Università degli studi di
Milano-Bicocca

Text Mining Search

Progetto Finale

Toxic Comment Classification

Autori:
Lorenzo Famiglini - 838675 - l.famiglini@campus.unimib.it
Giorgio Bini - 838674 - g.bini@campus.unimib.it

Febbraio 2020

Abstract

Il progetto si pone l'obiettivo di affrontare un problema di *text classification* al fine di individuare i commenti *tossici*, ovvero volgari, trasgressivi ed offensivi, degli utenti di Wikipedia. Sono stati implementati diversi modelli di reti neurali, inglobando in essi strati ricorrenti, convoluzionali e densi. Verrà inoltre definito lo sviluppo di diverse tecniche di rappresentazione del testo, quali Embedding e POS, evidenziando l'impatto sulle performance degli algoritmi di classificazione nell'applicazione per ciascuna di esse.

1 Introduzione

Il dataset utilizzato, come detto, consiste in una raccolta di commenti degli utenti di Wikipedia in lingua inglese, ed è disponibile su Kaggle sotto il nome di *Toxic Comment Classification Challenge*. Tra le feature disponibili per tale dataset è stata presa in considerazione *toxic*, che è la variabile target, in questo caso binaria. In totale, vi sono 159571 commenti ma solo il 10% di essi risulta essere *tossico*. Si presenta dunque un problema di classe sbilanciata, che è stato opportunamente trattato come vedremo nella sezione dedicata all'approcio metodologico. Una prima fase di esplorazione dei dati viene di seguito proposta attraverso due *word cloud* (Fig.1), costruite dopo aver rimosso le *stop word*. Si evidenzia come la classe di interesse (i commenti *tossici*) sia caratterizzata da parole volgari, razziste ed offensive, in contrapposizione agli altri commenti, che invece registrano principalmente parole inerenti alla lettura di documenti e articoli.

Figura 1 - Word cloud per commenti tossici e non

Quello dell'identificazione dei commenti volgari ed offensivi è un task ampiamente sviluppato in letteratura, in quanto gli utenti di internet sono ad oggi sostanzialmente privi di difese verso l'aggressività e le offese gratuite che spesso si leggono in rete, specialmente nei social network. Un'identificazione automatica dei commenti *tossici*, perciò, potrebbe essere un primo passo per combattere

un fenomeno sempre più in diffusione. Si è deciso dunque di implementare diversi algoritmi di classificazione, sia basandosi sulle indicazioni fornite dalla letteratura, sia creando degli sviluppi innovativi. In particolare, si è deciso di sfruttare non solo la rappresentazione del testo tramite embedding per apprendere la semantica del testo, ma di inglobare all'interno dei modelli anche informazioni relative alla grammatica di ogni commento attraverso il POS Tagging.

2 L'approccio metodologico

In questa sezione saranno descritti gli approcci adottati per la preparazione del testo e, successivamente, verranno proposte le architetture di Deep Learning costruite per la classificazione.

2.1 Il preprocessing

La preparazione del testo è certamente un aspetto fondamentale nei task di *text classification*. La prima fase del preprocessing è consistita nella rimozione delle seguenti componenti del testo:

- *emoji*
- *numeri*
- *web link*
- *caratteri speciali*
- *punteggiatura*

Un'ulteriore trasformazione dei dati è stata quella di normalizzare il testo riportando ogni carattere in formato *lower case*. Infine, visto che nella lingua inglese si utilizzano spesso gli apostrofi per le contrazioni, si è deciso di riportare tali forme lessicali alla loro rispettiva forma non contratta. Una volta effettuate tali operazioni, è stata effettuata la tokenizzazione; in totale, si contano 184883 parole uniche. Tale preparazione del testo, è stata utilizzata per la creazione degli embedding. Sono stati provati 3 tipi di embedding pre-addestrati:

- *GloVe*: la creazione dei vettori densi avviene considerando le relazioni nella *Term-Context Matrix*.
- *Word2Vec*: intuitivamente, lo si può definire come un *n-gram Language Model*, in cui gli embedding vengono ottenuti attraverso Skip Gram o Common Bag Of Words (CBOW).
- *fastText*: è una libreria disponibile in Python che permette di creare vettori densi dal testo seguendo i metodi descritti nel paper *Advances in Pre-Training Distributed Word Representations*[1].

La rappresentazione vettoriale migliore, in termini di performance dei modelli, è stata ottenuta grazie al GloVe, pre-addestrato su un dataset di 6 miliardi di parole. In particolare, è stata confrontata la versione del GloVe che conta 100 feature con quella che ne conta 200, e quest'ultima ha registrato risultati migliori. Una volta importati gli embedding pre-addestrati con il Glove, è stato deciso di rappresentare, su un piano cartesiano a due dimensioni, alcune delle parole più frequenti suddivise per tipologia di commento. Una volta ottenuto il vettore denso per ciascuna parola, è stata applicata una tecnica per la riduzione della dimensionalità chiamata $TSNE$. Tale metodo, effettua trasformazioni non lineari al fine di visualizzare dati con una elevata dimensionalità. Una volta selezionate 10 parole, di cui 5 relative ai commenti *tossici*, è stata effettuata una rappresentazione per mostrare come termini con una similarità semantica si trovano anche vicini nello spazio vettoriale (Figura 2). L'unica eccezione la si trova per la parola "hate", probabilmente perchè nel corpus utilizzato per la creazione degli embedding, tale termine si è spesso trovato in concomitanza con parole come "talk" e "think".

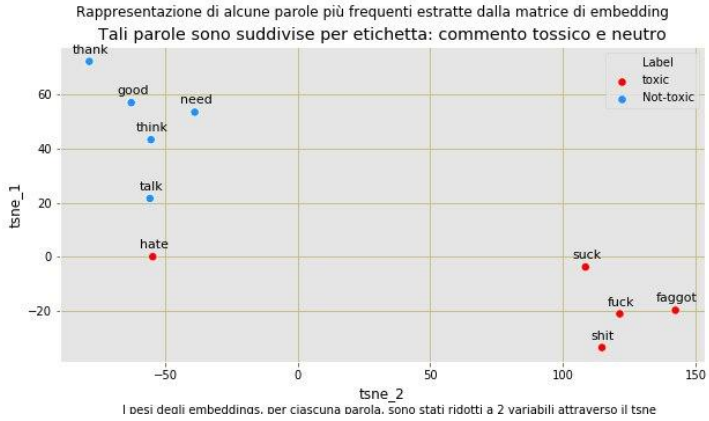

Figura 2 - TSNE per la visualizzazione degli embedding

Un'ulteriore rappresentazione del testo ricavata è stata quella del POS Tagging. Per far ciò è stata utilizzata la libreria spaCy, disponibile in Python, che utilizza come tag set per la lingua inglese la versione Penn Treebank. L'applicazione del POS Tagging al testo è avvenuta a seguito della rimozione delle stop word e della punteggiatura, e questo ha registrato miglioramenti nelle performance. Ulteriori trasformazioni dei dati, quali lemmatizzazione e stemming, non sono invece risultate efficaci per questo task, e pertanto non sono state utilizzate. Inoltre, dopo aver osservato che il valore medio della lunghezza dei commenti è di 100 parole, si è deciso di adottare tale valore per la dimensione dei vettori. Naturalmente, i commenti con meno di 100 parole saranno completati con uno zero-padding, mentre i commenti con più di 100 parole verranno troncati. Per quanto riguarda la spartizione dei dati, è stata effettuata una suddivisione del dataset in train (80%) e validation (20%). Per ogni modello implementato, i parametri dello strato di embadding non sono stati allenati, in quanto la re-

lazione tra le parole è risultata discriminativa per risolvere tale task utilizzando i valori pre-addestrati. Inoltre, per affrontare il problema della classe sbilanciata, è stato sviluppato un generatore dei dati, che permette di bilanciare il mini-batch per l'allenamento dei modelli stratificando il campione per la variabile target.
Per quanto riguarda l'ottimizzazione dei modelli, viene segnalato che alcuni iperparametri sono stati oggetto di ottimizzazione, e per effettuare il tuning, si è deciso di adottare il servizio *SigOpt* che permette di sfruttare tecniche di AutoML. Essendoci sia iperparametri continui che discreti, il processo di ottimizzazione avviene utilizzando come modello surrogato il Random Forest [2]. Per ogni modello, il processo di ottimizzazione è stato eseguito valutando 30 configurazioni diverse degli iperparametri. L'ottimizzazione è un'operazione che ha permesso non solo di ottenere un miglioramento delle performance, ma anche di confrontare i vari modelli a parità di budget (insieme di configurazioni valutate).

2.2 CNN

Il primo modello, consiste in una rete convoluzionale. L'inizializzazione casuale dei parametri è avvenuta utilizzando la distribuzione glorot normal. L'architettura viene, nel dettaglio, di seguito descritta.

1. Embedding layer con 200 features.

2. Convoluzione 1D con 32 filtri, dimensione del kernel pari a 10 e stride pari ad 1. Vi è in coda la funzione di attivazione ReLU. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 2 e stride pari a 2.

3. Convoluzione 1D con 64 filtri, kernel 10 e stride pari ad 1. Anche in questo caso vi è l'utilizzo della funzione ReLu. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 1 e stride pari a 1. Inoltre, vi è un Dropout con coefficiente pari ricercato nel range $d_1 \in [0, 0.85] \subset \mathrm{R}$. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 0.85.

4. Convoluzione 1D con 128 filtri, dimensione del kernel pari a 5 e stride pari ad 1. Vi è in coda la funzione di attivazione ReLU. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 4 e stride pari a 4.

5. Strato *Flatten* che ridimensiona i dati in un vettore 1D.

6. Strato *fully connected* con un numero di neuroni ricercato nel range $n_1 \in [32, 512] \subset \mathrm{N}$. Il valore ottimale ottenuto a seguito del processo di ottimizzazione è pari a 260. Anche in questo caso si utilizza la funzione ReLu.

7. Strato *fully connected* con un numero di neuroni ricercato nel range $n_2 \in [32, 512] \subset \mathrm{N}$. Il valore ottimale in questo caso è risultato pari a 311. In coda vi è la funzione ReLu. Inoltre, viene applicato il Dropout con coefficiente pari ricercato nel range $d_2 \in [0, 0.85]. \subset \mathrm{R}$. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 0.36.

8. Strato denso finale con due neuroni. La funzione di attivazione in questo caso è la *softmax* in quanto la loss utilizzata è la *categorical cross entropy*.

Le scelte architetturali di questa rete seguono la logica (adottata da reti neurali gran successo come l'Alex Net, la VGG16, ecc...) secondo la quale risulta conveniente, con il passare dei layer, aumentare il numero di filtri e ridurre la dimensione spaziale. Lo sviluppo della rete in profondità, infatti, permette di creare molteplici feature map, e ciascuna di esse si specializza nel captare determinate informazioni rappresentative dei dati a disposizione. Il modello è stato allenato in 50 epoche, e come ottimizzatore, è stato utilizzato il Nadam con coefficienti β_1=0.9, β_2=0.99. Anche il learning rate è stato oggetto di ottimizzazione, e dopo una ricerca nel range $lr \in [0.0005, 0.1] \subset \mathrm{R}$, il valore ottimale è risultato 0.0005. Per quanto riguarda la funzione surrogata, è stata massimizzata la seguente quantità:

$$Objective\ Function = \frac{1}{|acc_{train} - acc_{validation}|} + F1_{validation} \tag{1}$$

La scelta di tale funzione ha un duplice obiettivo. In primis, si cerca di evitare un overfitting dei modelli, andando a minimizzare la differenza di accuratezza tra training e validation. In secondo luogo, si cerca di massimizzare la F1-measure in validation, così da poter costruire modelli robusti verso la classe meno rappresentata.

2.3 CNN + LSTM

La seconda rete creata consiste in un modello simile al primo, al quale è stata aggiunto uno strato ricorrente per catturare la relazione tra le parole. L'architettura viene di seguito descritta.

1. Embedding layer con 200 features.

2. Convoluzione 1D con 32 filtri, dimensione del kernel pari a 10 e stride pari ad 1. Vi è in coda la funzione di attivazione ReLU. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 2 e stride pari a 2.

3. Convoluzione 1D con 64 filtri, kernel 10 e stride pari ad 1. Anche in questo caso vi è l'utilizzo della funzione ReLu. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 1 e stride pari a 1. Inoltre, vi è un Dropout con coefficiente pari ricercato nel range d_1

$\in$ [0, 0.85] $\subset$ R. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 0.61.

4. Convoluzione 1D con 128 filtri, dimensione del kernel pari a 5 e stride pari ad 1. Vi è in coda la funzione di attivazione ReLU. Successivamente viene applicato uno strato di Max Pooling1D con finestra di dimensione 4 e stride pari a 4.

5. Strato ricorrente $LSTM$ con numero di unità $lstm \in [30, 500] \subset$ N. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 390. A tale strato viene applicata una regolarizzazione con coefficienti $l1 = 0.01$ e $l2 = 0.01$.

6. Strato *fully connected* con un numero di neuroni ricercato nel range $n_1 \in [32, 512] \subset$ N. Il valore ottimale ottenuto a seguito del processo di ottimizzazione è pari a 39. Anche in questo caso si utilizza la funzione ReLu. A tale strato viene applicata una regolarizzazione con coefficienti $l1 = 0.1$ e $l2 = 0.01$. Inoltre, viene applicato il Dropout con coefficiente pari ricercato nel range $d_2 \in [0, 0.85] \subset$ R. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 0.60.

7. Strato denso finale con due neuroni. La funzione di attivazione in questo caso è la *softmax* in quanto la loss utilizzata è la *categorical cross entropy*.

Anche in questo caso è stato utilizzato il Nadam, con i medesimi iperparametri. Il valore del learning rate, a seguito del processo di ottimizzazione, in questo caso è risultato pari a $lr = 0.0005$.

2.4 Multi-input Neural Network

Un ultimo modello è stato creato considerando due diverse rappresentazioni del testo: l'embedding e il POS Tagging. L'input di embedding sostanzialmente è lo stesso descritto nel modello *CNN + LSTM* fino al punto 5 (compreso, in quanto vi è anche uno strato ricorrente). In questo caso, gli iperparametri ottimali a seguito del processo di ottimizzazione sono risultati i seguenti: $d_1 = 0.62$, $lstm = 135$. Vi è in coda a tale insieme di layer uno strato di Dropout con coefficiente d_2 che, dopo l'ottimizzazione, è risultato pari a 0. Un secondo input del modello, che prende invece in considerazione la rappresentazione POS Tagging, viene di seguito descritto.

1. Strato ricorrente $LSTM$ con numero di unità $lstm_2 \in [30, 500] \subset$ N. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 455. Inoltre, viene applicato il Dropout con coefficiente pari ricercato nel range $d_3 \in [0, 0.85] \subset$ R. Il suo valore ottimale, dopo aver eseguito il processo di ottimizzazione, è risultato pari a 0.14.

2. Strato *fully connected* con un numero di neuroni ricercato nel range $n_1 \in [32, 512] \subset$ N. Il valore ottimale ottenuto a seguito del processo di

ottimizzazione è pari a 40. Anche in questo caso si utilizza la funzione ReLu.

A questo punto, i due output parziali vengono connessi in un unico strato denso con numero di neuroni $n_2 \in [32, 512] \subset \mathrm{N}$. Il valore ottimale ottenuto a seguito del processo di ottimizzazione è pari a 421. Anche in questo caso lo strato denso finale ha due neuroni e funzione di attivazione *softmax*. L'ottimizzatore è sempre il Nadam, con gli stessi valori degli iperparametri dei modelli precedenti. Il valore del learning rate, in questo caso, è stato fissato a $lr = 0.01$.

3 Risultati e Conclusioni

In questa sezione, verranno esposti i risultati ottenuti per ogni modello ed essi verranno interpretati alla luce degli obiettivi prefissati. In particolare, i valori di accuratezza ed F1-Score vengono riportati rispettivamente in Tabella 1 e 2.

	CNN	**CNN + LSTM**	**Multi-Input NN**
Train	0.99	0.98	0.97
Validation	0.95	0.98	0.97

Tabella 1 - Valori di accuratezza calcolati in Training e Validation

	CNN	**CNN + LSTM**	**Multi-Input NN**
Train	0.96	0.91	0.87
Validation	0.74	0.91	0.88

Tabella 2 - Valori di F1-Score calcolati in Training e Validation

In Tabella 3 vengono invece descritti i valori di Precision e Recall per la classe meno frequente (quella dei commenti *tossici*) dei tre modelli.

	CNN	**CNN + LSTM**	**Multi-Input NN**
Precision Train	0.93	0.94	0.81
Precision Validation	0.75	0.94	0.81
Recall Train	1	0.89	0.94
Recall Validation	0.73	0.88	0.95

Tabella 3 - Valori di Precision e Recall calcolati in Training e Validation

I risultati, evidenziano come il processo di ottimizzazione sia stato efficace nel trovare dei modelli che sono ben performanti, e che al tempo stesso riescono a generalizzare i risultati anche nel validation set. Oltre ai valori puntuali di accuratezza, riportati in Tabella 1, viene di seguito proposto un confronto tra le reti riportando anche l'intervallo di confidenza di Wilson con α pari a 0.05 (Fig.3). In questo modo si ha una visione più completa sulle performance dei modelli, in quanto si ha una misura della variabilità dell'errore nelle previsioni.

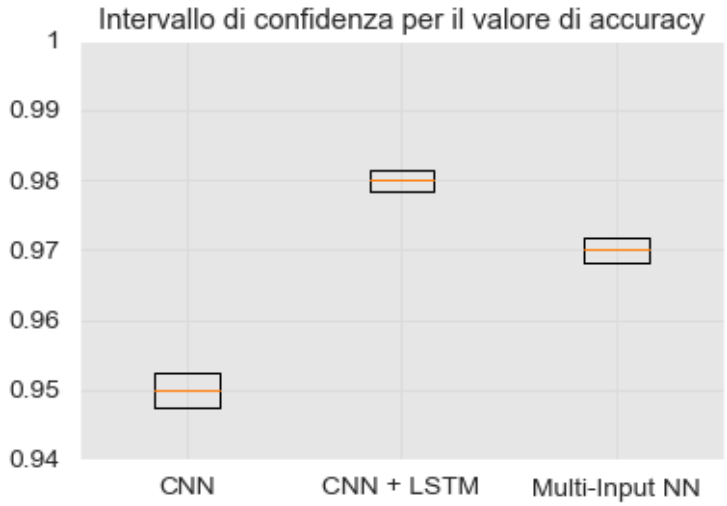

Figura 3 - Intervallo di confidenza per il valore di accuracy

La Figura 3 evidenzia come, dal punto di vista dell'accuracy, il miglior modello risulta essere *CNN + LSTM* e la differenza rispetto alle altre reti è statisticamente significativa per un livello di confidenza del 95%. Tuttavia, essendo un problema di classe sbilanciata, occorre riporre attenzione anche alle metriche di Precision, Recall e F1-Score. In particolare, possiamo vedere come in validation set il modello *CNN + LSTM* registri il più alto valore di F1-Score. D'altra parte, la rete *Multi-Input*, sfruttando il POS Tagging, è in grado di identificare una quota maggiore di commenti *tossici*, come è possibile vedere dal valore di Recall. L'utilizzo congiunto delle rappresentazioni embedding e POS, è dunque risultato utile in quanto, se si è maggiormente interessati ad identificare la classe meno frequente, si potrebbe utilizzare tale modello. Viceversa, se la necessità fosse quella di utilizzare un modello che sia più affidabile nel momento in cui la predizione fosse quella di commenti *tossici*, si dovrebbe utilizzare la rete *CNN + LSTM* in quanto registra il più alto valore di Precision.

4 Sviluppi futuri

Ulteriori sviluppi futuri potrebbero includere l'arricchimento dei dati a disposizione, al fine di rendere più robusti i modelli nella predizione. Inoltre, se si avesse a disposizione un budget maggiore nel processo di ottimizzazione, si potrebbero migliorare le prestazioni dei modelli. In particolare, la rete *Multi-Input*, avendo un'architettura più complessa rispetto alle altre, necessiterebbe di ulteriori iterazioni in fase di ottimizzazione. Infine, l'inserimento di particelle pragmatiche, come viene evidenziato in letteratura, potrebbe risultare efficace nel captare determinate sfumature nel linguaggio quali ironia e sarcasmo.

L'identificazione degli stessi, in alcuni commenti, potrebbe rendere robusti i modelli anche verso tali espressioni linguistiche.

References

[1] P. B. C. P. A. J. Tomas Mikolov, Edouard Grave, "Advances in Pre-Training Distributed Word Representations," 2017.

[2] S. C. Ian Dewancker, Michael McCourt, "Bayesian Optimization Primer."