meno significativi o scarsamente distintivi per le classi considerate, come era stato notato inizialmente.

**Conclusione**

Il classificatore ha mostrato performance abbastanza buone, con un F1 score di 0,807 per i testi generati da AI e di 0,79 per i testi scritti da esseri umani. Anche tutte le altre metriche riportate nel classification report si attestano intorno a questi livelli, il che significa che il modello classifica correttamente circa 8 testi su 10. Tuttavia, non possiamo considerarci completamente soddisfatti, poiché, come emerge dall'analisi effettuata con la libreria LIME, le feature estratte nella prima parte del progetto, che pensavamo fossero rilevanti, hanno avuto un impatto davvero limitato sulle decisioni del classificatore. Questo potrebbe essere dovuto alla qualità del nostro dataset, ma anche alla possibile scelta errata delle feature stesse. Per il futuro, sarebbe interessante applicare il modello a un dataset più ampio e variegato, con testi di qualità superiore e appartenenti a diverse categorie testuali. Inoltre, sarebbe utile approfondire la letteratura per identificare altre feature potenzialmente rilevanti.

**References**


Pandas documentation, https://pandas.pydata.org/docs/

Spacy documentation, https://spacy.io/usage/spacy-101

Matplotlib documentation, https://matplotlib.org/stable/users/index.html

Numpy documentation, https://numpy.org/doc/

Scikit-learn documentation, https://scikit-learn.org/

TensorFlow documentation, https://www.tensorflow.org/

Keras documentation, https://keras.io/

Lime documentation, https://lime-ml.readthedocs.io/

Seaborn documentation, https://seaborn.pydata.org/