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

The ABSTRACT is not a part of the body of the report itself. Rather, the abstract is a brief summary of the report contents that is often separately circulated so potential readers can decide whether to read the report. The abstract should very concisely summarize the whole report: why it was written, what was discovered or developed, and what is claimed to be the significance of the effort. The abstract does not include figures or tables, and only the most significant numerical values or results should be given.

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 svoglimento del progetto si valuteranno vari approcci al problema, soprattuto 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 Word, Tf-Idf, Word Embedding, Word Embedding pre-allenato (GloVe), Feature extraction con Transfomer pre-allenato (DistilBert); e per ognuno di essi è stato valutata l'utilità o meno della pulizia del testo.

2 Datasets

Il 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 esclusivamente per la challenge e non contenendo i prezzi target risulta inutile per la valutazione.

Il dataset di train rappresenta quindi il dataset preso in esame e ogni riferimento successivo al dataset è 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. Il suo valore medio è di circa $26, con un minimo pari a $0, un massimo di $2009 e una deviazione standard di circa $38. Analizzando meglio la distribuzione si nota che in generale i prezzi sono relativamente bassi: inferiori a $29 per il 75% dei prodotti.

Dopo aver appreso dal sito ufficiale di Mercari che i prezzi sono impostabili tra $5 e $2000 [**?**], sono stati rimossi tutti i prodotti con prezzo minori di 5 e maggiori di 2000.

Train_id: rappresenta l'identificativo del prodotto nell'elenco.

Name: è il nome del prodotto sotto forma di dato non strutturato.

Shipping: rappresenta di chi, tra venditore e acquirente, sono a carico le spese di spedizione: il valore 1 significa "a carico del venditore", 0 invece dell'acquirente.

Il 45% dei prodotti è spedito a carico del venditore (Shipping=1), mentre il restante 55% a carico dell'acquirente (Shipping=0).

Ci si potrebbe aspettare che i prodotti spediti a carico del venditore abbiano prezzi più elevati; tuttavia per quanto riguarda il dataset in esame, è vero il contrario: il prezzo medio dei prodotti spediti a carico degli acquirenti, circa $30, è superiore a quello dei prodotti restanti, circa $22.

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. Si può supporre che il valore 1, il più frequente, identifichi la condizione migliore, mentre il valore 5 la condizione peggiore. Tuttavia, alla luce dei prezzi medi per ogni condizione, la supposizione sembra essere errata: i prodotti in condizione 5 hanno mediamente il prezzo maggiore, quelli in condizione 4 il minore e le condizioni 1, 2, 3 hanno prezzo medi molto simili.

Category_name: rappresenta la categoria di prodotto a cui appartiene l'articolo.

Circa 6000 gli articoli non hanno nessuna categoria assegnata, mentre i restanti si suddividono in 1287 categorie, ad esempio *"Women/Tops & Blouses/T-Shirts"*, ed è facilmente osservabile una gerarchia di sottocategorie dalla più generica alla più specifica.

Siccome in più del 99% dei casi il campo contiene 3 livelli di sottocategorie il campo *Category_name* è stato diviso in 3 campi, uno per ogni livello, assegnando la stringa "NA" a quegli articoli con uno o più livelli mancanti.

Brand_name: rappresenta il marchio dell'articolo; nel dataset sono presenti 4809 brand differenti e più di 600 mila valori mancanti: poco meno della metà dei prodotti totali; 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".

Inoltre, non sembra esistere una relazione lineare tra lunghezza delle descrizioni e prezzo, in quanto l'indice di correlazione di Pearson è prossimo a zero; 0.048 per l'esattezza.

Analizzando le word cloud ottenute dai bigrammi delle descrizioni dopo aver suddiviso i prodotti in quattro fasce di prezzo (figure 1, 2, 3 e 4) si riescono a notare delle differenze sulle coppie di parole più frequenti, identificabili perchè di dimensioni più grandi.

Nella wordcloud dei prodotti con prezzo maggiore o uguale a $100 (Figura 1) sono molto frequenti bigrammi che danno informazioni sulle buone condizioni dei prodotti, ad esempio: "100 authentic", "great condition" e "good condition". Al diminuire del prezzo questi bigrammi diventano meno frequenti; aumentano invece i bigrammi relativi a descrizioni mancanti.

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

Figura 2: Descrizioni dei prodotti con prezzo tra $50 e $100 esclusi

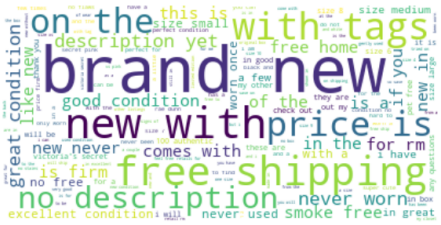

Figura 3: Descrizioni dei prodotti con prezzo maggiore di $30 fino a $50 incluso

Figura 4: Descrizioni dei prodotti con prezzo minore o uguale a $30

2.0.1 Preprocessing del testo

Il preprocessing 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 questo lavoro è stato preferito utilizzare la lemmatizzazione poiché usa un vocabolario per determinare la forma base di una parola, rispetto allo

stemming ovvero un processo euristico che rimuove le estremità delle parole cercando di ottenere la forma base corretta [3].

Siccome alcuni degli approcci di cui verrà discusso hanno dimostrato performance migliori con preprocessing limitato a conversione in minuscolo e rimozione della punteggiatura, in seguito, vi si farà riferimento con preprocessing limitato.

2.0.2 Encoding

Per quanto riguarda le variabili categoriche, ovvero *Brand_name* e le sottocategorie 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

3.1 Feature non testuali

Inizialmente sono state valutate le performance di regressione dei prezzi a partire dalle sole features non testuali in modo tale da stabilire un punto di partenza per la successiva analisi delle testuali, *name* e *item_description*, più complesse e computazionalmente costose.

È stato utilizzato un semplice modello composto da due layer Densi entrambi seguiti da un layer di Dropout impostato a 0.2 per contrastare l'overfitting ed infine un layer finale Denso con funzione d'attivazione lineare. Questo modello verrà in seguito ampliato per utilizzare anche le feature testuali a seconda dei vari approcci. La figura 5 ne mostra un template.

3.2 Rappresentazione del Testo

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

3.2.1 Bag of Word

Il modello bag-of-words permette di rappresentare il testo, come una sorta di "borsa" di parole dove per ogni "documento", o in questo caso per ogni

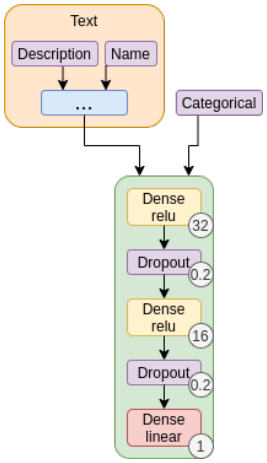

Figura 5: Template del modello con numero di neuroni e rate di dropout cerchiati

name o *item_description*, viene creato un vettore contenente il numero di occorrenze di ciascuna parola del vocabolario nel documento; di conseguenza questi vettori avendo cardinalità pari alla dimensione del vocabolario sono generalmente sparsi in quanto alle parole assenti dal documento corrisponde il valore 0. Regole grammaticali e l'ordine delle parole non sono rappresentabili tramite questa tecnica [4].

Ritornando al modello, per quanto riguarda Bag of Words i vettori delle feature testuali sono forniti in input direttamente al primo layer denso allo stesso modo delle variabili non testuali.

3.2.2 Tf-Idf

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

$$tf_{t,d} = \frac{n_{t,d}}{|d|} \tag{1}$$

$$idf_t = \log \frac{|D|}{|\{d : t \in d\}|} \tag{2}$$

$$Tf\text{-}Idf_{t,d} = tf_{t,d} \cdot idf_t \tag{3}$$

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

Per Tf-Idf come approccio di rappresentazione del testo si vuole indendere un approccio simile a Bag of Word i cui vettori, anzichè contenere semplicemente il numero di occorrenze della parola, ne contengono il valore Tf-Idf (3).

Per quanto riguarda il modello è stato utilizzato allo stesso modo di Bag of Words.

3.2.3 Word Embedding

Le precedenti tecniche sono semplici da realizzare, robuste e funzionali per numerosi task. Tuttavia sono poco performanti in alcune applicazioni poiché trattano le parole come unità atomiche, senza apprenderne relazioni complesse ad esempio di similarità o causalità.

Con lo sviluppo delle tecniche di Machine Learning si sono diffusi i cosiddetti *word embedding* che consentono di rappresentare parole per mezzo di vettori densi e di lunghezza fissa [5], fornendo di conseguenza una rappresentazione più efficiente rispetto alla *Bag of words*.

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

3.2.4 Keras Embedding Layer

Keras fornisce un implementazione di embedding sotto la forma di layer neurale e ne rende quindi possibile l'addestramento insieme al resto del modello.

Ritornando al modello, è necessario codificare le frasi di input sotto forma di vettori di interi, a tal proposito keras fornisce la classe *tf.keras.preprocessing.text.Tokenizer*; questi vettori saranno in seguito dati in input al modello.

Tenendo come riferimento il template in figure 5, 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.

In particolare sono stati utilizzati layer RNN poiché risultano ottimi per l'elaborazione di dati sequenziali, come il testo, essendo in grado di mantenere in "memoria" informazioni sugli elementi precedenti [7]; ad esempio sono in grado di cogliere relazioni tra le parole di una stessa frase.

Nello specifico sono stati considerati layer LSTM, GRU e LSTM bidirezionali di cui si parlerà più dettagliatamente in seguito.

Infine l'output di questi layer è stato assegnato come input al primo layer denso come mostrato. La figura 6 ne mostra uno schema.

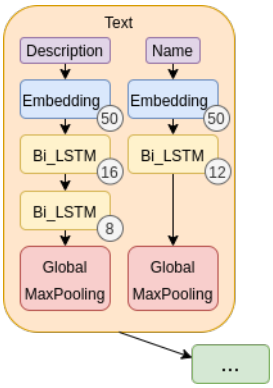

Figura 6: Schema riassuntivo sulla gestione dei dati testuali, dove per Bi_LSTM si intendono layer LSTM bidirezionali con dropout a 0.2 e i valori cerchiati identificano il numero di neuroni.

3.2.5 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 [8]. Sono inoltre disponibili modelli GloVe pre-addestrati; seguono quelli valutati nel progetto.

- Wikipedia 2014 + Gigaword 5: allenato su 6 miliardi di parole e con un vocabolario di 400 mila parole. Fornisce vettori 50 dimensioni ed è il più "piccolo" tra i pre-addrestrati.

- Common Crawl: allenato su 840 miliardi di parole e con un vocabolario di 2.2 milioni di parole; è il più grande e fornisce vettori di 300 dimensioni.

Per quanto riguarda il modello, trattandosi di word embedding, è stata utilizzata la stessa architettura del modello Keras assegnando i pesi preaddestrati e la corretta dimensionalità ai layer Embedding che sono stati impostati come non-trainabili.

3.2.6 Transformers

Il Transformer è un'architettura proposta successivamente alle RNN che vi si contrappone evitando la ricorrenza e utilizzando esclusivamente un meccanismo di *attention* per rappresentare i rapporti di dipendenza di input e output. Si basa su di una struttura *encoder-decoder* dove l'encoder fornisce al decoder una rappresentazione dell'input ed in seguito il decoder fornisce una frase in output [9].

Una delle più note architetture basata sul concetto di Transformer è **BERT**, ovvero *Bidirectional Encoder Representations from Transformers*; si tratta di un transformer multi-layer bidirezionale, cioè che dato un input è in grado di apprenderne relazioni sia con input precedenti che successivi, e che si basa esclusivamente su moduli encoder. Nasce al fine di fornire un modello pre-allenato adattabile semplicemente ad un vasto range di applicazioni tramite *fine-tuning*. Tuttavia anche approcci *feature-based* basati su BERT risultano efficaci [10].

In questo progetto è stato utilizzata una versione più leggera e veloce ottenuta da BERT tramite *Knowledge distillation*, ovvero DistilBert [**?**], con un approccio feature-based.

Ritornando al modello, occupa la stessa posizione dedicata agli embedding (figura 6), per entrambi i campi nome e descrizione.

3.3 Training e valutazione

3.3.1 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*.

3.3.2 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 (RMSLE)*, in quanto è la misura scelta dalla Kaggle challenge per confrontare

le performance dei vari partecipanti. Risulta inoltre adeguata al problema considerando il vasto intervallo dei valori dei prezzi.

Seguono la definizione e alcune osservazioni su di RMSLE e MSLE.
Mean Squared Logarithmic Error (MSLE), essendo calcolato a partire da un rapporto, riflette l'errore relativo e di conseguenza risulta efficace laddove i valore assoluti presentano variazioni considerevoli.

$$\frac{1}{n}\sum_{i=1}^{n}(\log(y_i+1)-\log(\hat{y}_i+1))^2=\frac{1}{n}\sum_{i=1}^{n}\log^2(\frac{y_i+1}{\hat{y}_i+1}) \tag{4}$$

Root Mean Squared Logarithmic Error (RMSLE) è la radice di MSLE.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i+1)-\log(\hat{y}_i+1))^2} \tag{5}$$

3.3.3 Valutazione dei costi computazionali

Per la valutazione dei costi computazionali sono stati annotati il numero dei parametri allenabili e non dei vari modelli, il tempo richiesto dall'allenamento sul trainset e quello richiesto dalla predict sul testset.

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

3.4 Parametri di training

Numero di neuroni Il numero di neuroni dei vari layer è stato impostato ai valori mostrati nelle figure 5 e 6 in modo da migliorare le performance di regressione.

Optimizer Come optimizer è stato utilizzato Adam impostando il learning rate a 1e-4, in quanto il training è risultato più stabile che con il default di 1e-3 per la maggior parte degli approcci ad eccezione dell'Embedding Keras che ha dimostrato un comportamento migliore con il learning rate di default.

Batchsize La batchsize è stata impostata a 256 dopo alcuni test sperimentali.

Regolarizzazione Oltre ai layer Dropout è stato testata la regolarizzazione L2, tuttavia non ha portato a evidenti miglioramenti ed è quindi stata omessa. Inoltre abbiamo utilizzato l'earlystopping sull'errore del validation con pazienza impostata a 3 epoche e in modo da ripristinare i pesi più performanti rispetto al validation.

4 Results and Evaluation

4.1 Modello finale

Per quanto riguarda i layer RNN le tipologie di layer valutate sono: LSTM, GRU e Bidirectional LSTM. Questi ultime, essendo in grado di catturare relazione sia con parole precedenti che con e successive [**?**], si sono dimostrati più performanti e sono stati utilizzati nel modello finale.

Sono stati inoltre valutati layer convoluzionali a monte dei layer RNN; tuttavia non avendo migliorato considerevolevolmente le performance sono stati omessi nel modello finale.

4.2 Transformers

Non è stato purtroppo possibile per motivazioni computazionali allenare questo modello sulla totalità del dataset ma è le limitazioni delle risorse di Google Colab hanno reso obbligata la scelta di utilizzare una piccola porzione del dataset per il training. Tuttavia i risultati sul validation sono particolarmente buoni considerando la ridotta quantità di esempi di training a disposizione rispetto ai modelli precedenti.

5 Discussion

The discussion section aims at interpreting the results in light of the project's objectives. The most important goal of this section is to interpret the results so that the reader is informed of the insight or answers that the results provide. This section should also present an evaluation of the particular approach taken by the group. For example: Based on the results, how could the experimental procedure be improved? What additional, future work may be warranted? What recommendations can be drawn?

6 Conclusions

Conclusions should summarize the central points made in the Discussion section, reinforcing for the reader the value and implications of the work. If the results were not definitive, specific future work that may be needed can be (briefly) described. The conclusions should never contain "surprises". Therefore, any conclusions should be based on observations and data already discussed. It is considered extremely bad form to introduce new data in the conclusions.

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] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: a comparison of retrieval performances," 2014.

[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] 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.

[8] 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.

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

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