BERT sentiment analysis 140

Marta Burocchi Cristina Coppola

Introduzione

L'obiettivo del nostro progetto è utilizzare l'architettura transformer BERT per fare sentiment analysis sul dataset *Sentiment140* [1], un dataset di tweet in inglese fornito dall'Università di Stanford.
La nostra relazione è strutturata come segue: nella prima sezione parleremo di BERT, delle architetture transformers e del fine-tuning di Bert per il nostro specifico task. Nella seconda sezione discuteremo brevemente il dataset utilizzato e come è stato pulito per adattarlo al task. Infine nell'ultima sezione commenteremo in modo dettagliato l'implementazione.

1 BERT e le architetture transformers

1.1 Le architetture Transformers

I transformers, basati sul paper "Attention is all you need"[2], sono un tipo particolare di reti neurali, nate a metà del 2017 e diventate tra le più interessanti tecniche per NLP(Natural language processing).
BERT, combinando l'architettura transformer con un grande pre-training non supervisionato, è stata la prima realizzazione a raggiungere lo stato dell'arte in molti task di NLP. I transformers sono utilizzati per diversi NLP tasks, tra cui la sentiment analysis; tuttavia possono essere adattati anche per altri task come ad esempio entity matching.

L'ambito di NLP è stato dominato per molto tempo dalle reti neurali RNNs(Recurrent Neural Networks). Le architetture per machine translations più utilizzate erano *seq2seq* o *encoder/decoder*, composte da due RNN, una per l'encoder, l'altra per il decoder.
Nonostante queste architetture sembrassero andar bene per la gestione di sequenze di dati, in realtà furono scoperti diversi problemi:

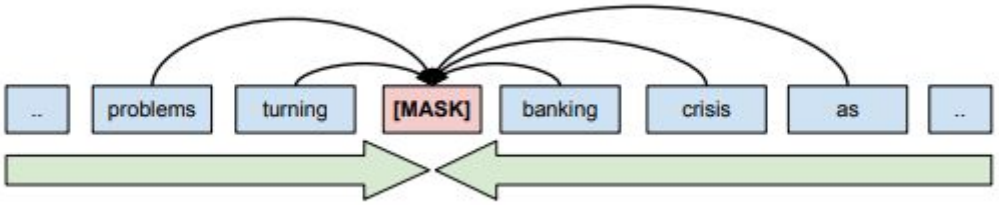
- il cosiddetto "collo di bottiglia" costituito dal vettore del contesto, che corrisponde all'input del decoder, il quale, basandosi solo su questo vettore, deve restituire come output una sequenza lunga N. È chiaro che maggiore è la lunghezza della sequenza in input, maggiore è la difficoltà nel trasferire la conoscenza relativa alla source sentence nella target sentence
- l'architettura RNN, a causa della sua natura sequenziale è difficile da parallelizzare e inoltre richiedere molto più tempo per il training rispetto ad una rete feed forward
- nelle frasi molto lunghe è difficile imparare dipendenze con le RNNs, anche usando LSTM e GRU. Più è lunga la sequenza, più è difficile imparare le dipendenze

Per risolvere il problema del collo di bottiglia viene introdotta una tecnica chiamata *“attention”*. L’intuizione di questo approccio si basa sull’idea di *“self-attention”*: una parola può essere rappresentata come una combinazione pesata delle parole vicine. Applicando un meccanismo di attenzione possiamo addestrare il language model a prestare attenzione alle parole rilevanti tra quelle vicine.

Rispetto all’architettura RNNs l’architettura Transformer rinuncia all’idea di costruire un modello basato solo sull’attention. I transformer mantengono la classica struttura encoder/decoder, ma sostituiscono la parte di RNN con un sub-layer chiamato “Multi Head Attention”,che permette al decoder di prestare attenzione solamente alle parole rilevanti della source sentence.
Avendo rimosso la ricorrenza, è stato necessario introdurre un’idea di ordinamento delle sequenze di input, attraverso un approccio “positional encoding” per ogni parola di input. Questo permette al modello di sfruttare sia la posizione di una parola, che la relativa distanza tra due parole.

1.2 BERT

BERT è un universal language model pre-addestrato su un gran numero di dati in formato testuale. L’acronimo BERT sta per “Bidirectional Encoder Representations from Transformers”. Durante il pre-addestramento BERT considera sia il contesto sinistro che il contesto destro di un token(da qui il termine “bidirectional”): questo può essere controintuitivo poichè la maggior parte dei task di training predicono il token successivo. Sfruttando anche il contesto destro il task training di BERT è leggermente diverso: invece di predire il “next token” cerca di predire un masked token all’interno di una sequenza.



Inoltre BERT è addestrato sul task *Next Sentence Prediction(NSP)*,in cui date due frasi in input stabilisce se la prima è seguita dalla seconda.

Gli addestramenti di BERT sono non-supervisionati, richiedono solamente un gran numero di dati di training. I dati etichettati sono necessari solo per fare il fine-tuning su task specifici.

1.3 Fine-tuning BERT per sentiment analysis

Il modo più diretto di utilizzare BERT è come classificatore binario su dati testuali. Nel nostro caso BERT prende in input una frase ed è in grado di stabilirne il sentiment. Per fare ciò al modello pre-addestrato di BERT è collegato un classificatore binario, costituito da una fully connected con 768 neuroni più 2 di output(1= sentiment positivo, 0= sentiment negativo).

Affinchè il modello creato possa essere addestrato eseguiamo un training supervisionato, utilizzando quindi un dataset etichettato.
Gli autori di HuggingFace[3] consigliano di utilizzare una batch size di 16 o 32, un valore di learning rate tra 5e-5, 3e-5 o 2e-5 e come numero di epoche 2,3 o 4. La nostra scelta è stata quella di utilizzare un batch size di 32, un learning rate di 5e-5 e 4 epoche.

2 Dataset

Il dataset utilizzato [1], è costituito da un CSV di tweet in inglese(senza emoticons), collezionati grazie alle API di Twitter. Il CSV riporta 6 campi:

- la polarità del tweet (0=negativo,4=positivo)
- l'id del tweet(es. 2087)
- la data del tweet
- la query, se non c'è la query il valore è NO_QUERY
- nickname dell'utente che ha scritto il tweet
- il tweet

Il dataset comprende sia il training set, che è stato opportunamento splittato in Training set e Validation set, che il Test set.

2.1 Pulizia del dataset

Il dataset contiene 800000 esempi positivi e altrettanti esempi negativi. Per una maggiore chiarezza abbiamo scelto di sostituire le etichette(0=sentiment negativo, 1=sentiment positivo) e di cancellare le colonne non necessarie ai fini del task, lasciando solamente 'target'(che rappresenterà la label) e 'text'(che rappresenterà il tweet).

2.2 Sottocampionamento del dataset

Abbiamo valutato che fosse sufficiente utilizzare solamente 5000 esempi per il training, pertanto abbiamo sottocampionato il dataset a disposizione, mantenendo comunque un equilibrio tra esempi negativi e positivi.

2.3 Training set e validation set

A partire dal training set abbiamo ricavato il validation set utilizzando una proporzione 90:10, ovvero il 90% degli esempi come training set e il restante 10% come validation set.

3 Implementazione

3.1 GPU per il training

Per far sì che la computazione fosse più performante abbiamo scelto di impiegare una GPU Tesla K80 fornita da Colab.

3.2 Tokenizzazione e pulizia dell'input

Prima di eseguire la tokenizzazione e formattare l'input nel formato richiesto da BERT, abbiamo ritenuto necessario pulire l'input. Pertanto sono stati rimossi i tag(es. '@chrishasboobs') e gli spazi bianchi finali e corretti gli errori (es. '&' in '&').

Original: @chrishasboobs AHHH I HOPE YOUR OK!!!
Processed: AHHH I HOPE YOUR OK!!!

3.2.1 Tokenizzazione

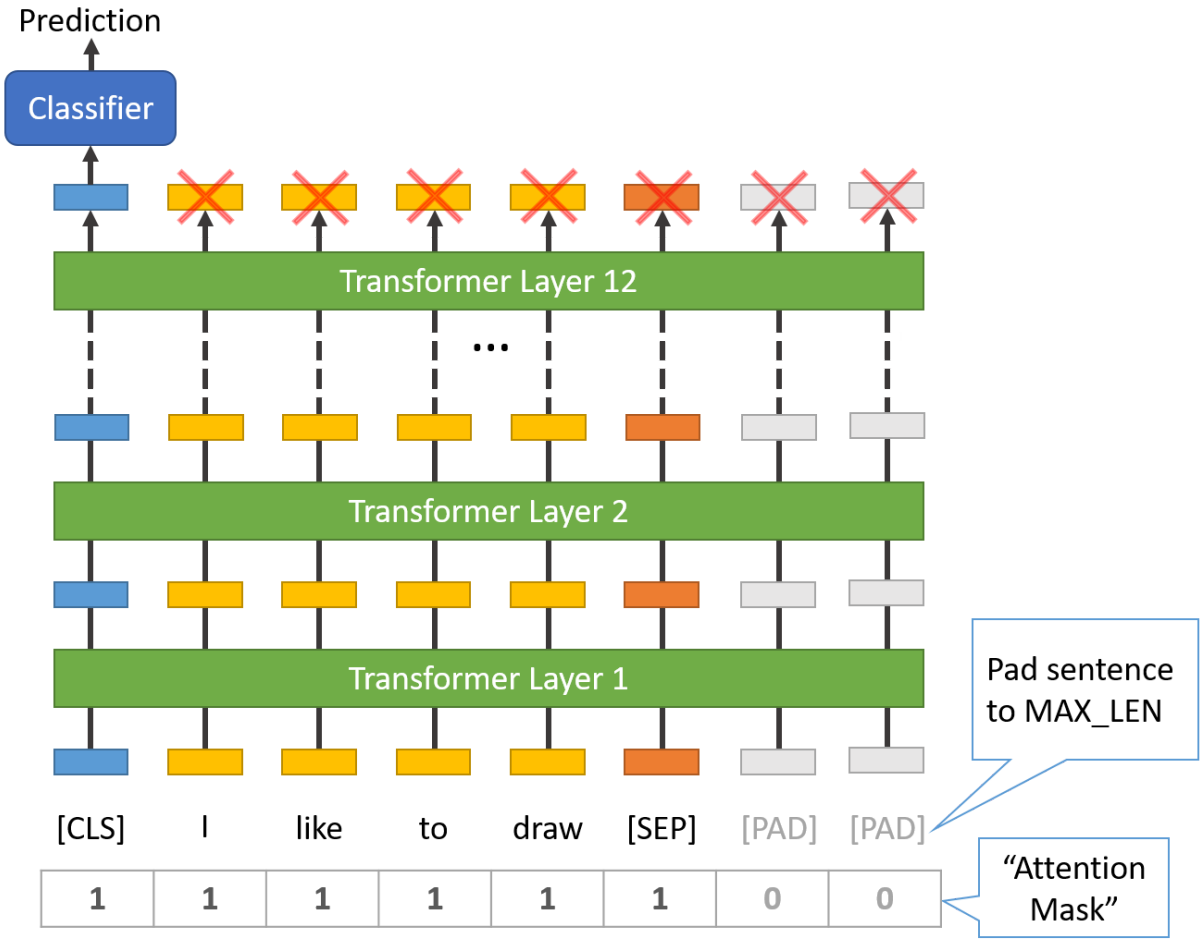
Per utilizzare BERT pre-addestrato è stato necessario impiegare la libreria BertTokenizer, per due motivi:

1) il modello ha un vocabolario specifico e fisso
2) BERT tokenizer ha un modo particolare di gestire le parole fuori dal vocabolario

Inoltre, prima di costruire gli embeddings che BERT riceverà, l'input è stato formattato aggiungendo dei token speciali all'inizio e alla fine di ogni frase.
Affinchè tutti i tweet fossero di uguale lunghezza è stata calcolata la lunghezza della massima sequenza (137) e conseguentemente tutti i tweet sono stati troncati o riempiti grazie al padding.
Il metodo *encode_plus* di BERT tokenizer:

1) splitta il testo in tokens
2) aggiunge i token speciali [CLS] e [SEP]
3) converte questi token in indici del vocabolario
4) riempie o tronca i tweet in base alla max_length
5) crea l'attention mask

Original tweet: @chrishasboobs AHHH I HOPE YOUR OK!!!
Token IDs: [101, 6289, 23644, 1045, 3246, 2115, 7929, 999, 999, 999, 102, 0]



3.2.2 Pytorch DataLoader

È stata impiegata la classe DataLoader di Pytorch per creare un iteratore sul set di dati. Ciò consente di risparmiare memoria e di aumentare la velocità di training.

3.3 BertClassifier

Il modello importato è bert-base-uncased, costituito da 12 transformers layers e addestrato su un corpus di documenti in inglese lower-cased. Ogni transformer layer prende in input degli embeddings di token e produce lo stesso numero di embeddings come output, forniti in input allo strato successivo.

La classe *BertClassifier* implementa il classificatore costruendo una rete neurale feed-forward fully connected con 768 neuroni di input, 50 di strato nascosto e 2 di output e la ReLu come funzione di attivazione.
La funzione *forward* permette di avviare Bert e memorizzare il risultato in *outputs* da cui viene estratto il token [CLS], che viene utilizzato come input del classificatore.

3.4 Ottimizzazione per fine-tune di BertClassifier

La scelta dell'ottimizzatore (Adam), del learning rate e degli altri iperparametri è basata su best-practices per simili task di classificazione. Gli autori di HuggingFace raccomandano i seguenti iperparametri:

- Batch size: 16 or 32
- Learning rate (Adam): 5e-5, 3e-5 or 2e-5
- Numero di epoche:2,3,4

Adam è un estensione dell'algoritmo stochastic gradient descent che ha una vasta applicazione del campo dell'NLP. Può essere utilizzato in alternativa a questo algoritmo di apprendimento per aggiornare i pesi della rete iterativamente.

3.5 Training ed Evaluation

BertClassifier è stato addestrato per 4 epoche e per ciascuna epoca il modello viene addestrato e ne vengono valutate le sue performance sul validation set.

La funzione di loss utilizzata è la Cross Entropy $H(p,q) = -\sum_{x} p(x) \cdot log(q(x))$ che serve per confrontare due generici vettori che rappresentano distribuzioni di probabilità, dove x si estende su tutti i valori potenziali della variabile causale su cui sono definite le probabilità, cioè le classi in output.

Nella fase di training come prima come prima cosa vengono decompressi i dati dal DataLoader.
Per ogni batch nel training data:

- i dati vengono caricati sulla GPU
- vengono azzerati i gradienti calcolati nel passaggio precedente
- viene calcolata la loss
- si calcola il gradiente con la funzione *backward*
- si cerca di prevenire l'esplosione del gradiente con il metodo *clip_grad_norm_* di Pytorch. Questo metodo cerca di ridimensionare i gradienti ogni volta che superano una determinata soglia. Ai gradienti viene impedito di esplodere, calcolandoli nuovamente in modo che la loro norma venga mantenuta a un valore inferiore o uguale alla soglia impostata
- aggiornamento parametri e learning rate (*optimizer.step()* e *scheduler.step()*)

Per la fase di evaluation, così come per il training, come prima cosa vengono decompressi i dati dal DataLoader e vengono caricati sulla GPU.

Per ogni batch nel validation set si calcolano la loss e le predizioni per valutare l'accuratezza del modello.

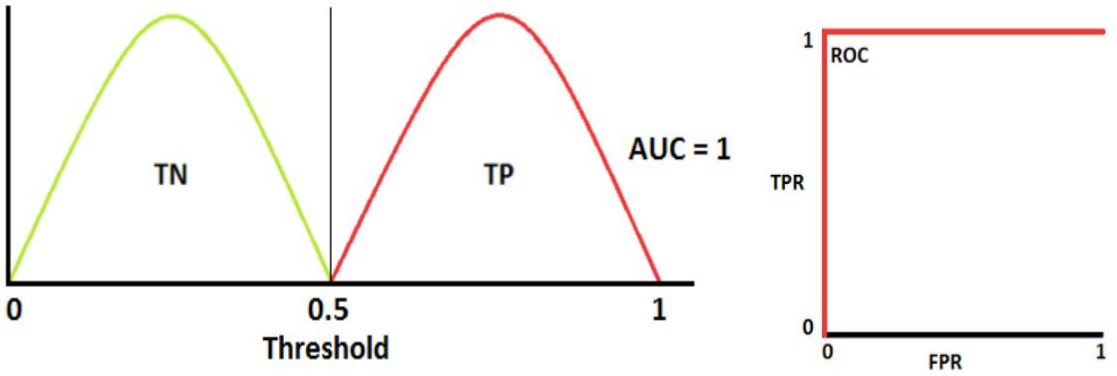
3.6 Predizione

Per calcolare la predizione abbiamo scelto di porre all'uscita del classificatore una funzione SoftMax, che consente di normalizzare l'output nell'intervallo [0,1] in modo da restituire un valore di probabilità.

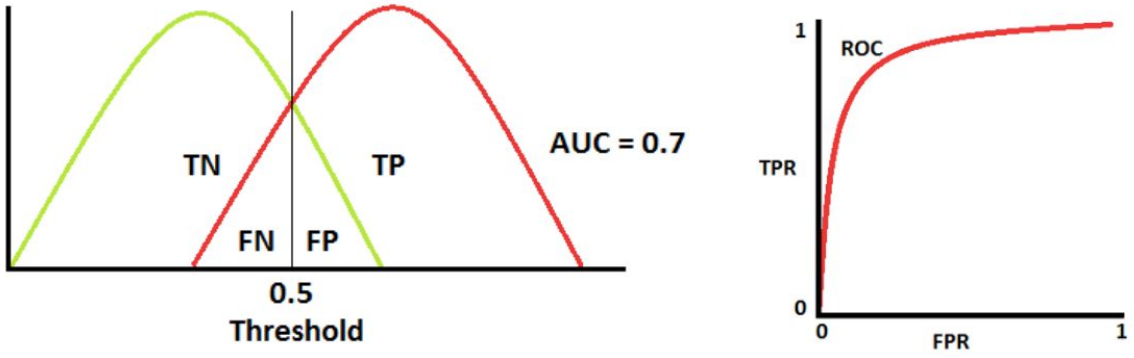
Per rappresentare l'accuratezza del modello abbiamo utilizzato una curva ROC. La curva ROC viene creata tracciando il valore del True Positive Rate (TPR, frazione di veri positivi) rispetto al False Positive Rate (FPR, frazione di falsi positivi) a varie impostazioni di soglia.

Attraverso l'analisi delle curve ROC è stato possibile analizzare la capacità del classificatore di discernere tra le due classi.

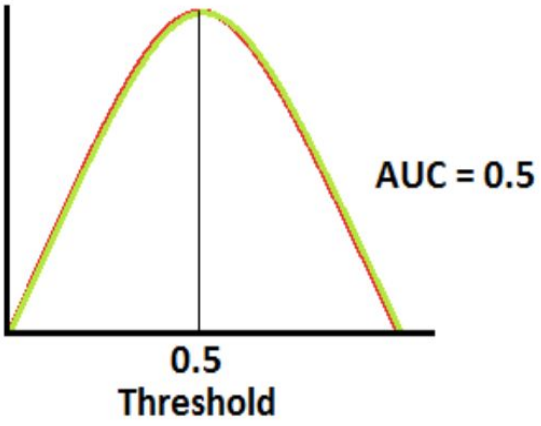
Un modello che riporta un AUC vicino a 1 ha una buona capacità di classificazione, mentre un modello con AUC prossimo allo 0 ha una pessima capacità di classificazione. Quando invece l'AUC è prossimo a 0,5 significa che il modello predice randomicamente le classi.

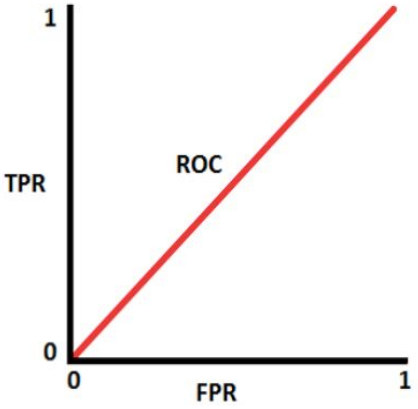


Questa è una situazione ideale in cui le due curve non si sovrappongono: significa che il modello ha una misura ideale di separazione. È perfettamente in grado di distinguere i campioni positivi da quelli negativi.

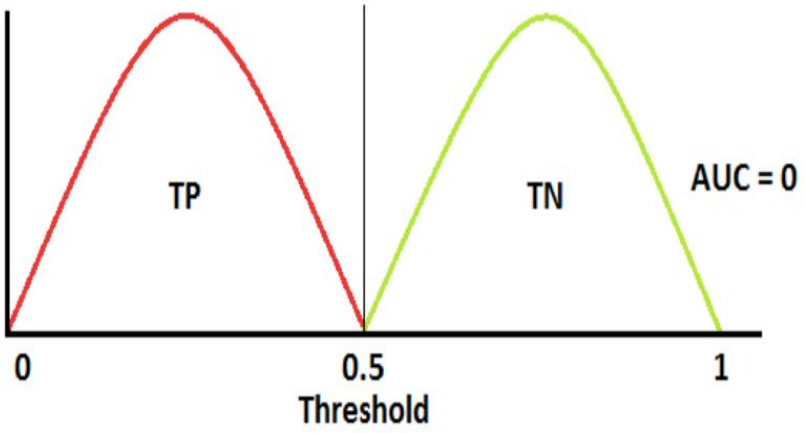


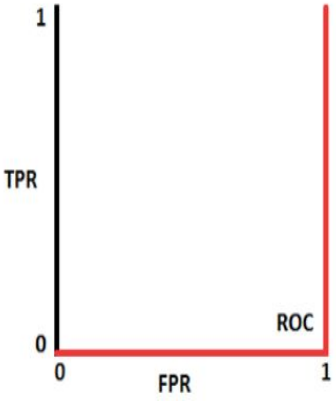
Quando le due distribuzioni si sovrappongono, vengono introdotti due tipi di errori. Un AUC uguale a 0,7 significa che il modello ha il 70% di probabilità di distinguere tra la classe positiva e la classe negativa.





Questo è il caso peggiore quando l'AUC è circa il 0,5, ciò significa che il classificatore non è in grado di distinguere tra le due classi. Si ha quindi un classificatore randomico.

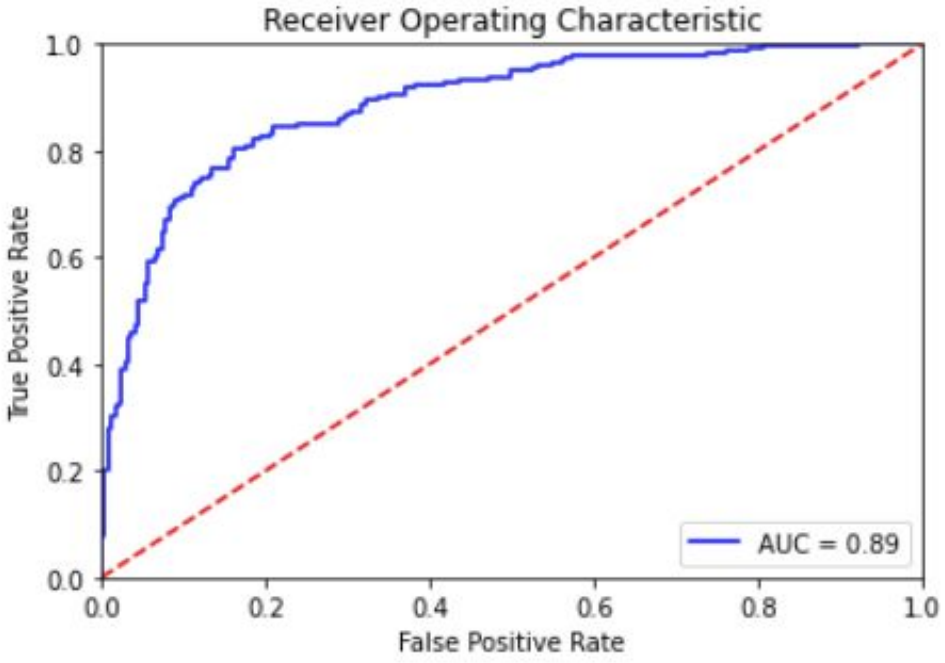




Quando l'AUC è vicino a 0 significa che il modello ha appreso i pattern corretti ma li sta utilizzando per fare previsioni capovolte. Significa quindi che il modello sta predicendo classi negative come positive e viceversa.

Il classificatore da noi implementato riporta un AUC di 0.8870, ciò vuol dire che il modello ha l'89% di probabilità di classificare correttamente.
Inoltre ha un'accuratezza dell'81,40%, come riportato nel grafico seguente.

```
AUC: 0.8870
Accuracy: 81.40%
```



3.6.1 Predizione sul test set

Affinchè possa essere effettuata la predizione sul test set, così come abbiamo fatto per il training e il validation set, l'input deve essere pre-processato attraverso il metodo *preprocessing*, che eseguirà anche la pulizia preliminare(metodo *text_clean*).

La soglia di classificazione è stata settata a 0.992, ciò garantisce che i tweet vengano classificati come positivi solamente quando riportano un valore di probabilità maggiore di tale soglia.

A titolo di esempio riportiamo 10 tweet che il nostro classificatore ha classificato come positivi e 10 classificati come negativi.

10 esempi di tweet classificati come positivi:
["reading Michael Palin book, The Python Years...great book. I also recommend Warren Buffet & Nelson Mandela's bio",
'RT @clashmore: http://bit.ly/SOYv7 Great article by Malcolm Gladwell.',
"@phyreman9 Google is always a good place to look. Should've mentioned I worked on the Mustang w/ my Dad, @KimbleT.",
'@johncmayer is Bobby Flay joining you?',
'RT @jquery: The Ultimate jQuery List - http://jquerylist.com/',
'Nike Air Yeezy Khaki/Pink Colorway Release - http://shar.es/bjfN',
'getting ready to test out some burger receipes this weekend. Bobby Flay has some great receipes to try. Thanks Bobby.',
'Any twitter to aprs apps yet?',

'just got back from the movies. went to see the new night at the museum with rachel. it was good',
'My dentist appt today was actually quite enjoyable.']

10 esempi di tweet classificati come negativi:
['Sony coupon code.. Expires soon.. http://www.coupondork.com/r/1796',
'Recovering from surgery..wishing @julesrenner was here :(',
"@ludajuice Lebron is a Beast, but I'm still cheering 4 the A..til the end.",
'@ambcharlesfield lol. Ah my skin is itchy :(damn lawnmowing.',
'is upset about the whole GM thing. life as i know it is so screwed up',
'Obama: Nationalization of GM to be short-term (AP) http://tinyurl.com/md347r',
'THE DENTIST LIED! " U WON\'T FEEL ANY DISCOMORT! PROB WON\'T EVEN NEED PAIN PILLS" MAN U TWIPPIN THIS SHIT HURT!! HOW MANY PILLS CAN I TAKE!!',
'annoying new trend on the internets: people picking apart michael lewis and malcolm gladwell. nobody wants to read that.',
'Naive Bayes using EM for Text Classification. Really Frustrating...',
"is Twitter's connections API broken? Some tweets didn't make it to Twitter..."]

4 Conclusioni

Il lavoro svolto ha portato a risultati soddisfacenti e ci ha permesso di sperimentare l'utilizzo di un'architettura transformer integrandola alle tradizionali tecniche e nozioni apprese durante i corsi.

Bibliografia

[1] http://help.sentiment140.com/for-students
[2] http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
[3] https://huggingface.co