Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Mercari Price Suggestion Challenge

Authors:
Gabriele Ferrario - 817518 - g.ferrario@campus.unimib.it
Riccardo Pozzi - 807857 - r.pozzi@campus.unimib.it

22 gennaio 2021


UNIVERSITA' DEGLI STUDI DI MILANO
BICOCCA

Sommario

Il progetto si incentra su di un problema di regressione derivato dalla Kaggle Challenge Mercari Price Suggestion che consiste nella predizione del prezzo di vendita dei prodotti a partire da feature testuali e non. È stata effettuata una valutazione sia in termini di performance di regressione che di costi computazionali di diversi approcci al problema, soprattutto relativi alla rappresentazione del testo, tra cui Bag of Words e Word Embedding. È stato valutato il contributo del pre-processing del testo e l'efficacia di diversi tipi di layer neurali nella gestione del testo. In conclusione sono state ottenute ottime performance utilizzando un Word Embedding allenabile tramite il layer Embedding di Keras in combinazione a layer LSTM bidirezionali. Bag of Words è risultata una buona alternativa pur essendo più semplice e computazionalmente più economica. Per quanto riguarda il pre-processing del testo, l'approccio condotto non ha portato miglioramenti rilevanti.

1 Introduction

Il progetto trae origine dalla *Kaggle challenge* **Mercari Price Suggestion Challenge** [1] aperta a fine Novembre 2017 che come viene reso chiaro dal sottotitolo *"Can you automatically suggest product prices to online sellers?"* pone l'obiettivo di stimare il prezzo dei prodotti a partire da alcune loro caratteristiche.
Alla base di ciò vi è l'esigenza dell'e-commerce *Mercari* [2] di offrire ai propri venditori un suggerimento sul prezzo di vendita dei prodotti inseriti.

Si tratta quindi di un problema di *regressione* che a partire dalle varie caratteristiche dei prodotti, testuali e non, vuole calcolare il prezzo da suggerire.
Durante lo svolgimento del progetto si valuteranno vari approcci al problema, sopratutto per quanto riguarda i dati di tipo testuale, sia in termini di errore rispetto ai dati di *train* che di costi computazionali.

In particolare saranno valutati i seguenti modelli di rappresentazione del testo: Bag of Words, Tf-Idf, Word Embedding, Word Embedding pre-allenato (GloVe), Feature extraction con Transfomer pre-allenato (DistilBert); verrà inoltre valutata l'efficacia del pre-processing del testo.

2 Datasets

I dati sono reperibili direttamente dalla pagina web della challenge; sono presenti un dataset di train e uno di test. Per la valutazione è stato utilizzato esclusivamente il train in quanto il test è pensato per la challenge e non contenendo i prezzi target risulta inutile per la valutazione; ogni riferimento successivo al dataset è quindi da intendersi al dataset di train.

Il numero di prodotti contenuti è di circa 1.39 milioni e per ognuno di essi sono fornite le seguenti caratteristiche.

- **Price:** rappresenta il prezzo di vendita dell'articolo, ovvero la variabile target della regressione. Sono stati considerato i prodotti con prezzo tra \$5 e \$2000 in base ai vincoli definiti dall'e-commerce [3]; la maggior parte dei prodotti (il 75%) ha un prezzo inferiore ai \$29.
- **Train_id:** rappresenta l'identificativo del prodotto nell'elenco.
- **Name:** è il nome del prodotto sotto forma di dato non strutturato.
- **Shipping:** rappresenta chi, tra venditore e acquirente, si fa carico delle spese di spedizione: 1 per "il venditore", 0 per l'acquirente. Inaspettatamente il prezzo dei prodotti spediti a carico degli acquirenti è mediamente superiore.
- **Item_condition_id:** rappresenta lo stato del prodotto; questo valore varia da 1 a 5. Il valore più frequente è 1, mentre 4 e 5 sono i più rari. La Kaggle challenge non ne fornisce una descrizione dettagliata del significato.
- **Category_name:** rappresenta la categoria di prodotto a cui appartiene l'articolo. Circa 6000 articoli non hanno nessuna categoria assegnata, mentre i restanti si suddividono in 1287 categorie, ad esempio *"Women/Tops & Blouses/T-Shirts"*, dove è facilmente osservabile una gerarchia di sottocategorie dalla più generica alla più specifica. Il campo *Category_name* è stato diviso in 3 campi corrispondenti alle 3 sottocategorie principali assegnando la stringa "NA" a quegli articoli con uno o più livelli mancanti.
- **Brand_name:** rappresenta il marchio dell'articolo; per più di 600 mila prodotti, ovvero poco meno della metà del totale, il campo risulta mancante: a questi prodotti è stato assegnato "NA" come *Brand_name*.

- **Item_description:** rappresenta la descrizione del prodotto sotto forma di dato non strutturato. Nel dataset sono presenti 4 istanze senza descrizione e circa 82 mila con la stringa "no description yet"; in entrambi i casi *item_description* è stato impostato a "NA".

  Nelle wordcloud ottenute dai bigrammi delle descrizioni (figure 1 e 2) si nota che i prodotti con prezzo maggiore o uguale a $100 (Figura 1) sono spesso descritti con informazioni che ne mostrano le buone condizioni, ad esempio: "100 authentic", "great condition" e "good condition". Al diminuire del prezzo queste coppie di parole diventano meno frequenti; aumentano invece quelle relative a descrizioni mancanti.

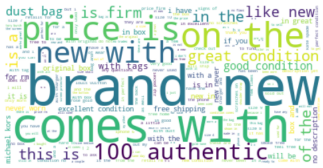

Figura 1: Descrizioni dei prodotti con prezzo maggiore o uguale a $100

Figura 2: Descrizioni dei prodotti con prezzo minore o uguale a $30

2.0.1 Preprocessing del testo

Il pre-processing del testo è stato effettuato tramite le seguenti operazioni:

1. conversione in caratteri minuscoli,
2. lemmatizzazione,
3. rimozione della punteggiatura,
4. rimozione delle *stopwords*,
5. rimozione delle parole di un solo carattere fatta eccezione per i numeri la cui rimozione spesso complica la comprensione della descrizione,
6. rimozione delle emoji.

In seguito si farà riferimento sia al pre-processing completo che al pre-processing limitato, ovvero composto dalle sole operazioni "conversione in minuscolo" e "rimozione di punteggiatura".

2.0.2 Encoding

Per quanto riguarda le variabili categoriche, ovvero *Brand_name* e le sotto-categorie di *Category_name*, i valori sono stati codificati in interi tramite *label encoding*.

I campi testuali invece sono stati codificati in forma vettoriale utilizzando vari approcci che verranno in seguito discussi.

3 The Methodological Approach

Inizialmente sono state valutate le performance a partire dalle sole features non testuali così da stabilire un punto di partenza per la successiva analisi di quelle testuali.

È stato utilizzato un semplice modello composto da due layer Densi entrambi seguiti da un layer di Dropout con un layer Denso finale con funzione d'attivazione lineare. Questo modello verrà in seguito ampliato per fare uso delle feature testuali a seconda dei vari approcci. La figura 3 ne mostra un template.

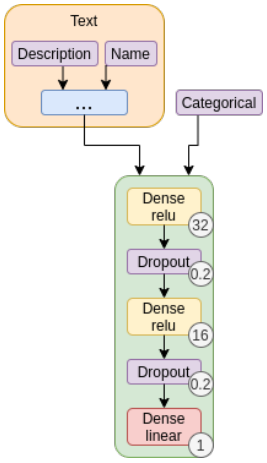

Figura 3: Template del modello con numero di neuroni e rate di dropout finali cerchiati

3.1 Rappresentazione del Testo

Sono stati valutati diversi approcci per il trattamento del testo partendo da modelli semplici come *Bag of Words* fino ai più recenti ed efficaci come *BERT*, passando per i *Word Embedding*.

3.1.1 Bag of Words

Il modello Bag of Words permette di rappresentare il testo tramite un vettore per ogni documento contenente il numero di occorrenze di ciascuna parola del vocabolario nel documento; questi vettori hanno cardinalità pari alla dimensione del vocabolario e assegnano 0 in caso di parole assenti dal documento; sono generalmente sparsi.

Regole grammaticali e ordine delle parole non sono rappresentabili tramite questa tecnica [4].

Ritornando al modello, i vettori delle feature testuali sono forniti in input direttamente al primo layer denso allo stesso modo delle variabili non testuali.

3.1.2 Tf-Idf

Tf-Idf (Term frequency-Inverse document frequency) [4] è una misura dell'importanza di una parola rispetto al documento che la contiene in rapporto all'importanza della stessa nell'intera collezione di documenti; segue la definizione.

$$Tf\text{-}Idf_{t,d} = tf_{t,d} \cdot idf_t = \frac{n_{t,d}}{|d|} \cdot \log \frac{|D|}{|\{d : t \in d\}|} \tag{1}$$

Dove t indica il termine, d il documento, $n_{t,d}$ la frequenza assoluta di t in d, D l'insieme dei documenti.

Per Tf-Idf come approccio di rappresentazione del testo si vuole intendere un approccio simile a Bag of words i cui vettori, anziché contenere il numero di occorrenze contengono il valore Tf-Idf (1) della parola.

All'interno del modello è stato utilizzato allo stesso modo di Bag of Words.

3.1.3 Word Embedding

I cosiddetti *Word Embedding* rappresentano parole per mezzo di vettori densi di lunghezza fissa [5], contrapponendosi a *Bag of Words*.

Inoltre questi vettori sono in grado di apprendere relazioni semantiche e sintattiche tra le parole [6].

3.1.4 Keras Embedding Layer

Keras fornisce un implementazione di embedding sotto forma di layer neurale e ne rende quindi possibile l'addestramento insieme ai layer restanti.
Ritornando al modello, il testo in input è stato codificato in forma di vettori di interi, tramite la classe *tf.keras.pre-processing.text.Tokenizer*.

Tenendo come riferimento il template in figure 3, nella sezione che tratta il testo, precisamente in sostituzione al box "..." sono stati aggiunti due layer embedding, uno per *Name* e uno per *Item_description*, seguiti da layer aggiuntivi per sfruttare al meglio le informazioni degli embedding di cui verrà discusso in seguito.

Infine l'output di questi layer aggiuntivi è stato assegnato come input al primo layer denso. La figura 4 ne mostra uno schema.

Quando in seguito si leggerà del modello Keras si intenderà questo approccio.

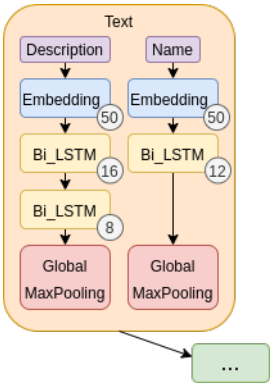

Figura 4: Schema riassuntivo sulla gestione dei dati testuali, dove per Bi_LSTM si intendono layer LSTM bidirezionali con dropout a 0.2 e i valori cerchiati identificano il numero di neuroni.

3.1.5 GloVe

GloVe (Global Vectors) è un modello non supervisionato in grado di costruire Word Embedding a partire da statistiche globali (rispetto all'intero corpus)

di co-occorrenza delle varie parole [7]. Sono inoltre disponibili modelli GloVe pre-addestrati. Seguono quelli valutati nel progetto.

- GloVe6B (Wikipedia 2014 + Gigaword 5): con vettori di 50 dimensioni allenati su 6 miliardi di parole e con un vocabolario di 400 mila parole.
- GloVe840B (Common Crawl): con vettori di 300 dimensioni allenato su 840 miliardi di parole e con un vocabolario di 2.2 milioni parole.

Per quanto riguarda il modello è stata utilizzata la stessa architettura del modello Keras assegnando i pesi pre-addestrati e la corretta dimensionalità ai layer Embedding e impostandoli come non-trainabili.

3.1.6 Transformers

Il Transformer è un'architettura basata su di una struttura *encoder-decoder* che utilizza un meccanismo di *attention* per rappresentare i rapporti di dipendenza di input e output [8].

Una dei più noti transformer è *BERT*, *Bidirectional Encoder Representations from Transformers*), dove per bidirezionale si intende capace di apprendere relazioni sia con input precedenti che successivi, che si basa esclusivamente su moduli encoder. Nasce per fornire un modello pre-allenato adattabile a numerose applicazioni tramite *fine-tuning*. Tuttavia anche utilizzi *feature-based* risultano efficaci [9].

In questo progetto è stato utilizzato *DistilBert*, una versione più leggera ottenuta da BERT tramite *Knowledge distillation* [10], con un approccio feature-based.

Ritornando al modello, occupa la stessa posizione dedicata agli embedding (figura 4) per entrambi i campi nome e descrizione.

3.2 Training e valutazione

3.2.1 Dataset split

Al fine di valutare i vari modelli più correttamente possibile il dataset è stato suddiviso in *training-set*, *validation-set* e *test-set*.

3.2.2 Valutazione dell'errore

Per quanto riguarda le performance in termini di errore la funzione di *loss* utilizzata in fase di training è il *Root Mean Squared Logaritmic Error (RM-*

SLE); è inoltre la misura scelta dalla Kaggle challenge per confrontare le performance dei vari partecipanti.

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\log^2(\frac{y_i + 1}{\hat{y}_i + 1})} \quad (2)$$

Essendo calcolato a partire da un rapporto, riflette l'errore relativo e di conseguenza risulta efficace laddove i valore assoluti presentano variazioni considerevoli, come per i prezzi in esame.

3.2.3 Valutazione dei costi computazionali

Per valutare i costi computazionali sono stati annotati il numero dei parametri allenabili e non dei vari modelli e il tempo richiesto dalle epoche di allenamento sul training set.

È stata utilizzata la piattaforma *Google Colab* abilitando l'accelerazione hardware GPU.

3.3 Parametri di training

I parametri di training sono stati impostati in seguito a test sperimentali in modo da minimizzare l'errore mantenendo accettabili i tempi richiesti.

4 Results and Evaluation

Tabella 1: Per ogni approccio sono riportati il numero di parametri totali e allenabili, la durata di una singola epoca, il numero di epoche eseguite e l'Rmsle sul test. Il suffisso *_clean* indica l'applicazione pre-processing completo in opposizione a quello limitato.

Modello	TotalParams	TrainParams	EpochTime	NumEpochs	Rmsle
BoW	6.678.049	6.678.049	250s	20	0.4486
BoW_clean	6.225.953	6.225.953	250s	19	0.4510
TfIdf	6.678.049	6.678.049	250s	20	0.4576
TfIdf_clean	6.225.953	6.225.953	250s	20	0.4540
GloVe6B	119.191.497	75.297	117s	39	0.4725
GloVe6B_clean	90.674.697	75.297	118s	53	0.4703
GloVe840B	119.191.497	75.297	135s	40	0.4607
GloVe840B_clean	90.674.697	75.297	121s	40	0.4633
Bert	66.543.009	180.129	1600s	5	0.5576
NonText	769	769	14s	40	0.6564
Keras	19.871.997	19.871.997	790s	7	0.4460
Keras_clean	15.119.197	15.119.197	549s	9	0.4525

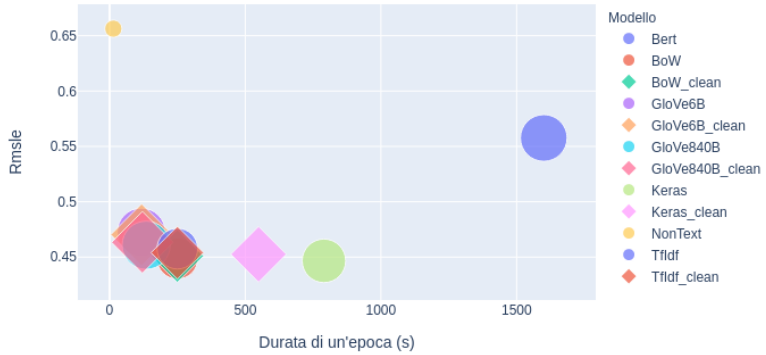

Figura 5: Rmsle sul test in funzione della Durata di un epoca di training. La dimensione corrisponde ai Parametri Allenabili. I risultati utilizzanti il pre-processing completo sono indicati da rombi.

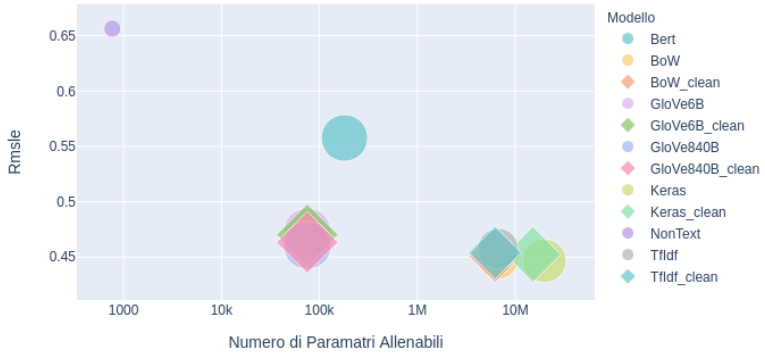

Figura 6: Rmsle sul test in funzione del Numeri di Parametri Allenabili. La dimensione corrisponde ai Parametri Totali. I risultati utilizzanti il preprocessing completo sono indicati da rombi.

5 Discussion

5.1 Modello Finale

5.1.1 Parametri di training

Numero di neuroni: Il numero di neuroni del modello finale dei vari layer è mostrato nelle figure 3 e 4 allo stesso modo del rate di dropout.

Optimizer: Come optimizer è stato utilizzato Adam con learning rate a 1e-4, in modo da migliorare la stabilità del training ad eccezione dell'Embedding Keras che ha dimostrato un comportamento migliore con il learning rate di default (1e-3).

Batchsize: La batchsize è stata impostata a 256.

Regolarizzazione: Oltre ai layer Dropout è stato testata la regolarizzazione L2, tuttavia non ha portato a evidenti miglioramenti ed è quindi stata omessa. Inoltre è stato impostato l'earlystopping a monitoraggio dell'errore sul validation impostando pazienza a 3 epoche e il ripristino automatico dei pesi più performanti.

5.1.2 Layer aggiuntivi per il testo

Per quanto concerne i layer aggiuntivi in coda agli Embedding sono state valutati layer ricorsivi (RNN) in quanto ottimi per i dati sequenziali, come il testo, essendo in grado di "memorizzare" informazioni sugli elementi precedenti [11]. Nello specifico sono stati valutati LSTM, GRU e Bidirectional LSTM. Queste ultime, permettono di catturare relazioni sia con parole precedenti che con parole successive [12], ed essendosi dimostrate le più performanti compaiono nel modello finale.

Sono stati inoltre valutati layer convoluzionali a precedere i layer RNN, ma non avendo migliorato le performance sono stati omessi nel modello finale.

L'aggiunta di un layer GlobalMaxPooling1D a seguito delle Bi_LSTM (figura 4) si è dimostrata efficace sia per rendere la dimensionalità compatibile con i successivi layer densi che nell'effettuare un *downsampling* riducendo dimensionalità e costi computazionali.

5.1.3 Dimensione Word Embedding

La dimensione del Word Embedding nel modello Keras è stata impostata a 50.

5.2 Pulizia del testo

Generalmente il pre-processing completo del testo non ha prodotto evidenti miglioramenti in termini di errore rispetto al pre-processing limitato, nonostante nel caso degli embedding pre-trainati permetta una maggiore copertura delle parole del vocabolario: da 32% a 38% per GloVe6B e da 42% a 52% per GloVe840B.

Per quanto riguarda gli embedding ciò può essere dovuto al fatto che essendo in grado di catturare relazioni semantiche tra le parole non necessitano di un alto grado di pre-processing, ma siano anzi in grado di sfruttare le stopwords ad esempio.

È invece inaspettato il risultato ottenuto combinando pre-processing totale a Bag of Word: l'errore è infatti leggermente più alto che con pre-processing limitato. Si deduce che le operazioni come rimozione di stopwords e lemmatizzazione non hanno offerto un contributo rilevante ma anzi un leggero peggioramento.

Si nota invece una generale diminuzione del numero di parametri (figura 6) che in alcuni casi si traduce in una riduzione del tempo richiesto dalle epoche di train (figura 5).

5.3 Performance

Il modello più performante dal punto di vista dell'errore risulta il modello Keras, evidenziato in verde nella tabella 1. Seguono gli approcci Bag of Words e Tf-Idf; quest'ultimo tuttavia non sembra migliorare i risultati del più semplice BoW.

Dal punto di vista computazionale invece, la durata delle epoche di Bag of Words è notevolmente minore rispetto a quella del modello Keras, ma considerando il numero di epoche eseguite ci accorgiamo che il tempo totale impiegato è simile: 5000s per BoW e 5530s per Keras. È necessario ricordare che Keras è stato allenato con un learning rate maggiore che per BoW risultava invece in una scarsa stabilità.

Sono invece gli Embedding GloVe pre-trainati a richiedere il minor tempo in una singola epoca di training (ignorando il modello senza feature testuali), avendo il numero minore di parametri allenabili come mostrato in figura 6. Anch'esse tuttavia hanno richiesto numerose epoche per raggiungere performance accettabili. Inoltre non risultano esserci notevoli differenze di errore o costi computazionali tra i due embedding pre-trainati GloVe6B e GloVe840B.

5.4 BERT

Non è stato computazionalmente possibile questo modello sull'intero dataset ma ne è stata utilizzata una porzione (circa 200 mila istanze) completando solo 5 epoche. Tuttavia considerando queste limitazioni i risultati sembrano promettenti.

6 Conclusions

Riassumendo dai risultati ottenuti si evince che il modello Keras permette di ottenere ottime performance in cambio di costi computazionali accettabili, perciò questa tecnica risulta consigliata nel caso in cui si abbia a disposizione una piattaforma almeno comparabile con Google Colab.

Bag of Words è stata una piacevole sorpresa infatti pur essendo l'approccio più semplice testato ottiene ottimi risultati ed è probabilmente attuabile anche da chi ha particolari limitazioni computazionali.

È stato inoltre evidenziato che il pre-processing del testo effettuato generalmente non giova alle performance e sarebbe interessante valutare approcci alternativi a quello utilizzato nel progetto.

Nel caso si abbiano a disposizione grandi risorse computazionali sarebbe utile valutare approfonditamente l'approccio basato su BERT che potrebbe portare a performance ottimali.

Un altro approccio che varrebbe la pena testare sarebbe l'utilizzo di un embedding pre-trainato con GloVe come punto di partenza per poi allenarne ulteriormente i pesi magari ampliando le dimensioni dell'embedding in caso si abbiano risorse sufficienti.

Riferimenti bibliografici

[1] Kaggle, "Mercari price suggestion challenge," https://www.kaggle.com/c/mercari-price-suggestion-challenge, 2017.

[2] Mercari, "Mercari," https://www.mercari.com.

[3] ——, "How to set a price," https://www.mercari.com/us/help_center/article/69.

[4] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[5] F. Almeida and G. Xexéo, "Word embeddings: A survey," 2019.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[7] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[10] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[11] H. Liang, X. Sun, Y. Sun, and Y. Gao, "Text feature extraction based on deep learning: a review," *EURASIP journal on wireless communications and networking*, vol. 2017, no. 1, pp. 1–12, 2017.

[12] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.