Università di Pisa

Seminario di cultura digitale

A.A. 2022/2023

Irene Mondella 584285
Elena Scaglione 645638

Indice

1 Introduzione

L'*hate speech*, o discorso di odio, è stato definito da Nockleby (2000) come:

> Any communication that disparages a person or a group on the basis of some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic.

Discorsi di questo tipo sono alimentati da una mentalità fondata sull'odio verso il diverso, e a loro volta contribuiscono ad alimentare questa mentalità, che spesso sfocia anche in azioni concrete: aggressioni, micro-aggressioni, discriminazioni sul lavoro, assenza di diritti civili, oltre a danneggiare la salute mentale e il benessere delle persone che subiscono l'*hate speech*. Una recente indagine ISTAT-UNAR[1] sulle discriminazioni lavorative verso persone LGBT+ in unione civile riporta che, all'interno del campione considerato, circa sei persone su dieci hanno sperimentato almeno una forma di micro-aggressione sul lavoro, mentre il 34,5% dei dipendenti dichiara di aver subito almeno un evento di discriminazione in ambito lavorativo. Inoltre, Il 38,2% delle persone ha subito discriminazione in altri contesti di vita: nel proprio quartiere, negli uffici pubblici, nei mezzi di trasporto, nei negozi, nell'ambito ospedaliero.

Per questo motivo, l'*hate speech* non è un fenomeno da sminuire e tralasciare, ma da combattere. Ovviamente, il modo principale per farlo dovrebbe essere l'educazione della popolazione, ma sono necessarie anche strategie più immediate. I social network, ad esempio, sono uno degli strumenti usati per diffondere odio e violenza verso le minoranze. Tuttavia, tali piattaforme possono adottare politiche rigorose per contrastare i discorsi di odio: permettere agli utenti di segnalare i contenuti offensivi, rimuovere tali contenuti e prendere provvedimenti contro i loro autori. In questo processo, l'intelligenza artificiale può essere un utile alleato. Una moderazione svolta completamente da esseri umani, infatti, presenta delle limitazioni in termini di tempo, costi e benessere psichico dei moderatori, in quanto la prolungata e continua esposizione a contenuti inopportuni può avere delle conseguenze dannose su chi ne è soggetto (nel 2020, ad esempio, dopo una *class action* intentata dai moderatori, Facebook ha accettato di risarcirli per problemi di salute mentale sviluppati sul lavoro, come si legge nell'articolo pubblicato su "The Verge" da Newton, 2020).

È molto utile e vantaggioso, dunque, implementare modelli di *machine learning* in grado di identificare automaticamente post e messaggi contenenti discorsi d'odio. Questo è lo scopo di un campo di ricerca molto attivo nella comunità NLP, ovvero l'*hate speech detection*. Tuttavia, come notato da Nozza (2021), questo task presenta due importanti limitazioni. Innanzitutto, l'*hate speech* può essere rivolto a gruppi diversi, assumendo caratteristiche differenti in base alla minoranza presa di mira (declinandosi quindi in misoginia, razzismo, omotransfobia...). Inoltre, la maggior parte dei lavori in questo campo considera solo la lingua inglese, mentre per le altre lingue non ci sono abbastanza corpora etichettati e modelli addestrati su di essi. L'approccio più immediato, in questi casi, è l'apprendimento *zero-shot*: un modello di *hate speech detection* sviluppato per l'inglese e per un tipo specifico di discorso d'odio (es. il razzismo) viene utilizzato anche in contesti diversi (ad esempio, identificare l'omofobia in testi italiani). Questa metodologia, pur risultando soddisfacente e vantaggiosa in task di altro tipo, presenta significative limitazioni nell'ambito dell'*hate speech*. Uno dei motivi consiste nell'utilizzo, in alcune lingue, di parole offensive in contesti che non sono di odio. Ad esempio, in italiano e in spagnolo termini misogini come "put**na" o "tr**a" vengono comunemente utilizzati in espressioni non di odio (ad esempio: "porca put**na"): un modello addestrato sull'inglese, dunque, identificherà come *hateful* messaggi contenenti queste espressioni anche se non lo sono.

Per questo motivo, dati i limiti dell'apprendimento *zero-shot* per l'*hate speech detection*, è necessario costruire dataset specifici per le singole lingue e i singoli contesti di odio, su cui effettuare il *fine-tuning* dei modelli. Un dataset di questo tipo è quello presentato da Nozza, Cignarella et al. (2023), che contiene esclusivamente post pubblicati su Twitter in lingua italiana, ed etichettati in base alla presenza o meno di omotransfobia al loro interno. Utilizzando questo dataset, cercheremo di addestrare un Language Model affinché apprenda a riconoscere correttamente quando un testo veicola o meno messaggi di omotransfobia. In seguito, testeremo il modello su dati nuovi, per misurarne l'efficacia. Infine, adotteremo tecniche di *explainable AI* per comprendere le motivazioni dietro alle predizioni fatte dal modello.

2 Stato dell'arte

2.1 Gli approcci tradizionali alla classificazione automatica di dati testuali

Fino a una decina di anni fa, classificare un testo in maniera automatica richiedeva l'estrazione manuale di *features* da dare in input a un modello di *machine learning* (Li et al., 2021). È infatti necessario

[1] `https://www.istat.it/it/files//2022/03/REPORTDISCRIMINAZIONILGBT_2022_rev.pdf`

passare da una rappresentazione simbolica, come è il linguaggio, a una numerica, comprensibile a una macchina. Per estrarre tali caratteristiche statistiche da un testo sono stati sviluppati diversi metodi:

- Bag of words (BOW; Zhang et al., 2010): trasforma il testo in un vettore della lunghezza del vocabolario i cui valori corrispondono alla frequenza di ciascun termine.
- N-grammi (Cavnar, Trenkle et al., 1994): calcola la frequenza di sequenze di n parole adiacenti.
- Term frequency-inverse document frequency (TF-IDF; Rajaraman e Ullman, 2011):pesa la frequenza di ciascun termine in base all'importanza che ha il termine in una collezione di documenti.

Le *features* possono essere anche caratteristiche linguistiche, come parti del discorso, alberi sintattici o ruoli semantici (Straka e Straková, 2017).

Una volta trovate le *features* più rilevanti per il task in questione, il dataset viene fornito al modello. Con questo tipo di approcci, gli algoritmi più utilizzati sono Naïve Bayes (NB; Maron, 1961), K-Nearest Neighbors (KNN; Cover e Hart, 1967) e Support Vector Machine (SVM; Joachims, 1998).

2.2 I modelli neurali

Grazie alla crescente disponibilità di dati su Internet e allo sviluppo di hardware sempre più potenti, a partire dal 2010 l'utilizzo delle reti neurali per processare sequenze testuali ha visto un aumento costante.

2.2.1 Il Neural Probabilistic Language Model

Già nel 2003 è stato teorizzato il Neural Probabilistic Language Model (NPLM) da Bengio et al., che consiste nell'addestrare una rete neurale al compito del *language modeling*, cioè predire una parola, data una sequenza di parole che la precede. Di fatto consiste nel trovare la parola con la massima probabilità date le parole precedenti. Possiamo definire la probabilità di una frase $f = [w_1, w_2, \ldots, w_N]$ come:

$$P(f) = \prod_{i=1}^{N} P(w_i | w_1, w_2, \ldots w_{i-1}) \tag{1}$$

Un modello statistico del linguaggio è rappresentato, quindi, dal prodotto delle probabilità condizionate di ogni parola data la sequenza di parole precedenti.

Bengio introduce il NPLM per risolvere il problema della *curse of dimensionality* negli approcci tradizionali (che utilizzano vettori sparsi, di grandi dimensioni e con molti zeri). Utilizza invece come *features* una rappresentazione distribuita dell'input costituita da vettori di numeri reali e densi. Infatti, il problema dei modelli a n-grammi è la sparsità dei dati linguistici: tranne un piccolo gruppo di parole funzionali, la maggior parte delle parole hanno frequenze molto molto basse. Stimare la probabilità di una frase diventa estremamente difficile perché bisogna aver già incontrato nel corpus di addestramento tale sequenza e, data la produttività del linguaggio, esisteranno sempre degli eventi linguistici non osservati. Bengio sviluppa una rete neurale che predice la parola successiva, dando in output un vettore della lunghezza del vocabolario dove l'i-esimo valore è la probabilità della parola corrispondente nel vocabolario. Il prodotto della rete sono anche gli *embeddings*, cioè dei vettori densi che costituiscono i parametri che la rete impara durante l'addestramento.

2.2.2 Word2vec

La più famosa famiglia di modelli per creare *word embeddings* è Word2vec (Mikolov, Sutskever et al., 2013; Mikolov, Chen et al., 2013). A tal fine, vengono implementate due diverse architetture: Continuous Bag of Words (CBOW) e Skipgram. CBOW predice la parola centrale, date le parole del contesto, mentre Skipgram predice il contesto data la parola al centro.

Per rendere il processo più computazionalmente efficiente, il task di predizione viene trasformato in un task di classificazione, in cui il modello deve predire se una parola si trova nel vicinato di un'altra parola. Le parole vengono selezionate con il processo di *negative sampling*, vogliamo cioè trovare un campione di parole che non fanno parte del vicinato della parola di cui vogliamo generare l'*embedding*, in modo che il modello sia in grado di discriminare tra le parole del contesto e parole casuali.

Alla fine dell'addestramento, la matrice dei pesi sarà la rappresentazione distribuita di ciascun termine.

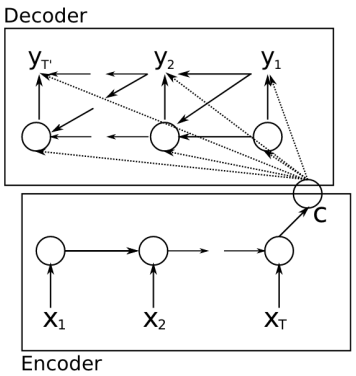


Figura 1: RNN Encoder Decoder. Fonte: Cho, van Merriënboer, Gulcehre et al., 2014

2.2.3 Gli embedding contestuali

Un problema di word2vec e simili è che producono un'unica rappresentazione, mentre una parola viene usata in diversi contesti, acquisendo perciò significati molteplici in base al contesto. Peters et al. (2018) propongono ELMo, un modello che fornisce una rappresentazione per ogni parola token (e non per ogni parola tipo) tramite una rete con architettura Long-Short Term Memory (LSTM; Hochreiter e Schmidhuber, 1997) bidirezionale. Questo vuol dire che predice non solo la probabilità della parola successiva dato il contesto sinistro, come in Equazione 1, ma anche al contrario, quindi dato il contesto destro:

$$P(f) = \prod_{i=1}^{N} P(w_i | w_{i+1}, w_{i+2}, \dots w_N) \tag{2}$$

Gli esperimenti hanno dimostrato grandi miglioramenti applicando ELMo a diversi task di NLP.

2.2.4 RNN Encoder–Decoder e il meccanismo dell'Attention

Un'ulteriore svolta nell'elaborazione di sequenze di testo avviene nel 2014, con l'introduzione dell'architettura Encoder-Decoder per migliorare il task della traduzione automatica (Cho, van Merriënboer, Gulcehre et al., 2014).

L'architettura Encoder-Decoder consiste in due Recurrent Neural Networks (RNN; Elman, 1990) capaci, congiuntamente, di creare una rappresentazione interna di una frase in input e generare una frase in output.

Recurrent Neural Networks Una RNN è una rete neurale che riceve in input due valori: una sequenza $\mathbf{x} = x_1, \dots, x_T$ e un *hidden state* $\mathbf{h}$, calcolato come:

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, x_t\right) \tag{3}$$

dove f è una funzione di attivazione non lineare (ad esempio una semplice funzione logistica, ma anche una più complessa unità LSTM, come proposto da Sutskever et al. (2014)). A differenza di una rete neurale *feed-forward fully connected*, le reti con connessioni ricorrenti possono gestire sequenze di dati di lunghezza variabile, conservando in memoria gli elementi della sequenza già elaborati.

RNN Encoder-Decoder Nel modello Encoder-Decoder l'*encoder* trasforma una sequenza di lunghezza variabile in un vettore di lunghezza fissa e il *decoder* ritrasforma il vettore ricevuto dall'*encoder* in una sequenza di lunghezza variabile. Le due reti vengono addestrate insieme per massimizzare la probabilità condizionata della sequenza target data una sequenza in input, cioè $p(y_1, \dots, y_{T'} | x_1, \dots, x_T)$, dove le lunghezze delle sequenze T' e T possono essere diverse.

L'*encoder* legge in sequenza ogni elemento di $\mathbf{x}$ e aggiorna l'*hidden state* secondo l'Equazione 3. Dopo aver letto il simbolo che segna la fine della frase, viene passato al *decoder* il *context vector* ($\mathbf{c}$ in Figura 1), cioè l'ultimo *hidden state*, che contiene una rappresentazione dell'input. Il *decoder* usa il *context vector* per inizializzare il proprio *hidden state*.

Il *decoder* è una rete ricorrente addestrata a predire il successivo elemento y_t della sequenza di output dato il proprio *hidden state* $\mathbf{h}_t$. Ma, a differenza di una RNN classica, y_t e $\mathbf{h}_t$ dipendono anche da y_{t-1} e il *context vector* $\mathbf{c}$, perciò $\mathbf{h}_t$ è calcolato come:

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{c}\right) \tag{4}$$

Tuttavia, tramite esperimenti empirici, si dimostra che la performance va diminuendo man mano che la frase da tradurre diventa più lunga, questo perché diventa sempre più difficile rappresentarla in maniera ottimale in un vettore di lunghezza fissa (Cho, van Merriënboer, Bahdanau et al., 2014).

Attention Viene quindi introdotta da Bahdanau et al. (2014) una nuova tecnica che migliora la qualità delle traduzioni, in quanto permette al modello di concentrarsi solo su alcune parti dell'input, le più rilevanti per la parola da generare in quel momento.

A differenza del semplice modello Encoder-Decoder, in questo caso l'*encoder* non passa al *decoder* solamente l'ultimo *hidden state*. Il *context vector* è invece una somma pesata di tutti gli *hidden states* di ogni elemento della sequenza di input e viene creato dinamicamente ogni volta che il *decoder* genera un nuovo elemento della sequenza di output. Ogni *hidden state* $\mathbf{h}_i$ è associato maggiormente con l'i-esimo elemento dell'input. In questo modo pesiamo le parole dell'input che più o meno influenzano l'output corrente del *decoder*.

2.2.5 Transformers

Il problema delle reti neurali ricorrenti è che richiedono calcoli sequenziali. I Transformers (Vaswani et al., 2017) invece, si basano sempre su un modello *encoder-decoder*, ma non possiedono connessioni ricorrenti, possono quindi effettuare i calcoli in parallelo, rendendo l'addestramento su grandi quantità di testo molto più scalabile.

La struttura di base di un Transformer è costituita da una serie di *encoder* e una serie di *decoder*. In Vaswani et al. (2017) sono 6 e 6. L'input dell'*encoder* passa da uno strato di *self-attention* e in seguito da una rete *feed-forward*. Il *decoder* invece ha un ulteriore strato di attention tra i due, detta *encoder-decoder attention*. L'architettura è visibile in Figura 2.

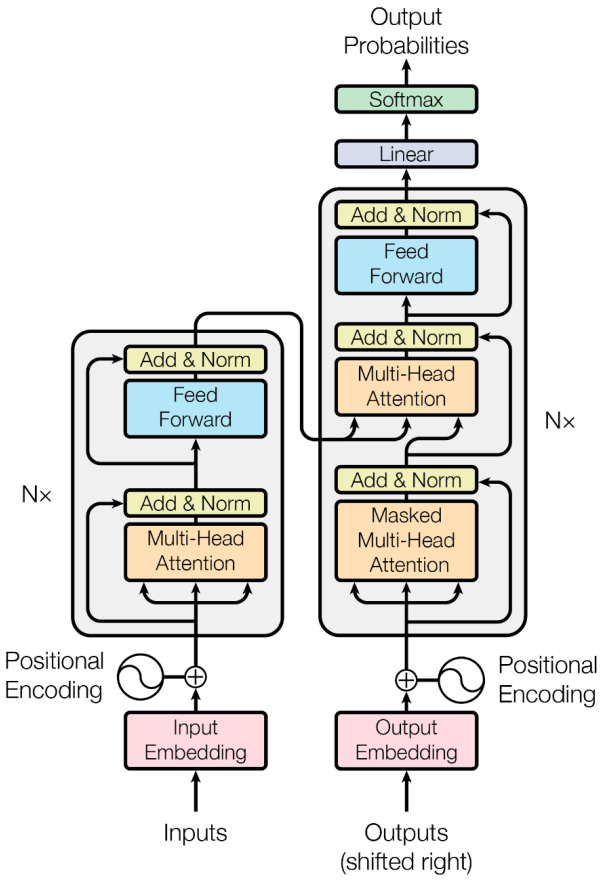


Figura 2: Architettura del Transformer. Fonte: Vaswani et al., 2017

Self-Attention Uno strato di *self-attention* associa una sequenza $\mathbf{x} = x_1, \ldots, x_n$ a una sequenza $\mathbf{y} = y_1, \ldots, y_n$ della stessa lunghezza. Nello specifico, tramite la *self-attention* paragoniamo ogni elemento

della sequenza di input con tutti gli altri elementi nella stessa sequenza (compreso ciascuno con sé stesso). Nei Transformers vengono introdotti tre set di pesi, che corrispondono a tre ruoli:

- *query*: l'elemento a cui stiamo paragonando tutti gli altri
- *key*: il secondo termine di paragone
- *value*: un vettore di valori che usiamo per ottenere l'output

Avremo quindi:

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i; \tag{5}$$

Dal momento però che ogni output può essere calcolato in maniera indipendente, possiamo avvalerci del calcolo tramite matrici:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q; \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K; \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V; \tag{6}$$

Dove $\mathbf{X} \in \mathbb{R}^{N \times d}$ è una matrice le cui righe corrispondono agli *embedding* di input. La Self-Attention è quindi calcolata come:

$$SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{7}$$

Nel *decoder*, per evitare che in un task di *language modeling* la parola successiva sia visibile al modello, il triangolo superiore della matrice quadrata $\mathbf{Q}\mathbf{K}^T$ viene posto a $-\infty$ (*masked self-attention*), come si può vedere in Figura 3.

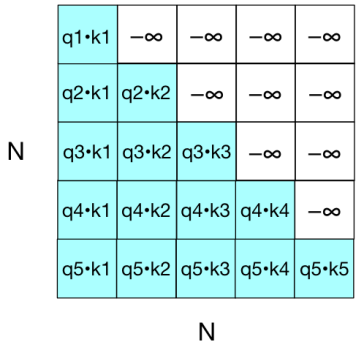


Figura 3: Matrice $\mathbf{Q}\mathbf{K}^T$ in cui i valori delle parole successive sono nascosti. Fonte: Jurafsky e Martin, 2023

Encoder-Decoder Attention Questo strato di *attention* permette al *decoder* di concentrarsi su diverse parti dell'input, rappresentate dall'*encoder*. Le *query* vengono dallo strato precedente del *decoder*, mentre le *key* e i *value* dall'output dell'*encoder*.

Multi-head Attention Dal momento che in una frase possono intercorrere relazioni tra le parole su vari livelli, si è scelto di usare degli strati paralleli di *self-attention*, creando una *multi-head attention*. Ogni "testa" ha le proprie matrici di *key*, *query* e *value*: $\mathbf{W}_i^K$, $\mathbf{W}_i^Q$ e $\mathbf{W}_i^V$.

$$Multihead(H) = [head_1, head_2, \ldots, head_h]W^O \tag{8}$$

W^O è un'altra proiezione lineare necessaria per riportare l'output da d_v a d, la dimensione dell'input.

Normalizzazione Tra uno strato e l'altro viene anche effettuata un'operazione di normalizzazione, per mantenere i valori entro un certo intervallo. Perciò per ogni vettore viene calcolata la media di tutti i valori e la deviazione standard. Poi a tutti i valori viene sottratta la media e vengono divisi per la deviazione standard. In questo modo la media dell'intervallo entro cui variano i valori di ciascun vettore sarà zero e la deviazione standard uguale a 1. Ciò assicura un gradiente più stabile attraverso i numerosi strati.

Positional Encoding Dal momento che la struttura della rete non dà informazioni sulla posizione di ciascun token nella sequenza (come nelle RNN), è necessario aggiungerle agli *embeddings* di input. Si è scelto dunque di aggiungere dei vettori prestabiliti all'input, in modo da caratterizzare ogni posizione con valori diversi.

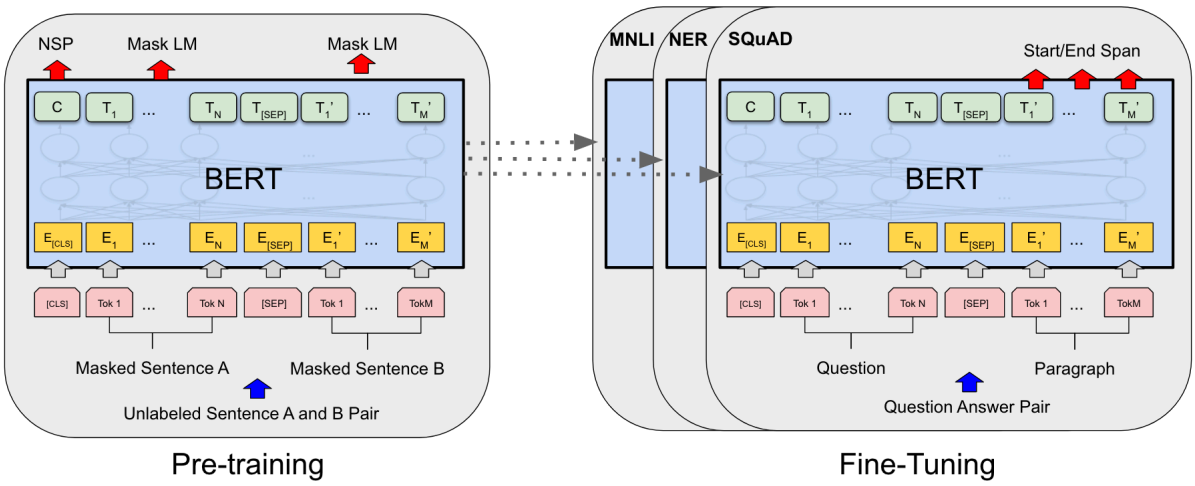


Figura 4: Pre-training e fine-tuning di BERT. Fonte: Devlin et al., 2019

Connessioni residue Oltre alla normalizzazione per ogni strato, vengono inserite anche delle connessioni residue, cioè viene aggiunto l'input di ogni strato al suo output (quindi è come se l'informazione fluisse saltando uno strato).

Pre-training e fine-tuning Una caratteristica fondamentale che ha contribuito al successo dei Transformers è il fatto che, dopo l'addestramento iniziale (*pre-training*), solo una piccola parte di parametri devono essere regolati per adattare il modello a molteplici applicazioni, con un dispendio computazionale relativamente modesto, se paragonato all'addestramento iniziale. Questo addestramento successivo è detto *fine-tuning* e viene fatto mantenendo la stessa architettura del preaddestramento. In Figura 4 *pre-training* e *fine-tuning* di BERT. Il vantaggio è anche relativo alla disponibilità di dati: per il *fine-tuning* sono necessari dati specifici, ma in quantità sicuramente minori rispetto ai dati necessari per addestrare un modello da zero. Questo nuovo paradigma, derivato dal concetto di *transfer learning*, rende questi modelli del linguaggio estremamente versatili.

2.2.6 BERT

BERT o Bidirectional Encoder Representations from Transformers (Devlin et al., 2019) è un modello del linguaggio di tipo Transformer. A differenza del Transformer originale di Vaswani et al. (2017), che è addestrato in maniera unidirezionale da sinistra a destra, BERT possiede un *encoder* bidirezionale, che quindi "vede" a destra e a sinistra della parola che deve predire. BERT, essendo privo di *decoder*, è un Transformer adatto a task che non richiedono generazione di output, come i task di classificazione. È dunque necessario modificare l'obiettivo per il preaddestramento del modello.

Masked Language Modeling Il task principale di BERT viene definito di Masked Language Modeling (MLM) ed è un *cloze task*. Il *cloze task* consiste nel predire una parola mancante all'interno della frase. Durante l'addestramento quindi delle parole vengono nascoste e il compito del modello è predire queste parole. Nello specifico, in BERT il 15% dell'input tokenizzato viene selezionato. Di questo 15%, l'80% viene rimpiazzato con il token `[MASK]`, il 10% viene rimpiazzato da parole casuali e il 10% rimane invariato.

Next Sentence Prediction Il secondo compito di BERT è la Next Sentence Prediction (NSP). Questo task è necessario per alcune applicazioni in cui bisogna determinare la relazione tra coppie di frasi, come ad esempio il Natural Language Inference (NLI) o il Question Answering (QA). Il modello deve quindi stabilire se delle coppie di frasi sono adiacenti o meno.

Sentence Classification Per svolgere i due compiti, vengono anche introdotti due token ausiliari:

- `[CLS]` viene aggiunto all'inizio di ogni frase.
- `[SEP]` viene aggiunto tra la fine della prima frase e l'inizio della seconda.

Inoltre, il token `[CLS]`, alla fine dell'addestramento, contiene il *sentence embedding* (la rappresentazione dell'intera frase) e serve come input dello strato preposto al task di classificazione (*classifier head*) durante la fase di fine-tuning. Mentre i pesi del modello preaddestrato rimarranno congelati, i parametri di questo ultimo strato saranno regolati tramite apprendimento supervisionato.

Knowledge Distillation Uno dei limiti di BERT, come di altri modelli del linguaggio con centinaia di milioni di parametri, è la sua dimensione. Una delle tecniche utilizzate per ridurre la dimensione di questi modelli è la *knowledge distillation* (Hinton et al., 2015). La *knowledge distillation* consiste nell'addestrare un modello più piccolo, detto "studente", a riprodurre il comportamento di un modello più grande, detto "insegnante". Per esempio, a partire da BERT è stato sviluppato DistilBERT (Sanh et al., 2020), che con una dimensione minore del 40% raggiunge performance simili ed è più veloce del 60%.

2.3 L'hate detection

Per fornire un quadro completo sullo stato dell'arte nell'ambito dell'*hate detection*, il nostro principale punto di riferimento è un'analisi condotta da Jahan e Oussalah (2023), in quanto fornisce una panoramica esaustiva e aggiornata sull'argomento.

I primi articoli sull'*hate detection* vengono pubblicati a partire dal 2010 (escludendo un unico contributo del 2005): dal 2010 al 2016 tuttavia l'argomento non gode di particolare popolarità, in quanto troviamo solo 25 pubblicazioni al riguardo. Dal 2017 fino al marzo 2021 il numero aumenta rapidamente, fino a raggiungere 341. Di queste, 96 riguardano metodi di *deep learning*.

La maggior parte degli studi (50%) si concentra su dataset di lingua inglese, mentre le altre lingue più rappresentate sono arabo (13%), turco (6%) e greco (4%). I dataset in lingua italiana costituiscono solo una percentuale ridotta, circa il 2% del totale.

Il social principale da cui vengono raccolti i dati è Twitter (47%), seguito da Facebook (12%), YouTube (9%) e Wikipedia (5%).

Per quanto concerne i modelli di *machine learning*, i metodi supervisionati rappresentano l'approccio predominante. In particolare, Support Vector Machine (SVM) è l'algoritmo più frequentemente utilizzato (29%). Tuttavia, è degno di nota il crescente interesse per i metodi di deep learning, reti neurali a molti strati, che hanno guadagnato popolarità a partire dal 2017 nell'ambito del trattamento automatico del linguaggio e non solo.

Tra i metodi di *deep learning*, il modello più ampiamente adottato è BERT (38%), nonostante sia stato introdotto solo nel 2019. Nello specifico, i modelli di BERT con più parametri si rivelano migliori del modello base, e combinare diversi modelli basati su BERT dà migliori risultati rispetto a usare un singolo modello. Altri approcci di *deep learning* degni di nota includono le reti neurali con architettura LSTM (23%) e le reti neurali convoluzionali (CNN) (14%). Una piccola percentuale è da attribuire anche ad architetture complesse, formate da più modelli combinati come BERT + CNN (2%), LSTM + CNN(9%), LSTM + GRU(1%), BERT + LSTM(2%).

Il successo dei modelli basati sulle reti neurali, in particolare i Transformer come BERT, è spiegato anche dalla loro capacità di eseguire un addestramento generale (*pre-training*) su grandi corpora di testo e successivamente affinare il modello per compiti specifici attraverso un addestramento mirato (*fine-tuning*). Di conseguenza, modelli basati su BERT sono stati adattati con successo a diverse lingue e domini, dimostrando una versatilità che ha ampliato significativamente le possibilità di applicazione in diversi ambiti.

3 Task: Homotransphobia Detection in Italian (HODI)

3.1 Descrizione dei task

Il nostro obiettivo, come anticipato nell'Introduzione (sezione 1), è quello di ottenere un Language Model specializzato nel riconoscere messaggi di omotransfobia, prendendo ispirazione da HODI[2], uno shared task di *hate detection* presentato a EVALITA 2023[3]. Lo scopo del task è quello di identificare l'omotransfobia all'interno di post italiani su Twitter, e per far ciò è stato messo a disposizione un dataset etichettato. Nello specifico, HODI è organizzato in due sub-task:

1. *Homotransphobia detection*: riconoscere se un testo è omotransfobico o no;
2. *Explainability*: trovare le motivazioni delle classificazioni effettuate dal modello addestrato per il subtask A.

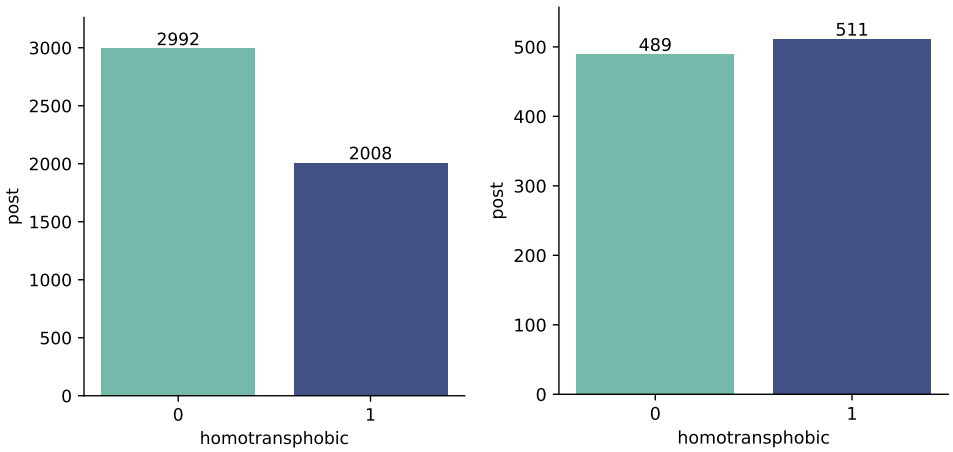
3.2 Descrizione del dataset

Sono stati messi a disposizione due dataset: uno per effettuare l'addestramento e uno per testare le performance del modello finale. Il dataset di *training* è composto da 5000 post in lingua italiana raccolti

[2] `https://hodi-evalita.github.io/#about`
[3] `https://www.evalita.it/`

da Twitter, di lunghezza compresa tra i 23 e i 368 caratteri (lunghezza media pari a 180 caratteri). Ognuno di questi post è accompagnato da un'etichetta binaria inserita manualmente dai creatori del dataset: '1' se il testo veicola messaggi di omotransfobia, '0' altrimenti. Il dataset di test, invece, è composto da 1000 tweet, anch'essi etichettati.

La distribuzione della variabile target varia leggermente tra i due dataset: nel primo, il 60% dei post è omotransfobico e il 40% non lo è (Figura 5a), mentre nel secondo il 49% dei tweet è omotransfobico e il 51% no (Figura 5b).



(a) Distribuzione nel dataset di training (b) Distribuzione nel dataset di test

Figura 5

Per entrambi i dataset, abbiamo effettuato un semplice *preprocessing*, allo scopo di eliminare dai post eventuali elementi di disturbo. In ogni tweet, dunque, abbiamo rimosso: le menzioni ad altri utenti (precedute dal carattere @), caratteri privi di significato come il ritorno a capo, il trattino basso ecc., le date, gli URL e tutte le emoticon. Infine, abbiamo trasformato tutti i post in minuscolo.

Un volta puliti i due dataset, abbiamo generato le *word cloud*: una *word cloud* è una visualizzazione delle parole di un documento, con la particolarità che le parole più importanti sono rappresentate con un font più grande. In questo caso, il criterio per definire l'importanza di una parola è la frequenza all'interno del dataset. Nello specifico, ci interessava vedere se, nel dataset di addestramento, i post etichettati come omotransfobici presentassero determinate parole di odio con una frequenza maggiore di quelli non omotransfobici. Il risultato è visibile in Figura 6, da cui però si nota che anche nei post etichettati con '0' sono presenti termini omofobi, volgari e offensivi. Questo dimostra come, in realtà, non è soltanto la presenza o meno di certe parole a veicolare messaggi d'odio: un testo può riportare parole offensive ma non essere *hate speech*, oppure esserlo pur non esplicitando un lessico di questo tipo. Ciò rende questo task fortemente dipendente dal contesto, e perciò più complesso.

(a) *Word cloud* dei post omotransfobici (b) *Word cloud* dei post non omotransfobici

Figura 6: *Word cloud* del dataset di training

4 Modelli ed esperimenti

Dall'avvento dei Transformers (di cui abbiamo parlato nel Paragrafo 2.2.5), l'utilizzo di tale architettura, e dei *Language Models* che si basano su di essa, è diventato predominante per la risoluzione di qualsiasi task linguistico, compresa l'*hate detection*. Anche noi, dunque, ci siamo rivolte a questo tipo di modelli.

Inizialmente era stata considerata l'idea di utilizzare architetture leggermente più semplici ma comunque recenti, quali le LSTM. In tal caso, avremmo addestrato tale rete da zero, ma utilizzando dei *word embeddings* (§2.2.2) preaddestrati. Questo approccio, però, permette di fornire al modello soltanto il significato delle parole, sotto forma di un vettore di numeri, ma non riesce a fornirgli anche l'informazione contestuale. Tuttavia, come abbiamo constato tramite le *word cloud*, il contesto è fondamentale per un task di questo tipo: perciò, l'addestramento di una semplice LSTM è stato scartato.

Per quanto riguarda i Transfomers, abbiamo utilizzato l'omonima libreria di Hugging Face[4], soffermandoci sui seguenti modelli: BERT finetunato sull'italiano[5], DistilBERT finetunato sull'italiano[6], BERT multilingue[7], DistilBERT multilingue[8], un BERT finetunato sull'*hate speech* per l'italiano[9] (Aluru et al., 2020) e un RoBERTa anch'esso specifico per *hate speech* italiano[10] (Nozza, Bianchi et al., 2022).

Abbiamo provato ben sei modelli con l'obiettivo di indagare quale fosse il migliore per il nostro task. Nello specifico, volevamo verificare se:

1. le versioni multilingue, conoscendo anche l'italiano, fossero competitive con i modelli appositi per l'italiano;
2. DistilBERT, pur essendo un modello più piccolo e con meno parametri, fosse in grado di risolvere questo task con un'accuratezza paragonabile a quella di un modello più grande come BERT;
3. modelli finetunati per l'*hate speech* permettessero effettivamente di ottenere performance migliori nel nostro task di *homotransphobia detection*.

Per rispondere a queste domande, abbiamo addestrato i sei modelli sopraelencati per 10 epoche sul 70% del dataset di *training* (quindi su 3500 post), mantenendo i dati restanti per fare *validation* (in questa fase, il test set non è stato utilizzato). Per giudicare le performance dei modelli, abbiamo analizzato gli F1-score ottenuti sul *validation set*, che sono riportati in Tabella 1, in cui viene anche indicata l'epoca in cui è stato ottenuto quel punteggio.

Nome	F1-score	Epoca
Italian BERT	**0.830**	6
Italian DistilBERT	0.800	9
Italian dehateBERT	0.791	8
BERT multilingual	0.772	6
DistilBERT multilingual	0.771	2
HATE-ITA Base	0.771	6

Tabella 1: *Language Models* provati e massimo F1-score raggiunto

Da questi esperimenti, dunque, emerge che il modello migliore risulta essere il BERT italiano, con un F1-score dello 0.830. In generale, si evince come, per nostro specifico task, i modelli finetunati sull'italiano sono migliori di quelli multilingue, che BERT raggiunge un'accuratezza migliore di DistilBERT e che i modelli specifici per l'*hate speech detection* in realtà non ci permettono di ottenere performance migliori di quelle ottenute da modelli più generici.

A questo punto, abbiamo effettuato una piccola *hyperparameter search*, per capire se, modificando gli iperparametri del modello, fosse possibile incrementarne ulteriormente l'accuratezza. Nello specifico, abbiamo voluto testare il modello con l'*optimizer* Adagrad, al posto del default Adam, e con altri valori di *learning rate*, uno leggermente più grande e uno leggermente più piccolo, per un totale di sei possibili configurazioni. Quindi, gli iperparametri considerati sono i seguenti (i valori di default sono evidenziati in grassetto):

- Optimizer: **Adam**, Adagrad;

[4] `https://huggingface.co/docs/transformers/index`
[5] `https://huggingface.co/dbmdz/bert-base-italian-cased`
[6] `https://huggingface.co/osiria/distilbert-base-italian-cased`
[7] `https://huggingface.co/bert-base-multilingual-cased`
[8] `https://huggingface.co/distilbert-base-multilingual-cased`
[9] `https://huggingface.co/Hate-speech-CNERG/dehatebert-mono-italian`
[10] `https://huggingface.co/MilaNLProc/hate-ita-xlm-r-base`

- Learning rate: 3e-5, **5e-5**, 7e-5.

Queste sei configurazioni di BERT-ita sono state addestrate per 10 epoche sul nostro *training set*. Ovviamente, non è stato necessario testare nuovamente la configurazione di default, poiché era stata già provata nella fase precedente. Anche in questo caso, il 30% del dataset di addestramento è stato lasciato per fare *validation*.

Le tre configurazioni migliori sono riportate in Tabella 2, in ordine di performance, da cui si nota come il modello migliore si conferma essere il BERT italiano con gli iper-parametri di default.

Optimizer	Learning rate	F1-score	Epoca
Adam	5e-5	**0.830**	6
Adam	3e-5	0.828	6
Adam	7e-5	0.817	6

Tabella 2: Configurazioni dei 3 modelli con F1 maggiore

Questo modello, dunque, è risultato essere il modello migliore, da valutare anche sul dataset di test. Tuttavia, anziché utilizzare il modello già addestrato sul 70% dei dati di *training*, abbiamo deciso di riaddestrarlo sull'intero *training set* (quindi, tutti i 5000 post). Infatti, a questo punto i dati di *validation* non erano più necessari, e li abbiamo utilizzati come dati di addestramento ulteriori.

Poiché, in questo caso, il *training* sarebbe avvenuto su più dati, addestrare per 6 epoche (cioè il numero che ha permesso di raggiungere l'F1-score maggiore) non avrebbe avuto senso, perché dimensioni diverse del *training set* richiedono durate di addestramento diverse. Per scegliere il numero di epoche più adatto, dunque, abbiamo adottato il seguente procedimento:

1. selezionare l'epoca con F1-score migliore sul *validation set*, cioè l'epoca 6;
2. controllare la *training loss* corrispondente, cioè la *loss* ottenuta sul dataset di addestramento in quella stessa epoca: la *training loss* è di 0.040;
3. Riaddestrare sull'intero *training set* finché la *training loss* non raggiunge lo stesso valore (cioè quello dell'epoca con F1-score maggiore sul *validation set*).

Questo metodo ci ha portato ad addestrare il modello per 7 epoche: a quel punto, infatti, è stata raggiunta una *training loss* pari a 0.035 (molto vicina allo 0.040 che era stato raggiunto sul 70% dei dati). Il modello così ottenuto è stato, infine, valutato sul dataset di test, in cui ha ottenuto un F1-score di 0.782 (Tabella 4). Le singole performance sulle due classi (0 e 1) sono riportate nella matrice di confusione in Figura 7.

	precision	**recall**	**f1-score**	**support**
0	0.73	0.89	0.80	489
1	0.87	0.68	0.76	511
accuracy			0.78	1000
macro avg	0.80	0.79	**0.78**	1000
weighted avg	0.80	0.78	0.78	1000

Tabella 4: Risultati classificazione

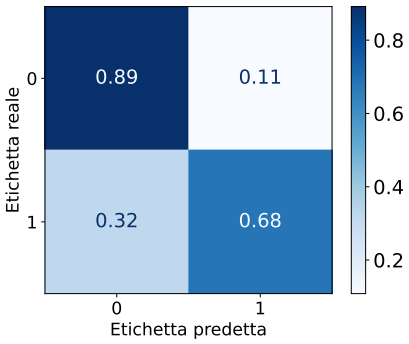


Figura 7: Confusion matrix

In Tabella 5 sono riportati i migliori dieci risultati ottenuti dai partecipanti a questo task, come condivisi dagli organizzatori[11]. Il nostro modello, dunque, si posiziona al nono posto sui 21 partecipanti totali, e al di sopra della baseline di 0.669.

[11] https://drive.google.com/file/d/1tlvVTTmNXPpx65XZpB_WRggPckzGqQFc/view

Rank	Team	F1
1	LCTs_3	0.8108
2	LCTs_2	0.8000
3	O-Dang_1	0.7959
4	DH-FBK_1	0.7950
5	extremITA_2	0.7942
6	O-Dang_2	0.7920
7	DH-FBK_2	0.7837
8	O-Dang_3	0.7804
9	LCTs_1	0.7709
10	CHILab_2	0.7525

Tabella 5: Top 10 HODI Evalita (in pedice al nome del team, il numero della run)

5 Explainability

Come anticipato nella sezione 3.1, il sub-task B ha come obiettivo l'*explainability*, cioè comprendere le motivazioni che hanno portato il modello a classificare i post come omotransfobici o meno. Infatti, la maggior parte dei sistemi decisionali che vengono impiegati oggigiorno, compresa l'architettura Transformer che sta alla base del modello da noi scelto, risultano essere piuttosto opachi, poiché la loro logica e i loro criteri decisionali non vengono resi espliciti e non risultano pienamente comprensibili. Per questo motivo, si utilizzano dei metodi di *explainability*, o *interpretability*, cioè metodi in grado di spiegare il modello e il significato delle sue decisioni in termini comprensibili agli esseri umani (Doshi-Velez e Kim, 2017).

Seguendo la survey pubblicata da Guidotti et al. nel 2018, possiamo individuare due tipi di modelli: quelli che sono intrinsecamente interpretabili (*decision tree*, classificatori a regole e modelli lineari), e quelli a *black box*, interpretabili a posteriori, come le reti neurali. Nel caso dei primi, infatti, è il design stesso del sistema ad essere comprensibile a un essere umano: un *decision tree*, ad esempio, è un grafo composto da nodi interni che rappresentano verifiche su determinate *features* (ad esempio, se una certa variabile del dato in input ha un valore minore, uguale o maggiore di una soglia), e nodi foglia che indicano l'etichetta della classe. Seguendo il percorso dalla radice alla foglia, si ottiene il risultato e, ripercorrendo il ramo, si può comprendere come si è arrivati a una certa decisione. Con una rete neurale, invece, è impossibile ottenere e ripercorrere in modo immediato il ragionamento alla base delle decisioni finali.

Nei modelli spiegabili per design, inoltre, l'interpretabilità è globale, cioè il modello è completamente interpretabile: siamo in grado di comprendere la logica alla base di tutte le possibili decisioni. Le *black box*, invece, possono essere interpretate globalmente o localmente, in base al tipo di *explainer* che si sceglie: l'interpretabilità è locale quando è possibile capire soltanto le ragioni per un output specifico che viene selezionato in precedenza.

5.1 Explainer scelti

Il modello da noi selezionato per la risoluzione di questo task è, chiaramente, una *black box*; perciò, è possibile adottare sia *explainer* locali che globali. In questo caso, sono stati utilizzati gli *explainer* LIME e *Integrated Gradient*.

5.1.1 LIME

Introdotto per la prima volta in *"Why Should I Trust You?": Explaining the Predictions of Any Classifier* (Ribeiro et al., 2016a), LIME sta per Local Interpretable Model-agnostic Explanations. Si tratta di un *explainer* locale, che, cioè, è in grado di spiegare soltanto le singole decisioni prese dal modello. L'intuizione alla base di questo metodo è che, per comprendere la *black box*, è possibile perturbare l'input e vedere come cambia la predizione. Questo facilita l'interpretabilità, perché l'input è, solitamente, più comprensibile agli umani (può trattarsi, ad esempio, di parole o parti di immagini), anche se poi il modello usa come *features* rappresentazioni più complesse (come *embeddings* di parole o sequenze di pixel). Perciò, sulla base dell'esempio da spiegare vengono creati degli esempi sintetici, che sono perturbazioni dell'originale (ottenute, ad esempio, rimuovendo parole o nascondendo parti dell'immagine). I dati perturbati vengono poi classificati dal modello *black box*, per vedere come i cambiamenti nell'input generino predizioni diverse. Su questi nuovi dati etichettati viene poi addestrato un modello interpretabile, come un modello lineare: quest'ultimo, quindi, tenderà ad approssimare il comportamento della *black box*, poiché è addestrato sulle sue predizioni. Tale approssimazione, tuttavia, è locale: il modello semplice

è in grado di imitare quello oscuro soltanto per l'istanza scelta e per quelle simili ad essa, ma non riesce ad approssimarlo globalmente.

Nel caso del riconoscimento di immagini, ad esempio, la singola immagine viene suddivisa negli elementi che la compongono. In seguito, come mostrato in Figura 8, vengono generati gli esempi sintetici nascondendo alcune di queste componenti. Ogni esempio perturbato viene poi classificato dal modello *black box* (nell'esempio in figura, il modello restituisce la probabilità che l'immagine contenga una raganella), e su questo dataset viene addestrato un modello lineare, pesato localmente (cioè, le istanze più simili all'esempio originale sono più importanti di quelle molto diverse). Il modello lineare è in grado di dirci quale *feature* è più importante nella classificazione, perché essa avrà un coefficiente più alto: perciò, le *features* che contribuiscono più positivamente alla classificazione permettono di spiegare la decisione della *black box* per quella istanza e per quelle simili ad essa.

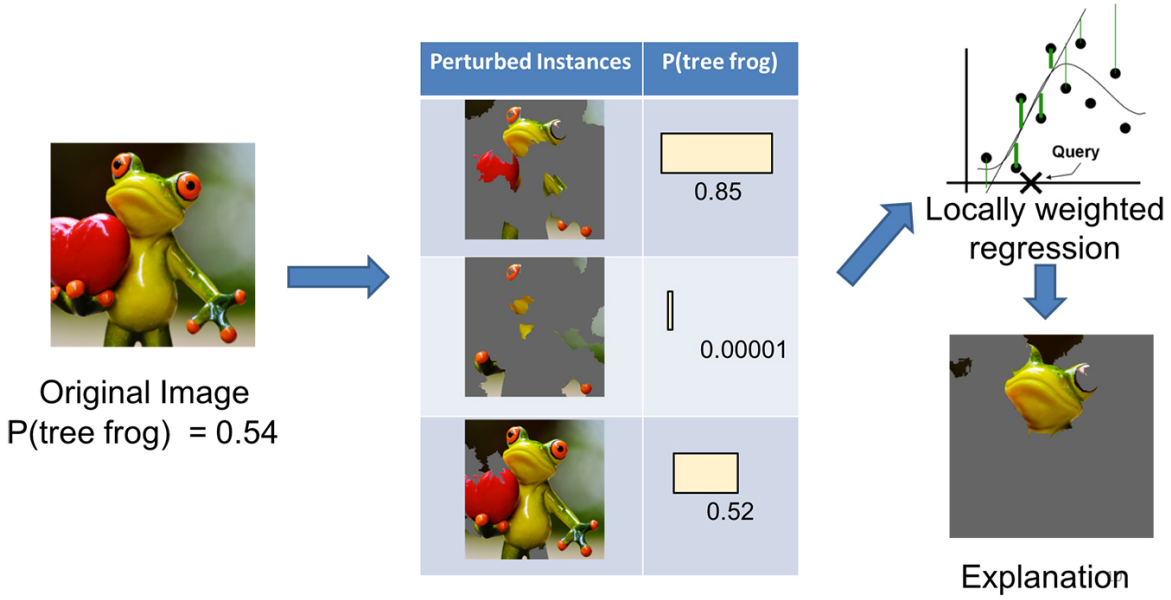


Figura 8: Spiegazione del funzionamento di LIME. Fonte: Ribeiro et al., 2016b

5.1.2 Integrated Gradient

Integrated Gradient è stato introdotto per la prima volta nel paper *Axiomatic Attribution for Deep Networks* (Sundararajan et al., 2017). A differenza di LIME, si tratta di un *explainer* globale e può essere applicato solo a modelli differenziabili (quali le reti neurali). Infatti, questo metodo parte da un input di base, chiamato *baseline*, cioè un punto di inizio che non fornisce alcuna informazione: per il riconoscimento di immagini, ad esempio, la *baseline* può essere un'immagine tutta nera, mentre per i task di NLP la *baseline* può essere formata soltanto dai token che segnalano l'inizio e la fine di una frase (ad esempio, [START] ed [END]). In seguito, si costruisce un percorso che va dalla *baseline* all'input effettivo. Tale percorso consiste semplicemente in una serie di modifiche consecutive dell'input di partenza (per semplificare, è come se, nel caso dei task di NLP, a ogni passo si aggiungesse un nuovo token alla *baseline*).

Per ogni diversa versione dell'input, il modello calcola una predizione. Questo ci permette di vedere quanto la nuova *feature* aggiunta in quella versione abbia influenzato la decisione finale. Nello specifico, il contributo di ogni *feature* in input si ottiene calcolando il gradiente del modello rispetto a quella *feature*.

5.2 Interpretazioni delle predizioni del modello finale

L'approccio che abbiamo adottato per cercare di comprendere le motivazioni dietro alle scelte del modello consiste, innanzitutto, nell'individuare a mano alcuni esempi interessanti: nello specifico, abbiamo selezionato tre tweet: uno classificato correttamente come omotransfobico, uno classificato incorrettamente come omotransfobico e uno classificato incorrettamente come non omotransfobico.

La Tabella 6 mostra un esempio di post in cui è stato correttamente individuato l'*hate speech* (cioè, il modello ha predetto l'etichetta '1'). L'*explainer* LIME spiega questa predizione in modo plausibile, assegnando maggiore importanza ai token che formano un insulto omofobo. Integrated Gradient, invece, non assegna un'importanza elevata a queste stesse parole, ma esse hanno comunque un punteggio più alto della maggior parte degli altri token nella frase.

Token	sai	sparare	ma	sei	rotto	in	culo
LIME	-0.03	-0.01	0.18	0.01	0.43	0.03	0.30
Integrated Gradient	-0.27	-0.18	0.01	-0.15	-0.06	-0.14	-0.06

Tabella 6: Spiegazioni di un tweet classificato correttamente come omotransfobico

La Tabella 7, invece, fa riferimento a un tweet che non è stato classificato correttamente: non è omotransfobico ma il modello l'ha considerato tale. Anche stavolta, LIME riesce a dare una spiegazione accettabile di questo errore, assegnando un'importanza maggiore ai token che formano un insulto omofobo. Perciò, non sorprende che il Language Model abbia classificato il post come *hate speech*, data la presenza di tale insulto; tuttavia, la frase da sola non veicola necessariamente un messaggio omotransfobico, ma solo una constatazione: servirebbe il contesto per inquadrare correttamente questo tweet e l'eventuale *hate speech*. Integrated Gradient, invece, non spiega la predizione del modello tramite questi token, che, in realtà, hanno un punteggio più basso di tutti gli altri token presenti nella frase.

	pensa	,	prima	di	diceva	ricchi	##one	.	..1	..2
LIME	-0.04	0.03	-0.10	0.11	-0.15	0.33	0.13	0.06	-0.02	0.04
Integrated Gradient	-0.05	0.08	-0.11	0.09	-0.02	-0.19	-0.15	-0.13	-0.06	-0.04

Tabella 7: Spiegazioni di un tweet classificato incorrettamente come omotransfobico

Infine, la Tabella 8 mostra le spiegazioni di un'altra predizione scorretta: il messaggio è omotransfobico ma il modello non è riuscito a riconoscerlo. Per spiegare questa classificazione, LIME assegna un punteggio alto a parole in realtà neutre, quali "sarà" e "anche"; i token che costituiscono un insulto omofobo, invece, hanno chiaramente un punteggio basso, perché sono quelle che avrebbero portato il modello alla classificazione opposta. Per quanto riguarda Integrated Gradient, invece, l'insulto omofobo ha un punteggio alto, come se avesse avuto un'influenza decisiva nella predizione finale, che però è '0'.

Token	sarà	cul	##atto	##ne	anche	lui	che	c	è	di	male
LIME	0.14	-0.02	-0.16	0.03	0.23	-0.01	-0.15	0.03	-0.08	-0.02	0.03
Integrated Gradient	0.07	0.31	0.12	-0.05	0.02	0.05	0.03	-0.11	0.04	0.07	0.01

Tabella 8: Spiegazioni di un tweet classificato incorrettamente come non-omotransfobico

Dagli esempi sopra riportati, si nota come i due *explainer* considerati siano spesso in disaccordo, dando importanza a token diversi. In particolare, le spiegazioni fornite da LIME risultano essere quelle più plausibili: i termini omofobi contribuiscono molto positivamente quando il modello riscontra omofobia, e molto negativamente quando non ne riscontra. Invece, Integrated Gradient dà spiegazioni meno condivisibili. Da ciò si conclude che, per un modello che risolve un task complesso quale l'*hate speech detection*, è meglio utilizzare un'*explainer* locale, il quale, concentrandosi sulle singole istanze, riesce a fornirne un'interpretazione più adatta.

6 Conclusioni

Per risolvere il task di *homotransphobia detection* proposto in Evalita 2023, abbiamo sperimentato diversi Large Language Model disponibili su Hugging Face. Ne è emerso che il modello che permette di ottenere la performance migliore è BERT finetunato sull'italiano, che, in questo task, risulta superare DistilBERT e le versioni multilingue. Una piccola ricerca degli iperparametri ha portato a stabilire che quelli di default si confermano i più adatti all'addestramento. Il modello selezionato è stato poi testato sul *test set*, dove ha raggiunto un F1-score dello 0.782.

Infine, le predizioni del modello sono state analizzate con strumenti di *explainability*, per comprendere quali parole sono state più determinanti. Gli *explainer* utilizzati sono LIME (locale) e Integrated Gradient (globale). Dagli esempi analizzati, è emerso che LIME permette di trovare spiegazioni più plausibili e condivisibili: probabilmente, è più semplice trovare una spiegazione alla singola istanza che a tutte le predizioni nel complesso. Chiaramente, gli insulti omofobi tendono sempre a indirizzare il modello a predire l'*hate speech*, anche laddove la frase nel complesso non veicola un messaggio di omotransfobia. Questo dimostra la complessità del task, in cui anche le piccole sfumature di significato e il contesto di utilizzo determinano il vero intento del tweet.

Riferimenti bibliografici

Aluru, Sai Saket, Binny Mathew, Punyajoy Saha e Animesh Mukherjee (2020). «Deep Learning Models for Multilingual Hate Speech Detection». In: *arXiv preprint arXiv:2004.06465*.

Bahdanau, Dzmitry, Kyunghyun Cho e Yoshua Bengio (2014). «Neural Machine Translation by Jointly Learning to Align and Translate». In: *CoRR*.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent e Christian Jauvin (2003). «A Neural Probabilistic Language Model». In: *Journal of Machine Learning Research* 3, pp. 1137–1155.

Cavnar, William B, John M Trenkle et al. (1994). «N-gram-based text categorization». In: *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. Vol. 161175. Las Vegas, NV, p. 14.

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau e Yoshua Bengio (2014). «On the Properties of Neural Machine Translation: Encoder–Decoder Approaches». In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. A cura di Dekai Wu, Marine Carpuat, Xavier Carreras e Eva Maria Vecchi. Doha, Qatar: Association for Computational Linguistics, pp. 103–111. DOI: 10.3115/v1/W14-4012.

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk e Yoshua Bengio (2014). «Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. A cura di Alessandro Moschitti, Bo Pang e Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

Cover, Thomas e Peter Hart (1967). «Nearest neighbor pattern classification». In: *IEEE transactions on information theory* 13.1, pp. 21–27.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee e Kristina Toutanova (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805[cs].

Doshi-Velez, Finale e Been Kim (2017). *Towards A Rigorous Science of Interpretable Machine Learning*. DOI: 10.48550/ARXIV.1702.08608.

Elman, Jeffrey L. (1990). «Finding Structure in Time». In: *Cognitive Science* 14.2, pp. 179–211. ISSN: 0364-0213. DOI: 10.1016/0364-0213(90)90002-E.

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Franco Turini, Dino Pedreschi e Fosca Giannotti (2018). *A Survey Of Methods For Explaining Black Box Models*. arXiv: 1802.01933[cs].

Hinton, Geoffrey, Oriol Vinyals e Jeff Dean (2015). *Distilling the Knowledge in a Neural Network*. DOI: 10.48550/arXiv.1503.02531. arXiv: 1503.02531 [cs, stat].

Hochreiter, Sepp e Jürgen Schmidhuber (1997). «Long Short-Term Memory». In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Jahan, Md Saroar e Mourad Oussalah (2023). «A systematic review of hate speech automatic detection using natural language processing». In: *Neurocomputing* 546, p. 126232. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2023.126232.

Joachims, Thorsten (1998). «Text categorization with support vector machines: Learning with many relevant features». In: *European conference on machine learning*. Springer, pp. 137–142.

Jurafsky, Dan e James H. Martin (2023). *Speech and Language Processing*. 3rd ed. draft.

Li, Qian, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S. Yu e Lifang He (2021). *A Survey on Text Classification: From Shallow to Deep Learning*. DOI: 10.48550/arXiv.2008.00364. arXiv: 2008.00364 [cs].

Maron, Melvin Earl (1961). «Automatic indexing: an experimental inquiry». In: *Journal of the ACM (JACM)* 8.3, pp. 404–417.

Mikolov, Tomas, Kai Chen, G. Corrado e J. Dean (2013). «Efficient Estimation of Word Representations in Vector Space». In: *International Conference on Learning Representations*.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado e Jeff Dean (2013). «Distributed Representations of Words and Phrases and Their Compositionality». In: vol. 26. Curran Associates, Inc.

Nockleby, John T. (2000). *Hate Speech*. In: *Encyclopedia of the American Constitution*. A cura di Adam Winkler Leonard W. Levy Kenneth L. Karst. Second. Vol. 3. New York USA: Macmillan, pp. 1277–1279.

Nozza, Debora (2021). «Exposing the limits of Zero-shot Cross-lingual Hate Speech Detection». In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. ACL-IJCNLP 2021. Online: Association for Computational Linguistics, pp. 907–914. DOI: 10.18653/v1/2021.acl-short.114.

Nozza, Debora, Federico Bianchi e Giuseppe Attanasio (2022). «HATE-ITA: Hate Speech Detection in Italian Social Media Text». In: *Proceedings of the 6th Workshop on Online Abuse and Harms*. Association for Computational Linguistics.

Nozza, Debora, Alessandra Teresa Cignarella, Greta Damo, Tommaso Caselli e Viviana Patti (2023). ≪HODI at EVALITA 2023: Overview of the Homotransphobia Detection in Italian Task≫. In: *Proceedings of the Eighth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2023)*. Parma, Italy: CEUR.org.

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee e Luke Zettlemoyer (2018). ≪Deep Contextualized Word Representations≫. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. A cura di Marilyn Walker, Heng Ji e Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

Rajaraman, Anand e Jeffrey David Ullman (2011). ≪Data Mining≫. In: *Mining of Massive Datasets*. Cambridge University Press, pp. 1–17. DOI: 10.1017/CBO9781139058452.002.

Ribeiro, Marco Tulio, Sameer Singh e Carlos Guestrin (2016a). *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. DOI: 10.48550/ARXIV.1602.04938.

Sanh, Victor, Lysandre Debut, Julien Chaumond e Thomas Wolf (2020). *DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter*. DOI: 10.48550/arXiv.1910.01108. arXiv: 1910.01108 [cs].

Straka, Milan e Jana Straková (2017). ≪Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe≫. In: *Proceedings of the CoNLL 2017 shared task: Multilingual parsing from raw text to universal dependencies*, pp. 88–99.

Sundararajan, Mukund, Ankur Taly e Qiqi Yan (2017). *Axiomatic Attribution for Deep Networks*. DOI: 10.48550/ARXIV.1703.01365.

Sutskever, Ilya, Oriol Vinyals e Quoc V Le (2014). ≪Sequence to Sequence Learning with Neural Networks≫. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser e Illia Polosukhin (2017). ≪Attention is All you Need≫. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc.

Zhang, Yin, Rong Jin e Zhi-Hua Zhou (2010). ≪Understanding bag-of-words model: a statistical framework≫. In: *International journal of machine learning and cybernetics* 1, pp. 43–52.

Sitografia

Newton, Casey (2020). *Facebook will pay $52 million in settlement with moderators who developed PTSD on the job — theverge.com*. [Visitato il 03 gennaio 2024]. URL: https://www.theverge.com/2020/5/12/21255870/facebook-content-moderator-settlement-scola-ptsd-mental-health.

Ribeiro, Marco Tulio, Sameer Singh e Carlos Guestrin (2016b). *Local Interpretable Model-Agnostic Explanations (LIME): An Introduction — oreilly.com*. [Visitato il 29 dicembre 2023]. URL: https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/.