Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Toxic Comment Classification

Authors:
Sandro Erba - 856327
Alice Hoa Galli - 856648
Elia Gaviraghi - 869493

1 febbraio 2024

Sommario

Il presente studio esamina in dettaglio la classificazione di commenti tossici, concentrandosi su una valutazione comparativa tra due approcci di apprendimento automatico: *Recursive Neural Networks* (RNN) e *Transformer BERT*. Attraverso un'analisi approfondita del dataset, sono state identificate sfide legate al riconoscimento della tossicità nei commenti online, presentando il confronto dei modelli e delle tecniche sperimentate tramite metriche oggettive. Pro e contro di ogni modello vengono successivamente studiati ed eviscerati.

1 Introduction

I social media, veloci e accessibili, possono diventare terreno fertile per scambiare informazioni, ma la loro espansione li rende anche vulnerabili. Dibattiti online spesso sfociano in scontri e commenti tossici, minacciando la qualità delle discussioni. Questi commenti possono assumere varie forme, dall'offensivo all'odioso. Le piattaforme online devono affrontare la sfida di garantire un dibattito equo, ma spesso si trovano a dover limitare i commenti o chiudere intere comunità. [1]

Perciò nel 2017 Jigsaw e Google hanno aperto la competizione nella quale si cerca un modello che sia in grado di rilevare diversi tipi di tossicità, come minacce, oscenità, insulti e odio basato sull'identità. Questo tipo di problema prende il nome di TCC (*Toxic Comment Classification*). Per questo problema è stato messo a disposizione un set di dati di commenti provenienti dalla talk page di Wikipedia. Il nostro approccio al problema è stato quello di suddividere lo studio e l'implementazione dei modelli, in modo che ognuno sviluppasse le proprie soluzioni nella direzione che ritenesse più opportuna pur partendo da una base comune. I modelli utilizzati consistono in una RNN e un transformers di tipo BERT. Le RNN sono architetture più conosciute e assodate in letteratura; nonostante possano essere considerate "passate" e limitate rispetto all'analisi di sequenze lunghe, rispetto ai transformers sono molto più semplici e veloci in comprensione e sviluppo. Nel secondo modello, più recente, i transformers preaddestrati su grandi quantità di dati possono essere utilizzati come punto di partenza per task specifici. Entrambi i modelli utilizzano un approccio bidirezionale, per considerare il contesto sia a sinistra che a destra di ciascuna parola, particolarmente utile per i *tasks* di NLP.

2 Datasets

Il dataset della *Toxic Comment Classification Challenge* [2] contiene 159571 commenti provenienti da Wikipedia, i quali sono stati etichettati da valutatori umani nelle diverse tipologie di tossicità. I tipi di tossicità presentati sono 6: *toxic*, *severe_toxic*, *obscene*, *threat*, *insult* e *identity_hate*.

Per semplificare, in questo report si definiscono come *toxic* i commenti con almeno una *label* impostata a 1, mentre quelli *non-toxic* hanno tutte le *label* a 0. Lo sbilanciamento è evidente, con i commenti *non-toxic* che rappresentano l'89.93% dell'intero dataset. Questo è ben visibile in Figura 1.

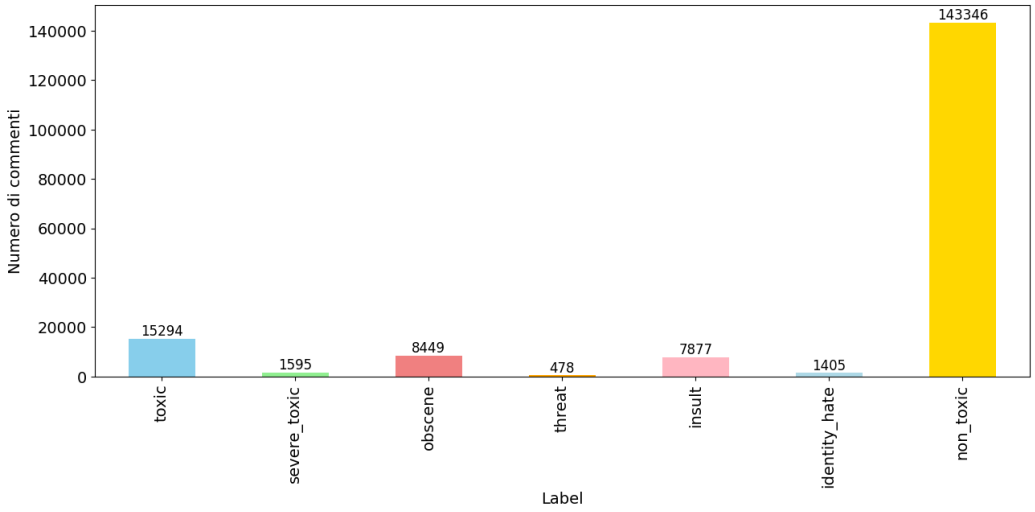

Figura 1: distribuzione delle *label* nel dataset

Vi è un secondo squilibrio nelle etichette dei commenti tossici, evidenziato dalla presenza di soli 478 commenti con etichetta *threat* attiva, rispetto ai 15294 commenti etichettati come *toxic*. È importante notare che l'uso del termine *classe* nel contesto del problema è utile per la comprensione, ma leggermente improprio. Nel trattamento del problema, si tratta effettivamente di una classificazione *multi-etichetta*, in cui la previsione coinvolge singolarmente le sei diverse etichette binarie.

Le operazioni comuni sul dataset includono la pulizia di diversi pattern tramite espressioni regolari (*regex*) e la conversione di tutti i commenti in minuscolo, con l'obiettivo di ridurre la complessità del dataset. In particolare,

le espressioni regolari sono state utilizzate per rimuovere termini con diversa sintassi ma identica semantica (ad esempio, sostituendo *'ve* con *have*) e per eliminare parti di testo "sporco" e insignificante, come la keyword `\n` derivata dalla formattazione del testo o i link presenti nei commenti. Queste tecniche uniformano i testi, riducendo la varietà dei *token* e la dimensione complessiva del vocabolario. Ciò facilita la cattura dei pattern rilevanti da parte dei modelli, migliorando l'efficienza computazionale durante l'addestramento.

Oltre all'operazione di pulizia dei dati appena descritta sono stati presi in analisi altri tre processi di pulizia dei dati che non vengono eseguiti per le scarse performance raggiunte. Tra questi si era pensato di eliminare tutti i segni di punteggiatura, e.g., virgole, punti. La seconda operazione è la *lemmatization*, che prevede la sostituzione di tutte le parole presenti in ogni commento con il corrispondente lemma, ovvero con la forma canonica che si andrebbe ad usare per cercare la parola originaria all'interno di un dizionario. La terza operazione considerata è stata quella di rimozione delle cosiddette *stop-word*, i.e., articoli, pronomi, preposizioni.

Gli sbilanciamenti nel dataset hanno portato all'esame di tre approcci diversi, non necessariamente alternativi e variabili in base al modello selezionato: *random undersampling* dei commenti con tutte le etichette a 0 (*non-toxic*), oversampling tramite data augmentation e calcolo dei pesi basati sulle frequenze.

Il *random undersampling* dei commenti *non-toxic* coinvolge la diminuzione casuale di tali commenti. Questa tecnica si è rivelata utile e parzialmente efficace solo con alcuni modelli di transformers, dove l'addestramento sull'intero dataset risultava computazionalmente oneroso.

La *data augmentation*, sperimentata nel tentativo di bilanciare il numero di commenti etichettati come *threat* e *severe_toxic*, ha prodotto risultati limitati. Ispirandosi alla letteratura [3], l'aumento dei commenti coinvolge la sostituzione di alcune parole con sinonimi dal thesaurus WordNet, utilizzando la libreria *nlpaug*[4].

Il calcolo dei pesi in base alle etichette è stato l'unico metodo effettivamente implementato in ogni modello: in fase di addestramento i pesi vengono passati come parametro, cosicché il modello ne tenga conto durante l'addestramento. Questo metodo è immediato e non comporta alcuna modifica del dataset in esame.

3 The Methodological Approach

Nel contesto del *Natural Language Processing - NLP*, per poter consentire ai modelli di *machine learning* di ricevere in ingresso i commenti del dataset, il passaggio fondamentale consta nel processo di *tokenization*.

La *tokenization*[5]. è il processo di suddivisione di testi, come paragrafi o frasi, in unità più piccole chiamate *token*. Questa fase facilita l'assegnazione di significato a ciascuna unità. L'obiettivo è rappresentare i commenti del dataset come sequenze di *token*, creando un vocabolario composto da tutti i *token* presenti nel corpus di testo. Ogni *token* è mappato a un numero intero univoco e il vocabolario è ordinato in base alla frequenza delle parole nel dataset. Dopo la pulizia dei pattern, il vocabolario risultante conteneva inizialmente $187,000$ parole. Per ridurre il volume di dati nelle fasi successive, è stata limitata la grandezza del vocabolario a meno delle prime $30,000$ parole più frequenti, adattando il parametro alle capacità del modello utilizzato.

Per completare questa fase, si esegue la *data encoding*, dove si converte ogni *token* in una sequenza di numeri [6]. Infine, poiché i modelli utilizzati richiedono *input* di dimensione fissata, si esegue un'operazione di *data padding*, dove si aggiungono dei *token* di *padding* alle sequenze per raggiungere la lunghezza massima impostata di 200: questo valore è stato scelto poiché il 50-esimo percentile dei commenti ha una lunghezza inferiore a 127 e il 75-esimo percentile è inferiore a 270. Questo parametro verrà poi adattato secondo le esigenze del modello considerato.

Prima di iniziare l'esposizione dei modelli, si noti che la metrica di *accuracy*, tendenzialmente alta per tutti i modelli in questo task, potrebbe trarre in inganno sull'andamento delle performance. Utilizzando un classificatore che predice sempre 0 per ogni etichetta, dunque non risolve il riconoscimento dei commenti tossici, si ottiene il 90.08% di *accuracy*: questo è dovuto al forte sbilanciamento del dataset verso i commenti con tutte etichette 0. Per questo motivo le metriche successivamente presentate sono state di vitale importanza.

3.1 RNNs

Le *Recursive Neural Networks* (RNNs) sono un tipo di rete neurale che facilita la connessione ricorsiva tra gli strati, consentendo di catturare dipendenze a lungo termine. Sono state ampiamente impiegate per il riconoscimento di pattern nel testo, e a differenza dei transformers, le RNNs risultano

meno complesse da progettare e comprendere, oltre che essere più veloci nell'addestramento.

Per la valutazione e il confronto delle diverse architetture, come funzione di perdita si è scelta la *binary cross entropy*, la quale è comunemente utilizzata nei problemi di classificazione binaria.

La sfida iniziale di muovere i primi passi nel dominio del problema è stata superata attraverso la consultazione di diversi notebook partecipanti alla challenge su Kaggle[7][8]. Particolarmente significativa è stata la guida all'implementazione di un modello estremamente semplice [9], che ha contribuito a una migliore comprensione dei motivi dietro alcuni layer specifici.

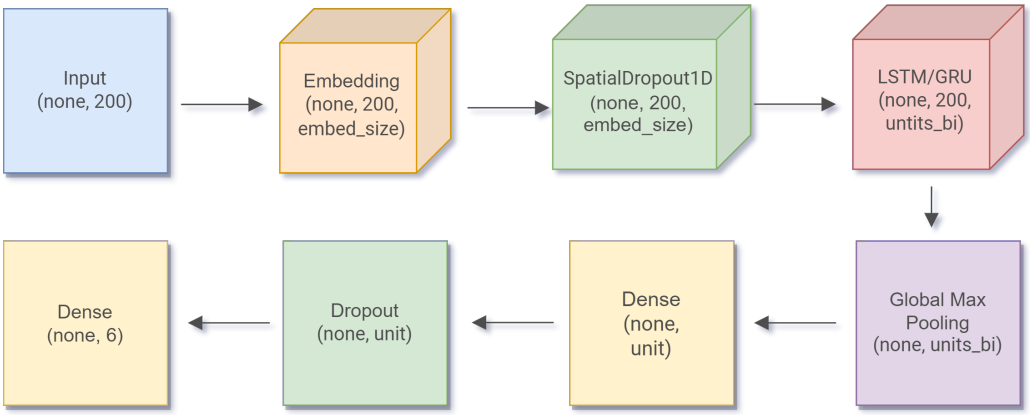

Figura 2: Strati della RNN sviluppata

Acquisendo maggiore dimestichezza con il dominio del problema, è iniziata l'implementazione di una RNN personale in figura 2, prendendo spunto da alcune *baseline*[9][10]. Le modifhe apportate a queste *baseline* sono state ponderate per equilibrare la complessità al problema in esame, senza aggiungere complessità senza l'apporto di un grande miglioramento.

La rete costruita inizia con uno strato di *input* standard, dove la lunghezza delle sequenze in input è fissata a 200 durante la fase di tokenization. Lo strato di *Embedding* converte le sequenze in input in vettori densi di dimensione *embed_size*, catturando relazioni semantiche tra le parole. Un valore più alto per *embed_size* generalmente migliora le performance, ma aumenta il tempo computazionale. Embedding preallenati, come *glove.840B.300d* e *glove.twitter.27B.200d*[11][10], sono stati esplorati per arricchire l'apprendimento con rappresentazioni semantiche più complesse. Tuttavia, in li-

nea con quanto preannunciato [12] l'utilizzo di questi embedding, incluso *glove.840B.300d*, non ha migliorato le performance del modello.

Segue uno strato di *SpatialDropout1D*, comunemente adottato dopo lo strato di *embedding* per prevenire l'*overfitting*. A differenza del generico *Dropout*, è progettato per trattare dati sequenziali a 1 dimensione, eliminando casualmente vettori di *embedding* e mantenendo quelli fortemente correlati.

L'architettura centrale utilizza una LSTM bidirezionale, che analizza la sequenza in input sia dall'inizio che dalla fine, combinando le informazioni da entrambe le direzioni. La scelta tra LSTM e GRU è stata effettuata durante l'esplorazione degli iperparametri, tenendo conto dei vantaggi e svantaggi di entrambi. I parametri chiave, come il numero di unità nei *layer* e l'*embed_ size*, sono stati selezionati in base alle risorse computazionali e alle prestazioni desiderate. I valori ottimali di iperparametri, come il *dropout* e il numero di unità, sono stati determinati durante la fase di *tuning* del modello, utilizzando come riferimento il modello più performante. Il parametro `return_sequences=True` è impostato per far sì che il layer restituisca una sequenza di output per ogni passo temporale. Il parametro `recurrent_dropout` è stato escluso fin dall'inizio, poiché Tensorflow non fornisce l'ottimizzazione *cuDNN* quando questo parametro è attivo.

Il *GlobalMaxPooling1D*, posizionato dopo la LSTM, è una scelta comune per ottenere una rappresentazione compatta e informativa della sequenza temporale, evidenziando le caratteristiche più significative. Alcuni codici online propongono architetture che concatenano questo *layer* con un *GlobalAveragePooling1D*, cercando di conservare caratteristiche rilevanti sia dalle parti più salienti (*GlobalMaxPooling1D*) che dalla distribuzione media delle caratteristiche (*GlobalAveragePooling1D*). Tuttavia, sperimentazioni hanno dimostrato che questa modifica non migliora, ma addirittura deteriora leggermente le prestazioni di questo modello, considerando l'incremento di complessità e parametri.

A seguire poi uno strato densamente connesso, uno strato di *Dropout* per contenere l'overfitting e infine l'output della rete, il quale è stato costruito con 6 unità, come le etichette del dataset, e una sigmoide come funzione di attivazione, permettendo di ottenere una probabilità in output per ogni etichetta.

Dopo l'esplorazione dell'architettura, è iniziata una fase di (*exploitation*), con l'obiettivo di massimizzare le performance del modello attraverso il *tuning* degli iperparametri mediante una *random search*, utilizzando la libreria Keras

Tuner[13]. Il training per ogni singolo modello è rapido: servono solo 3 epoche, con un `batch_size` di 32 per convergere al minimo valore di loss.

Tabella 1: Parametri coinvolti nel *tuning*

	Value range	**Best model**
Embed size	[100, 400]	250
Spatial dropout	[0.1, 0.6]	0.5
Recurrent layer	LSTM, GRU	LSTM
Units rec layer	[60, 300]	120
Dropout rec_layer	[0.1, 0.6]	0.3
Units dense	[50, 300]	125
Activation dense	[relu, elu, selu]	selu
Dropout	[0.1, 0.6]	0.3
Optimizer	Adam, RMSprop, SGD	Adam
Learning rate	[0.01, 0.0001]	0.00072

3.2 Transformers

Viene descritto ora un paper trovato in letteratura [14] che tratta la TCC e utilizza un trasformer BERT. In questo si analizza come un livello di *subjectivity* possa essere aggiunto al modello per aumentarne le performance. In particolare, gli *identity terms* come "Muslim", "black", "women" e "democrat" vengono spesso ignorati, poiché portano a un alta presenza di falsi positivi (commenti non tossici con *identity terms*). Utilizzando un modello chiamato Subdentity-Sensitive BERT (SS-BERT), gli autori sono stati in grado di fare uso della struttura di BERT per attivare le caratteristiche di soggettività. Nonostante questo vada ben oltre l'obiettivo iniziale, la pubblicazione contiene molte informazioni riguardo all'utilizzo di BERT per la TCC.

Altre informazioni sono state prese da [15], che è tra i primi a fare un *fine-tune* con un modello BERT. La soluzione presentata qui non è però completamente adattabile al nostro problema, in quanto si poneva l'obiettivo solo di classificare tra *toxic* e *non-toxic*, e non una classificazione *multi-etichetta*. Un'informazione importante che viene però specificata è che il modello *"non si affida alla data augmentation"* [15]. Da questo paper vengono inoltre pre-

se alcune configurazioni utili nella fase di train, come il *batch size* a 32 e l'utilizzo di un *optimizer* Adam con *learning rate* a 2e-5.

BERT (Bidirectional Encoder Representations from Transformers), presentato nel 2019 in [16], è stato addestrato con un vocabolario di 30000 parole su Wikipedia inglese e sul Book Corpus, che contengono rispettivamente 2500 e 800 milioni di token.

È stato preaddestrato con due obiettivi [17]:

- *Masked language modeling* (MLM): prendendo una frase, il modello maschera casualmente il 15% delle parole in ingresso, poi fa passare la frase attraverso il modello che deve predire le parole mascherate.
- *Next sentence prediction* (NSP): il modello concatena due frasi mascherate come input durante il pretraining. Dovrà predire se le due frasi si susseguono o meno.

Come mostrato nella figura 3, BERT contiene 12 strati (blocchi di transformers), 12 *self-attention heads* e 110 milioni di parametri. Nell'equazione 1 viene mostrato il comportamento interno del blocco di *self-attention* che si basa sulle matrici delle query Q, delle chiavi K e dei valori V, scalando sulle dimensioni di queste.

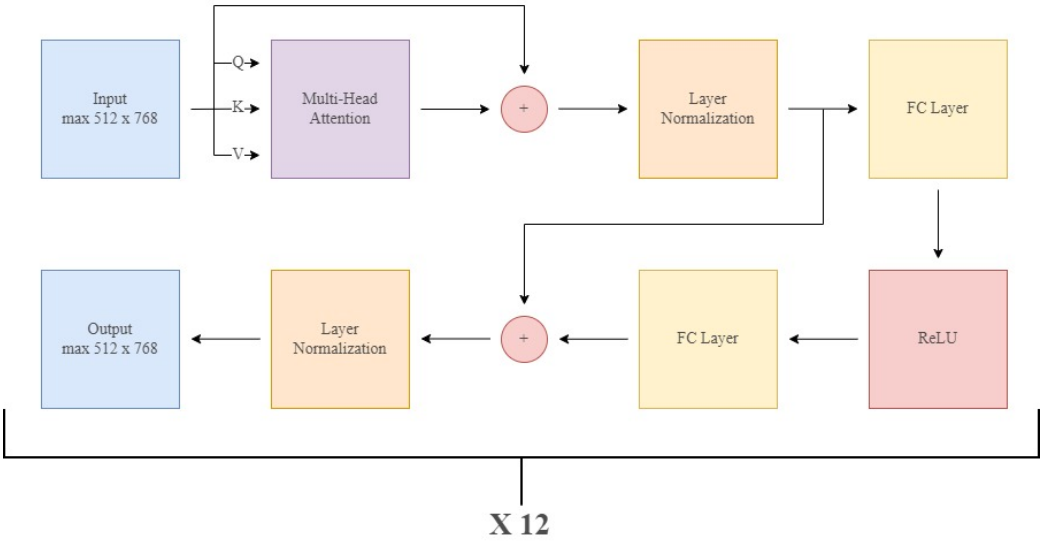

Figura 3: Schema di BERT

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

BERT ha due versioni: *uncased* e *cased*. La versione *uncased* ha solo lettere minuscole, ed è stata scelta per aumentare la semplicità del modello e rispecchiare le operazioni effettuate sui dati. Una sequenza di *token*, ovvero una frase pre-elaborata, di lunghezza massima 520 viene data in input al modello BERT. La lunghezza massima in token delle frasi è uno dei possibili parametri da tunare. A ciascuna sequenza vengono poi aggiunti due segmenti [CLS] e [SEP] dal *tokenizer* BERT. L'*embedding* [CLS], che è il primo *token* della sequenza in ingresso, viene utilizzato come *token* di classificazione. Nel nostro caso è un vettore di punteggi di lunghezza 6. Il *token* [SEP] è un artefatto della *task* di NSP e serve a separare le frasi. Come output BERT produce un vettore di dimensione 768 per rappresentare ogni sequenza di input. Poiché nel nostro compito abbiamo a che fare con contenuti testuali e il modello BERT è preaddestrato su testi generici, è fondamentale analizzare le informazioni estratte dagli strati del BERT preaddestrati. Secondo [16], gli strati inferiori del modello BERT possono contenere informazioni più generali, mentre gli strati superiori contengono informazioni specifiche per il compito (come per le CNN). Per questo motivo abbiamo bisogno di un *fine-tune* sul nostro compito di rilevamento di tossicità con set di dati annotati [18]. Il codice utilizzato è stato in parte preso da [19]. Qui veniva fatto un bilanciamento equo dei dati tra commenti *toxic* e *non-toxic*, mantenendo circa il 20% dei dati totali. Questo bilanciamento è in netta contrapposizione con la distribuzione del dataset, e i pro e contro sono stati studiati. Con un *undersampling* bilanciato il modello è in grado di analizzare molti commenti tossici, in particolare delle etichette con minore frequenza, che risultano le più difficili da predirre. Questo permette di discriminare ognuna delle sei classi, e infatti in Tabella 3 Rank 2 si ha un buon f1-score macro poiché in ogni etichetta vengano predetti degli 1. Questo non è banale, trattandosi infatti di un problema molto sbilanciato, etichette poco frequenti come *severe_toxic*, *threat* e *identity_hate* verranno predette sempre con 0 da modelli non ottimali. Problematica che emerge con un undersampling bilanciato, visibile in Tabella 3 Rank 3. Aumentando il numero di dati in input il problema viene risolto, e la soluzione proposta è quella mostrata in Tabella 3 Rank 1. Qui la lunghezza del *tokenizer* è stata abbassata a 120 e il vocabolario a 4000 *words* per le limitate risorse computazionali. Nonostante un'analisi precedente dimostrava come 200 fosse una buona dimensione delle sequenze,

l'innalzamento della lunghezza da 120 a 150 non ha apportato a un miglioramento delle performance. Un grande problema, visibile in Tabella 4, risulta essere l'elevato tempo di addestramento, trattandosi di un modello pesante. Un'alternativa sperimentata è stata la sperimentazione di distilBERT, modello simile ma molto più leggero. Il modello non è stato descritto in dettaglio poiché ha ottenuto scarse performance, non essendo in grado di classificare *threat* e *identity_hate* nemmeno avendo in input tutto il set di addestramento. Le performance vengono comunque riportate in Tabella 3 Rank 4. Inoltre il tempo di computazione risparmiato è trascurabile, passando da $8080s$ per il modello BERT a $7700s$ per il distilBERT.

4 Results and Evaluation

Tabella 2: Performance della RNN nelle differenti prove

Rank	Descrizione	Metrica	precision	recall	f1-score
1	**Dataset intero + tuning**	**weighted**	0.58	0.77	**0.66**
		macro	**0.54**	0.63	**0.57**
2	Glove.840B.300d + tuning	weighted	0.53	**0.83**	0.64
		macro	0.47	**0.72**	0.56
3	Dataset intero	weighted	**0.63**	0.66	0.62
		macro	0.41	0.38	0.37
4	Undersampling	weighted	0.45	0.8	0.55
		macro	0.4	0.58	0.44

Tabella 3: Performance del transformer BERT nelle differenti prove

Rank	Descrizione	Metrica	precision	recall	f1-score
1	**Dataset intero**	**weighted**	**0.56**	**0.83**	**0.67**
		macro	0.52	**0.75**	**0.61**
2	Undersampling	weighted	0.53	0.81	0.61
		macro	**0.57**	0.60	0.55
3	Undersampling sbilanciato	weighted	0.55	0.75	0.63
		macro	0.37	0.46	0.41
4	DistilBERT	weighted	0.56	0.70	0.61
		macro	0.38	0.41	0.39

Tabella 4: Confronto tra i due modelli nelle rispettive prove migliori

Modello	Train	Peso	Metrica	precision	recall	f1-score
Transformer	132min	420MB	weighted	0.56	**0.83**	**0.67**
			macro	0.52	**0.75**	**0.61**
RNN	6 min	**20.55MB**	weighted	**0.58**	0.77	0.66
			macro	**0.54**	0.63	0.57

5 Discussion

Per ambedue le architetture, la discriminante che ha portato a scartare o selezionare un modello riguarda la capacità di predizione nelle etichette minoritarie. Per raggiungere questo obiettivo è stato necessario focalizzarsi sui valori di *f1-score*, mettendo in secondo piano l'*accuracy*. Questa, a causa del forte sbilanciamento dei dati, risultava molto elevata anche in modelli che predicevano ogni etichetta a 0. Evitare tali soluzioni e muoversi verso un modello in grado di etichettare correttamente anche etichette poco frequenti è stata una delle principali sfide. Mentre i modelli ottenuti possiedono performance *weighted* simili sulle metriche presentate, il transformer riesce a raggiungere valori più alti nelle performance *macro*, ovvero effettua predizioni più accurate nelle etichette con meno esempi; la RNN tuttavia, come preannunciato, detiene un enorme vantaggio riguardo le tempistiche di addestramento, trattandosi di un modello molto più leggero.

Entrambi i modelli risultano *recall oriented*, ovvero in grado maggiormente di individuare gli insulti, caratteristica apprezzata pensando a in un'ipotetica applicazione nella realtà: di maggior importanza è l'individuazione di quanto più commenti tossici possibili, accettando il compromesso di segnalare dei *falsi positivi*.

Per quanto concerne le RNN, possibili miglioramenti futuri possono interessare la modifica di alcuni strati specifici all'architettura. Un esempio di modifica tentata riguarda la concatenazione del *GlobalMaxPooling1D* con il *GlobalAveragePooling1D*; queste modifiche sono state abbandonate fin da subito, in quanto non producevano miglioramenti e introducevano solo complessità nella comprensione. Una modifica più importante al modello riguarderebbe l'incrocio della RNN sviluppata con una *Convolution Neural Networks*, servendosi degli strati convoluzionali, tipici di queste reti, per estrarre le *feature* dal testo in ingresso. Un diverso approccio possibile consta nell'uso di due classificatori differenti su due problemi differenti: il primo per

distinguere solo commenti *toxic* e *non-toxic*, riducendo il problema a un'unica etichetta; per ogni commento classificato come *toxic*, si utilizzerebbe un secondo modello, simile a quello sviluppato, che distingua le diverse tipologie di tossicità.

Riguardo il dataset, lo sbilanciamento tra commenti *toxic* e commenti *non-toxic* è indubbiamente incisivo. Sarebbe utile considerare e avere la possibilità di integrare con dati esterni, sempre provenienti da problemi di TCC, per aumentare ancora il livello di generalizzazione dei modelli. C'è anche la possibilità di ricercare diversi modelli per il transformer, magari più leggeri e rapidi, non legati a BERT. L'approccio utilizzato che prevede lo sviluppo di modelli indipendenti riducendo la comunicazione necessaria è risultata efficace per lo sviluppo concorrente nei tempi prefissati; convergere in una fase finale integrando i due modelli sviluppati in un modello *ensemble* potrebbe essere interessante per cercare di raggiungere performance ancora più alte. Sono state prese direzioni di ricerca differenti, anche nella fase di pre-processing dei dati, dove è stata mantenuta la versione migliore.

6 Conclusions

I due modelli finali selezionati, un *Transformer* e una *RNN*, presentano performance simili nelle metriche *weighted* presentate; il Transformer eccelle nelle performance *macro*, mostrando maggiore precisione nelle etichette minoritarie. La RNN, sebbene più veloce nell'addestramento, è superata in questo aspetto. Entrambi i modelli sono *recall oriented*.

Per le RNN, possibili miglioramenti futuri possono coinvolgere la modifica dell'architettura, come l'integrazione con *Convolutional Neural Networks* per estrarre *feature* dal testo. Riguardo al dataset sbilanciato, si suggerisce l'integrazione con dati esterni per migliorare la generalizzazione. Si propone anche la ricerca di modelli più leggeri di Transformer, al di fuori del contesto di BERT. Infine, l'approccio di sviluppare modelli indipendenti e integrarli successivamente in un modello *ensemble* potrebbe realmente portare a un incremento delle performance.

Riferimenti bibliografici

[1] T. A., D. R., L. S D, N. D, R. R, R. M, R. L, and K. K, "Semantic-based classification of toxic comments using ensemble learning," *E3S Web of Conferences*, vol. 399, 07 2023.

[2] J. E. L. D. M. M. n. W. C. cjadams, Jeffrey Sorensen, "Toxic comment classification challenge," 2017. [Online]. Available: https://kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge

[3] M. Ibrahim, M. Torki, and N. El-Makky, "Imbalanced toxic comments classification using data augmentation and deep learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 875–878.

[4] "nlpaug 0.0.5," 2019. [Online]. Available: https://pypi.org/project/nlpaug/0.0.5/

[5] J. Webster and C. Kit, "Tokenization as the initial phase in nlp," 01 1992, pp. 1106–1110.

[6] "Nlp — text encoding: A beginner's guide," 2020. [Online]. Available: https://medium.com/analytics-vidhya/nlp-text-encoding-a-beginners-guide-fa332d715854

[7] "Go even deeper with char-gram+cnn," 2018. [Online]. Available: https://www.kaggle.com/code/sbongo/for-beginners-go-even-deeper-with-char-gram-cnn

[8] "Bi-gru-cnn-poolings," 2018. [Online]. Available: https://www.kaggle.com/code/konohayui/bi-gru-cnn-poolings

[9] "Tackling toxic using keras," 2018. [Online]. Available: https://www.kaggle.com/code/sbongo/for-beginners-tackling-toxic-using-keras

[10] "Improved lstm baseline: Glove + dropout," 2018. [Online]. Available: https://www.kaggle.com/code/jhoward/improved-lstm-baseline-glove-dropout

[11] "Capsule net with gru," 2018. [Online]. Available: https://www.kaggle.com/code/chongjiujjin/capsule-net-with-gru

[12] "Do pretrained embeddings give you the extra edge?" 2018. [Online]. Available: https://www.kaggle.com/code/sbongo/do-pretrained-embeddings-give-you-the-extra-edge/notebook

[13] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Kerastuner," https://github.com/keras-team/keras-tuner, 2019.

[14] Z. Zhao, Z. Zhang, and F. Hopfgartner, "Utilizing subjectivity level to mitigate identity term bias in toxic comments classification," *Online Social Networks and Media*, vol. 29, p. 100205, 05 2022.

[15] B. Kennedy, X. Jin, A. M. Davani, M. Dehghani, and X. Ren, "Contextualizing hate speech classifiers with post-hoc explanation," 2020.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[18] M. Mozafari, R. Farahbakhsh, and N. Crespi, "Hate speech detection and racial bias mitigation in social media based on bert model," *PLOS ONE*, vol. 15, no. 8, pp. 1–26, 08 2020. [Online]. Available: https://doi.org/10.1371/journal.pone.0237861

[19] "Toxic comment classification using bert," 2023. [Online]. Available: https://www.geeksforgeeks.org/toxic-comment-classification-using-bert/