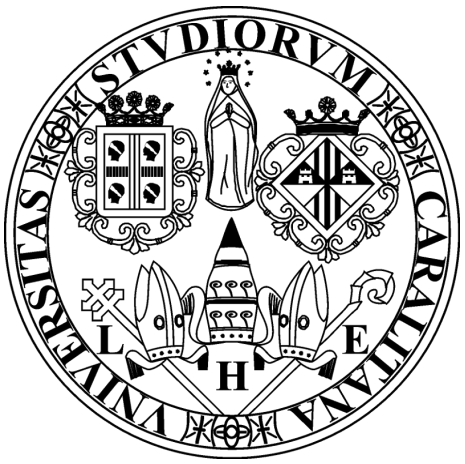


Relazione DLA

Deep Learning and Applications

Sentiment analysis on Amazon.com reviews

Giacomo Balloccu

Github: https://github.com/giacoballoccu/DLA-SentimentAnalysis

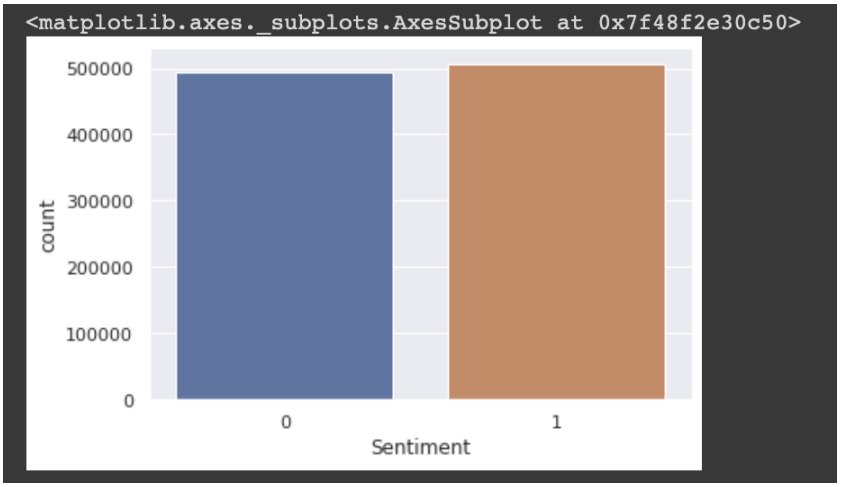
Indice degli argomenti

Introduzione

Lo scopo del progetto è stato quello di creare delle reti di tipo ricorrente per effettuare il sentiment analysis delle review di amazon. A partire da una recensione di un utente sotto forma di testo la rete si pone l'obiettivo di discriminare se il sentimento espresso dal testo sia positivo, ovvero all'utente è piaciuto l'item recensito, o non è piaciuto senza conoscere il rating reale che è stato dato nella recensione. Il training delle reti è stato effettuato usando l'intero dataset composto da circa 1 milione di recensioni di utenti ed è stato salvato per consentire il caricamento senza dover rieffettuare il train.
Nella prima parte della relazione si descriverà il dataset utilizzato, verrà spiegato e motivato il preprocessing effettuato. Successivamente si parlerà delle topologie delle reti utilizzate, si parlerà della fase di train e si giustificheranno le scelte effettuate, per poi parlare dei risultati ottenuti durante la fase di valutazione. Infine verranno tratte alcune conclusioni e idee per eventuali sviluppi futuri.

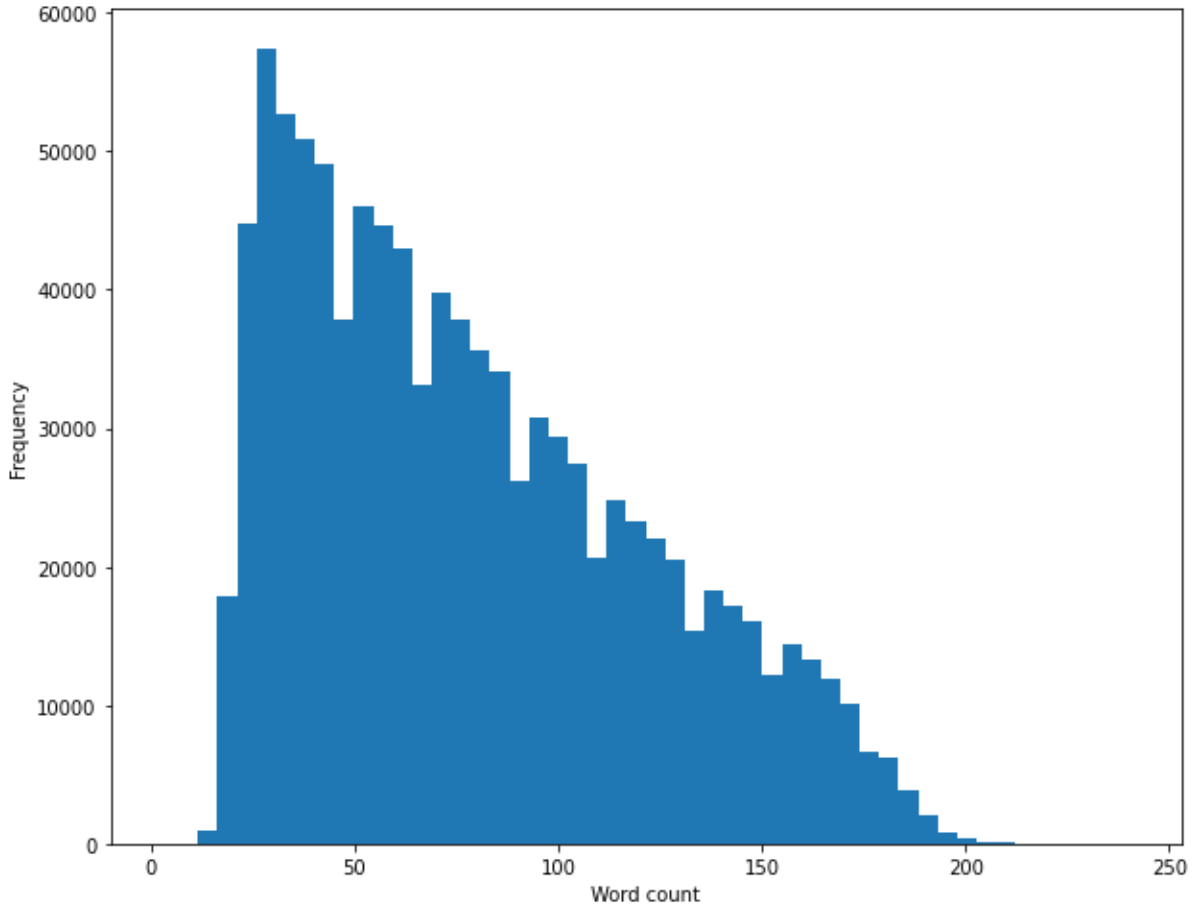
Dataset e problema

Il dataset utilizzato è stato "amazonreviews" di Kaggle disponibile al seguente link: https://www.kaggle.com/bittlingmayer/amazonreviews. Il dataset è composto da 1000000 di coppie tratte dalle review di vari prodotti amazon. Le coppie sono composte dal testo della recensione e dalla label di classe 0 o 1. La label 0 rappresenta le review a cui è stata assegnata 1 o 2 stelle mentre la label 1 rappresenta le recensioni con votazione 4 e 5. La composizione del dataset è stata visualizzata tramite il seguente plot:



Il numero di record appartenenti alla classe positiva è 505678 mentre quelli appartenenti alla classe negativa sono 494322. Questo mostra che il dataset per la sua composizione è bilanciato per quanto riguarda il numero di record appartenenti alla classe positiva e negativa e non necessita di ulteriori normalizzazioni.

Un'altra analisi che è stata effettuata sul dataset per capirne la sua composizione è stata quella di contare il numero di parole per ogni recensione e plottare il risultato:



Dal plot emerge che la maggior parte dei record presenti nel dataset hanno un numero di parole compreso tra 25 e 75, è inoltre possibile notare che tra le review in pochissime vanno oltre le 200 parole

Preprocessing

Essendo che le review sono scritte da persone il testo presenta del rumore che potrebbe interferire con la buona riuscita dell'addestramento della rete. E' stato dunque necessario rimuovere gli elementi del testo che non hanno nessuna informazione utile al fine di riconoscere il sentimento della recensione. Come per esempio URL, parti di HTML che potrebbero essere state catturate durante l'estrazione delle review e della punteggiatura. Queste rimozioni sono state effettuate utilizzando delle espressioni regolari ad hoc.

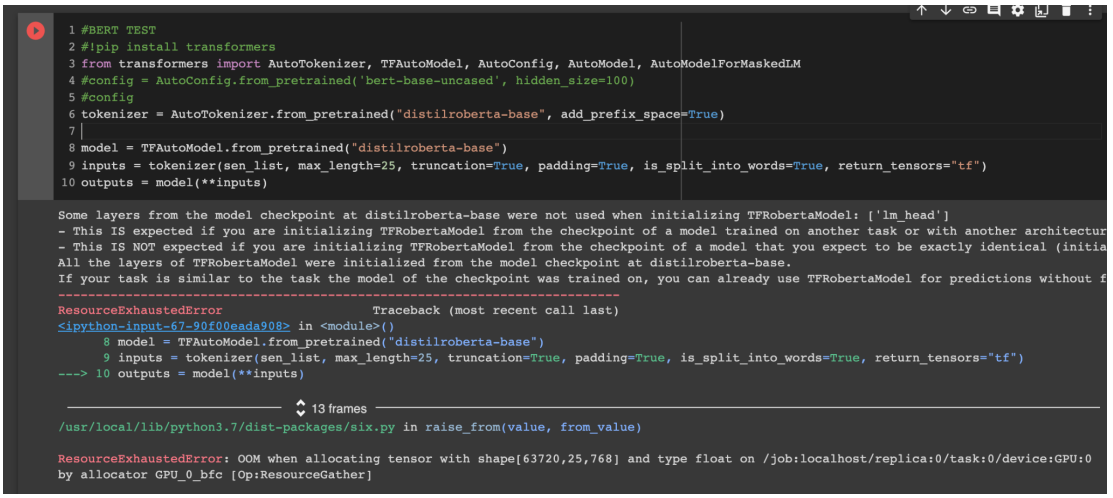
Un altro step di pre processing è stato quello di rimuovere le stopwords presenti nel testo, per far ciò è stata utilizzata la lista di english stopword presente nel pacchetto di feature extraction di sklearn e rimuovere da ogni testo le parole etichettate come stopwords.

Una volta effettuato questo preprocessing per effettuare l'handling delle OOV words e la trasformazione in word embeddings è stato utilizzato word2vec. Word2vec è una rete neurale progettata per elaborare il linguaggio naturale importabile tramite la libreria python gensim. La rete prende in ingresso un corpus e restituisce un insieme di vettori che rappresentano la distribuzione semantica delle parole nel testo. Per ogni parola contenuta nel corpus, in modo univoco, viene costruito un vettore in modo da rappresentarla come un punto nello spazio multidimensionale creato. In questo spazio le parole saranno più vicine se riconosciute come semanticamente più simili. Il testo delle review del training set è stato quindi trasformato grazie a word2vec in degli embeddings 3D che saranno poi gli input della rete.

Handling delle OOV

Le Out Of Vocabulary words (OOV) sono la maggior limitazione dell'uso degli embedding. Sono delle parole che non sono state viste nell'addestramento di modelli come Word2Vec o GloVe e quindi non hanno una rappresentazione sotto forma di embedding. Il modo più banale per la gestione è quello di assegnare un embedding con tutti valori zero alla parola ovvero ignorarla completamente nel momento dell'addestramento. Questa soluzione è particolarmente limitante dato che si stima che circa il 12% di tutte le parole dei testi del web siano un errore di battitura, dunque molte informazioni vengono perse utilizzando questa gestione.

La gestione ottimale sarebbe quella di utilizzare word embedding moderni come Bert o Roberta che sono in grado di ricostruire con una buona accuratezza le parole scritte in maniera errata e ricondurre alla parola originale utilizzando quindi il suo embedding. E' stato però impossibile effettuare dei tentativi usando Bert o Roberta che essendo dei modelli più complessi rispetto a word2vec mandavano il notebook di colaboratory PRO out of memory.



```
#BERT TEST
#!pip install transformers
from transformers import AutoTokenizer, TFAutoModel, AutoConfig, AutoModel, AutoModelForMaskedLM
#config = AutoConfig.from_pretrained('bert-base-uncased', hidden_size=100)
#config
tokenizer = AutoTokenizer.from_pretrained("distilroberta-base", add_prefix_space=True)

model = TFAutoModel.from_pretrained("distilroberta-base")
inputs = tokenizer(sen_list, max_length=25, truncation=True, padding=True, is_split_into_words=True, return_tensors="tf")
outputs = model(**inputs)
```

```
Some layers from the model checkpoint at distilroberta-base were not used when initializing TFRobertaModel: ['lm_head']
- This IS expected if you are initializing TFRobertaModel from the checkpoint of a model trained on another task or with another architectur
- This IS NOT expected if you are initializing TFRobertaModel from the checkpoint of a model that you expect to be exactly identical (initia
All the layers of TFRobertaModel were initialized from the model checkpoint at distilroberta-base.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFRobertaModel for predictions without f
---------------------------------------------------------------------------
ResourceExhaustedError                    Traceback (most recent call last)
<ipython-input-67-90f00eada908> in <module>()
      8 model = TFAutoModel.from_pretrained("distilroberta-base")
      9 inputs = tokenizer(sen_list, max_length=25, truncation=True, padding=True, is_split_into_words=True, return_tensors="tf")
---> 10 outputs = model(**inputs)

13 frames
/usr/local/lib/python3.7/dist-packages/six.py in raise_from(value, from_value)

ResourceExhaustedError: OOM when allocating tensor with shape[63720,25,768] and type float on /job:localhost/replica:0/task:0/device:GPU:0
by allocator GPU_0_bfc [Op:ResourceGather]
```

Una soluzione sarebbe quella di utilizzare versioni ridotte di Bert o Roberta ma purtroppo al momento la versione più ridotta di Bert o Roberta compatibile con tensorflow ha un valore di hidden_layers=256 che risulta ancora troppo alta per la ram allocata dal notebook.

Un compromesso allora è stato quello di utilizzare un word embedding più moderno e con una gestione migliore di word2vec ovvero FastText. Nonostante FastText non sia ai livelli di Bert o Roberta è comunque un buon compromesso per gli esperimenti effettuati.

```
from gensim.models import fasttext
ft_model = fasttext.FastText(sen_list,size=100)
```

```
[37] ft_model.wv.most_similar("beutiful")
```

```
[('beautiful', 0.9853143692016602),
 ('Beautifuln', 0.9836337566375732),
 ('beautifuln', 0.9824850559234619),
 ('Beautiful', 0.9787737131118774),
 ('beautifull', 0.9720891714096069),
 ('beautifullyn', 0.9621118307113647),
 ('beautifully', 0.9471719264984131),
 ('Beautifully', 0.9436919093132019),
 ('beatiful', 0.9338065385818481),
 ('Powerful', 0.9276737570762634)]
```

Addestramento e valutazione

Input size

Gli embeddings ottenuti tramite word2vec a sotto forma di vettori numpy hanno dimensione per l'addestramento delle reti hanno una dimensione di [nRecord x 25 x 100]. Dove 25 rappresenta il numero massimo di parole e 100 la lunghezza massima della finestra temporale. Il vocabolario ottenuto dal training set escludendo le stopwords ha una dimensione di 8805 parole.

Trainset, validation e test set

Lo split scelto è stato 80% dell'intero dataset per il trainset, 10% per la validazione durante l'addestramento e 10% per la fase di testing. Gli embedding del test set sono stati salvati per poter riprodurre l'esperimento ed evitare di ripetere le operazioni di preprocessing.

Baseline Models

Per avere una base di riferimento per l'accuratezza ottenuta con il dataset si sono addestrati due classificatori con dei metodi di machine learning utilizzando lo stesso preprocessing e split sul dataset per addestrare le reti.

SVM

L'SVM usato è quello presente nella libreria sklearn. E' stato addestrato con il meta parametro C=1 che esprime quanto il margine sia soft, il valore 1 è quello che mediamente si comporta meglio ed è quello che ha dato il miglior risultato. Il tempo di addestramento del SVM è stato il più lungo tra tutti i modelli impiegando 7 minuti e 15 secondi ottenendo un'accuratezza sul test set di 77,80%.

Random forest

Il random forest utilizzato è quello presente anch'esso nella libreria di sklearn. Il random forest è un metodo di machine learning di tipo ensemble ovvero combina le predizioni di diversi predittori (Decision Tree) in un'unica predizione tramite majority voting. La profondità massima dell'albero è stata impostata a 500 ed il criterio di split scelto è stato l'indice di GINI. Il modello ha impiegato 24.73 secondi per il suo addestramento (la migliore tra tutti i modelli) e un'accuratezza di 79,40% superiore a quella ottenuta dal modello SVM.

Recurrent Neural Networks

Per il confronto con i modelli di baseline si sono create ed addestrate due topologie di rete, entrambe le reti sono di tipo ricorrente bidirezionale (BDRNN).
Le reti neurali ricorrenti bidirezionali collegano due strati nascosti di direzioni opposte allo stesso output. Con questa forma, il livello di output può ottenere informazioni dagli stati passati (indietro) e futuri (avanti) simultaneamente. Questo tipo di reti sono note per la loro efficacia nel dominio del NLP per diversi problemi tra cui appunto la sentiment analysis.
La differenza principale fra le due topologie è stato l'utilizzo di layer differenti all'interno del layer bidirezionale.

Bidirectional con GRU layers

In questa versione il layer bidirezionale è composto da gated recurrent units (GRUs). Le GRU sono simili alle LSTM con forget gate, ma hanno meno parametri di un LSTM non avendo un output gate. Le performance delle GRU in certi task, tra cui l'NLP, risultano quasi equivalenti o talvolta (in certi dataset) superiore alle LSTM.

BDNN GRU

```
[51]  1 input_st  = Input(shape=(25,100))
      2 lstm1 = Bidirectional(GRU(200,input_shape=(25,100),activation='relu',return_sequences=True),merge_mode='mul')(input_st)
      3 lstm2 = Bidirectional(GRU(1,input_shape=(25,100),activation='relu',return_sequences=True),merge_mode='mul')(lstm1)
      4 print(lstm1.shape,' ',lstm2.shape)
      5 lstm2 = Reshape((-1,))(lstm2)
      6 lstm2 = Activation('sigmoid')(lstm2)
      7 lstm2 = Reshape((-1,1))(lstm2)
      8 mult = Multiply()([lstm1,lstm2])
      9
     10 add = Lambda(lambda x: K.sum(x,axis=1))(mult)
     11 dense = Dense(100,activation='relu')(add)
     12 output = Dense(1,activation='sigmoid')(dense)
     13
     14 model = Model(inputs=input_st, outputs=output)
     15 print(model.summary())
```

Nella rete a partire dall'input si susseguono due strati bidirectional dove le loro parti forward e backward vengono combinate tramite la moltiplicazione, definita su keras con merge_mod=mul. All'interno di ogni layer bidirezionali c'è un layer di tipo GRU quindi avendo 2 layer bidirezionali avremo due GRU con rispettivamente 200 e 1 units di output. Successivamente dopo alcuni aggiustamenti di dimensionalità si ha uno strato denso con 100 units con attivazione relu e uno strato denso con 1 unit per l'output label con attivazione sigmoidale.

Bidirectional con LSTM layers

Come precedentemente accennato le due reti differiscono nel tipo di layer scelti all'interno dei due layer bidirezionali. In questa topologia all'interno dei layer bidirezionali si sono utilizzati dei layer LSTM. Un layer LSTM è in grado di avere una "memoria" grazie a dei loop interni. Più nello specifico l'LSTM è composta da una cella, un input gate, un output gate e un forget gate, la cella ricorda i valori su un arbitrario intervallo di tempo e i tre gate servono per regolare il flusso di informazioni in entrate e in uscita della cella.

BDNN LSTM

```
[44]  1 input_st  = Input(shape=(25,100))
      2 lstm1 = Bidirectional(LSTM(200,input_shape=(25,100),activation='relu',return_sequences=True),merge_mode='mul')(input_st)
      3 lstm2 = Bidirectional(LSTM(1,input_shape=(25,100),activation='relu',return_sequences=True),merge_mode='mul')(lstm1)
      4 print(lstm1.shape,' ',lstm2.shape)
      5 lstm2 = Reshape((-1,))(lstm2)
      6 lstm2 = Activation('sigmoid')(lstm2)
      7 lstm2 = Reshape((-1,1))(lstm2)
      8 mult = Multiply()([lstm1,lstm2])
      9
     10 add = Lambda(lambda x: K.sum(x,axis=1))(mult)
     11 dense = Dense(100,activation='relu')(add)
     12 output = Dense(1,activation='sigmoid')(dense)
     13
     14 model = Model(inputs=input_st, outputs=output)
     15 print(model.summary())
```

Nel caso della seconda tipologia i layer bidirezionali hanno al loro interno dei layer di tipo LSTM con 200 e 1 units di output. Equivalentemente all'altra rete si hanno poi degli aggiustamenti per la shape in output dai layer bidirezionali e due strati densi.

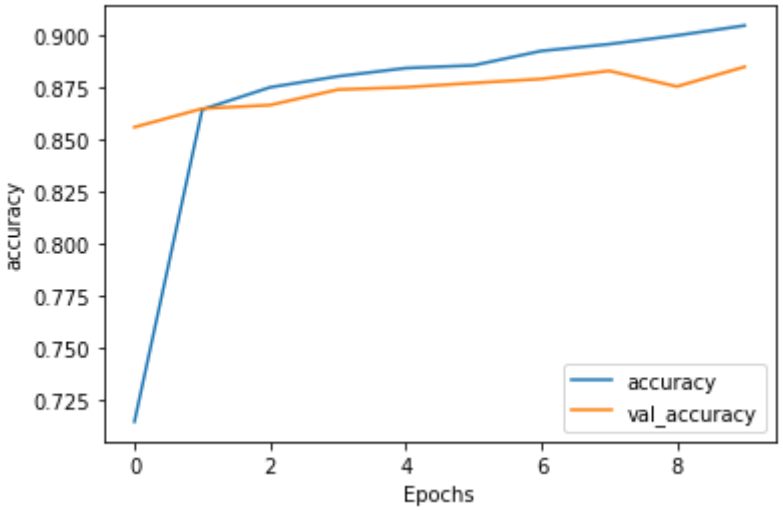
Metaparamentri, loss function e optimizer

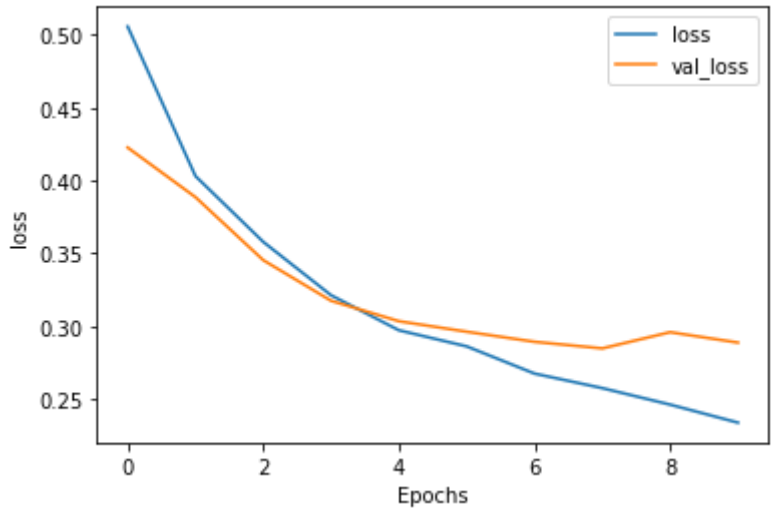
Le reti sono state entrambe addestrate con una batch size di 512 essendo un buon compromesso tra velocità di addestramento e accuratezza che otteneva un incremento di solo qualche centesimo con la riduzione della batch size.
La loss function usata essendo il problema di carattere binario è stata la binary cross entropy e l'optimizer adam per le sue affidabili prestazioni.
I modelli sono stati trainati per un totale di 10 epoche visto che un ulteriore prolungamento del train non portavo benefici ma incrementa l'overfitting. L'accuratezza del modello durante il train è stata monitorata usando un validation set della dimensione del 10% dell'intero dataset.

Analisi dei risultati

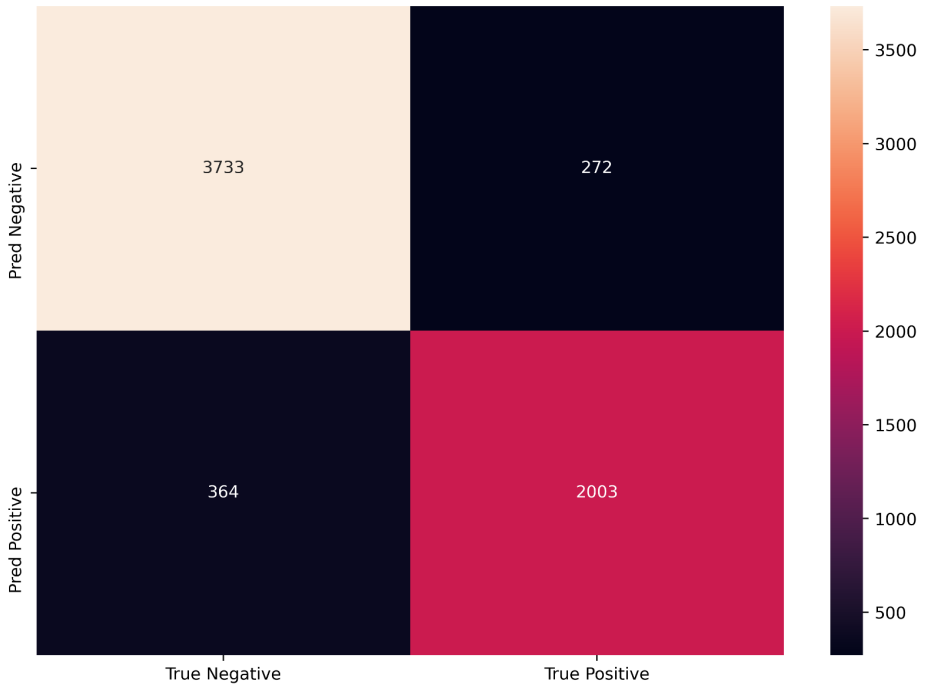
Per l'analisi dei risultati la metrica di riferimento è stata l'accuracy, le reti hanno avuto entrambe prestazioni simili affermandosi sul 90% di accuracy nella valutazione usando il test set split. Più precisamente l'accuracy per il BDGRU è stata di 89.28% mentre per BDLSTM è stata 89.17%. Le prestazioni della BDGRU sono leggermente inferiori dimostrando che le GRU nonostante siano più semplici in certi domini/problemi riescono a performare in maniera equivalente alle LSTM.
Successivamente sono riportati alcuni grafici e le matrici di confusione per i due modelli:

Bidirectional con GRU layers



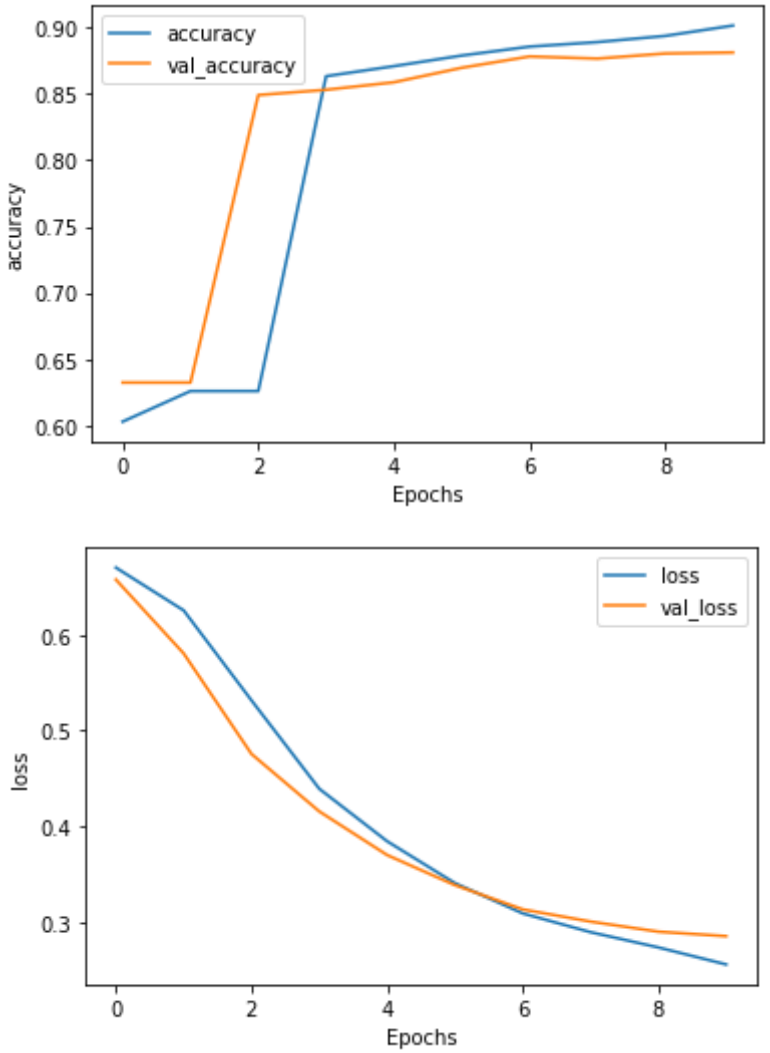


Come emerge dai plot la velocità con cui il modello viene addestrato è molto alta avendo un accuratezza del validation di circa 88% già nelle prime 2 epoche questo è probabilmente dovuto alla enorme quantità di dati in input che agevola l'addestramento. Come si può vedere da entrambi i grafici sia l'accuratezza del test che la sua loss function tendono a stabilizzarsi intorno alle 10 epoche.

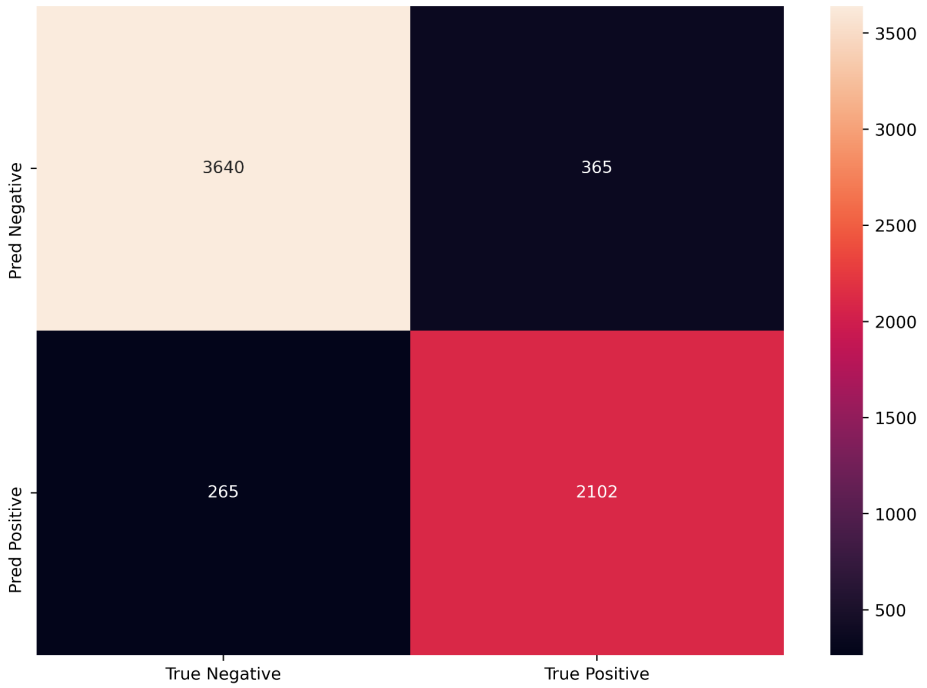


Come è possibile vedere dalla matrice di confusione questa rete favorisce i falsi positivi rispetto ai falsi negativi. Per la natura del task entrambi hanno lo stesso peso è un vero positivo non è più grave di un falso positivo.

Bidirectional con LSTM layers



Si possono trarre conclusioni equivalenti dai risultati ottenuti dalla BDLSTM visto che i modelli hanno avuto performance e comportamenti molto simili, come ci saremmo aspettati dalla letteratura.



A differenza del BDGRU la rete BDLSTM ha avuto più falsi positivi che falsi negativi. Nel caso i falsi positivi siano preferibili ai falsi negativi sarebbe preferibile utilizzare la BDLSTM rispetto alla BDGRU.

Considerazioni sui failure cases

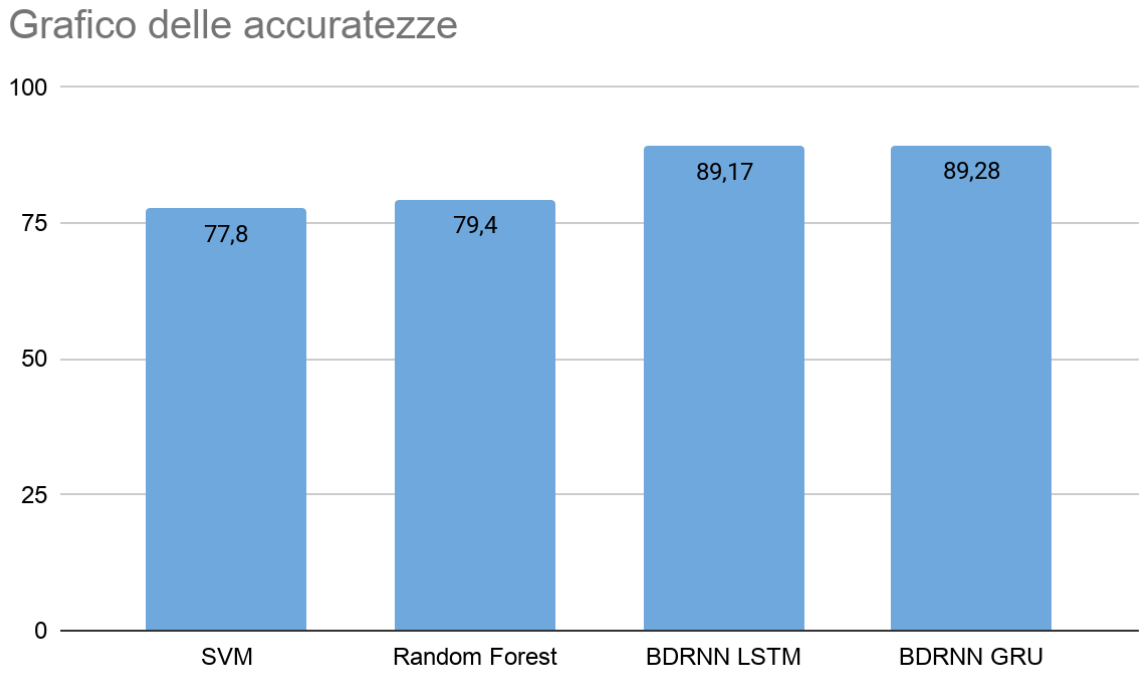
Nonostante i failure cases siano relativamente pochi è utile individuare le possibili cause per poter migliorare i risultati ottenuti ulteriormente. Le seguenti cause hanno probabilmente contribuito agli errori di predizione da parte dei due modelli.

- Mancata coerenza rating/review: Probabilmente nel dataset essendo review estratte dal mondo reale ci sono delle review che hanno un rating non coerente alla review scritta creando “dati sporchi” da cui il modello impara nella fase di addestramento.
- Recensioni confuse: Ci potrebbero essere recensioni che esprimono sentimenti confusi come per esempio “This item is so great, way better than the old version that I bought last year and it was so bad that was impossible to use it properly” dove è difficile per un sistema artificiale indurre il sentimento corretto.
- Recensioni che non esprimono nessun sentimento: Potrebbero esserci all’interno del dataset delle recensioni che non esprimono un reale sentimento verso un item ma piuttosto dei messaggi scorrelati completamente come per esempio un messaggio di spam.

Confronto con i modelli baseline

Come era prevedibile le reti hanno avuto performance nettamente superiori ai classificatori standard creati come baseline. Questo perchè le reti oltre che beneficiare maggiormente dalle dimensioni del dataset riescono a cogliere dei pattern temporali (grazie ai layer di tipo ricorrente) che i classificatori standard non riescono a cogliere. Ecco la tabella conclusiva dei risultati:

Modello	Accuratezza	Tempo di addestramento (s)
SVM	77,80%	429 s
Random Forest	79,40%	24 s
BDRNN LSTM	89.17%	157 s
BDRNN GRU	89.28%	200 s



Conclusioni

In conclusione si può affermare che le reti di tipo ricorrente in particolare le due testate hanno avuto risultati molto superiori rispetto ai classificatori statistici per via della loro maggiore espressività nell'interpretazione di pattern temporali. Per quanto riguarda le due reti si può dire che le GRU in certi task e con certi dataset hanno delle performance equivalenti alle LSTM nonostante siano più semplici per costruzione. Gli esperimenti hanno inoltre dimostrato l'efficacia di un buon preprocessing quando si lavora con il testo o sequenze temporali.
Come sviluppi futuri sarebbe interessante provare ad utilizzare gli stessi modelli ma utilizzando degli embedding con una maggiore gestione delle OOV ovvero creati con reti complesse quali Bert o Roberta e confrontare i risultati con le reti prodotte da questo lavoro.