Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Mercari Price Prediction

Authors:
Leonardo Riva - 830647 - l.riva37@campus.unimib.it
Riccardo Merlo - 829805 - r.merlo2@campus.unimib.it

January 24, 2022

Abstract

This project is taken from the **Mercari Price Suggestion Challenge** available on Kaggle [1]. The goal is to suggest the price of an item given some categorical and textual features. This has been achieved with some preprocessing of the textual data, as well as many analysis and grouping on the categorical data. Models with different level of deepness have been tried, including the use of Word2Vec and DistilBert. The results seem to show that the performances don't tend to increase with the dimensions of the model, with simpler ones scoring better results.

1 Introduction

The project is created after the Kaggle challenge "Mercari Price Suggestion Challenge" [1], whose goal is to automatically suggest product prices to online sellers. It's a regression problem, concerning the prediction of price given various product features, such as name, description, condition, etc. Different approaches are going to be considered; in particular, different techniques to manage textual features, such as Keras embeddings, Word2Vec and DistilBert. To evaluate these different models, the metric used is the Root Mean Squared Logarithmic Error (RMSLE). The reason is that, this way, the performances are directly comparable to the ones on the website leaderboard.

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

2 Datasets

The dataset is composed of a train and test csv. Since the challenge is closed, labels for test are not available; therefore, only the train file has been used.

The dataset contains 1.39 million entries, which are described by the following features:

- **train_id** [int]. Unique id for each entry.
- **name** [str]. Product name (e.g. *Celine Tri Colored Suede Mini luggage*).
- **item_condition_id** [int]. Product state, encoded as a rating from 1 (New) to 5 (Poor condition).
- **category_name** [str]. Product category. While being a string, it is a hierarchical structure of 3 sub-categories, separated by a "/" (e.g. *Women/Women's Handbags/Totes & Shoppers*).
- **brand_name** [str]. Product brand (e.g. *Celine*).
- **shipping** [int]. Represents whether the seller (1) or the buyer (0) will pay for the shipping.
- **item_description** [str]. Longer product description.
- **price** [float]. Label to be predicted.

2.1 Exploration

The dataset presents missing values: 0.4% of category names, 42.7% of brand names, 0.0003% of descriptions. Every other feature is complete.

The train_id feature does not actually look useful for our study.

The price feature is not well distributed, as shown in figure 2.1. This suggests that applying a logarithm will make the neural network perform better.

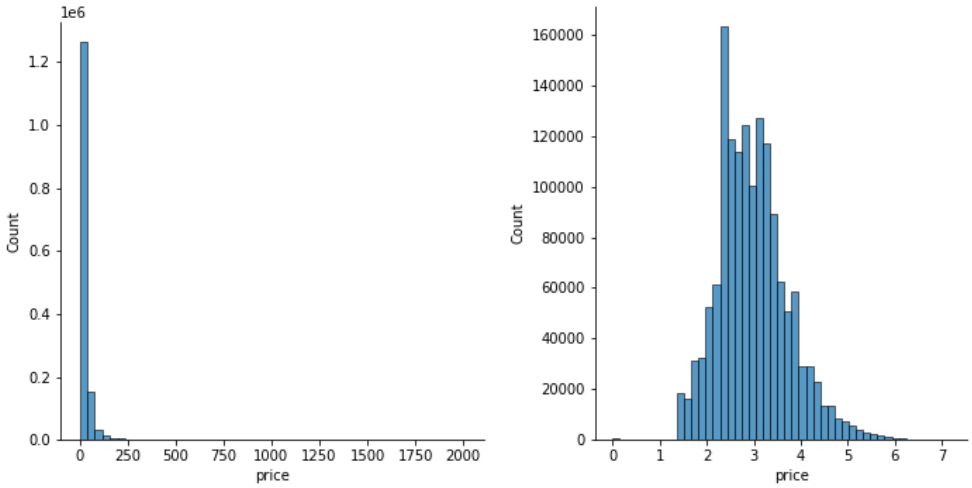

Figure 2.1: Price distribution before and after applying a logarithm.

In figures 2.2 and 2.3 we can look at some features distributions (not only existing features but also two new attributes, discussed in the next section). Each of them is, in some way, correlated to the price.

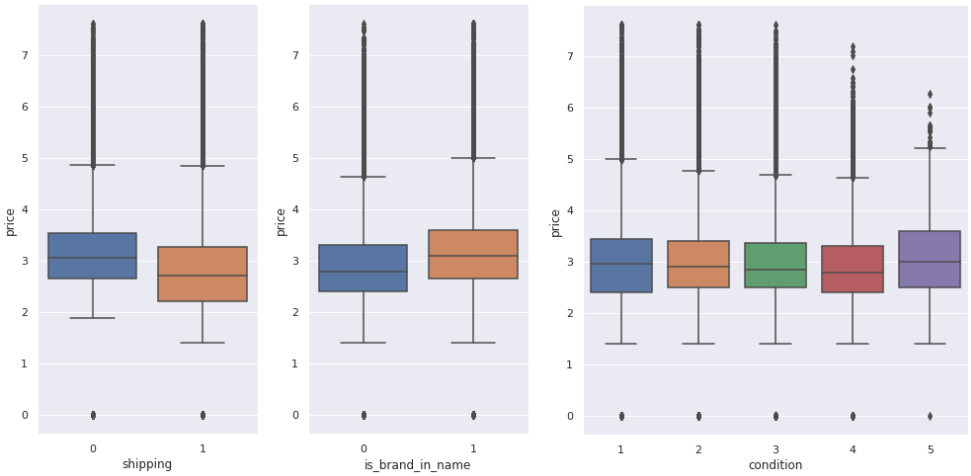

Figure 2.2: Distribution of price over shipping, is_brand_in_name and condition.

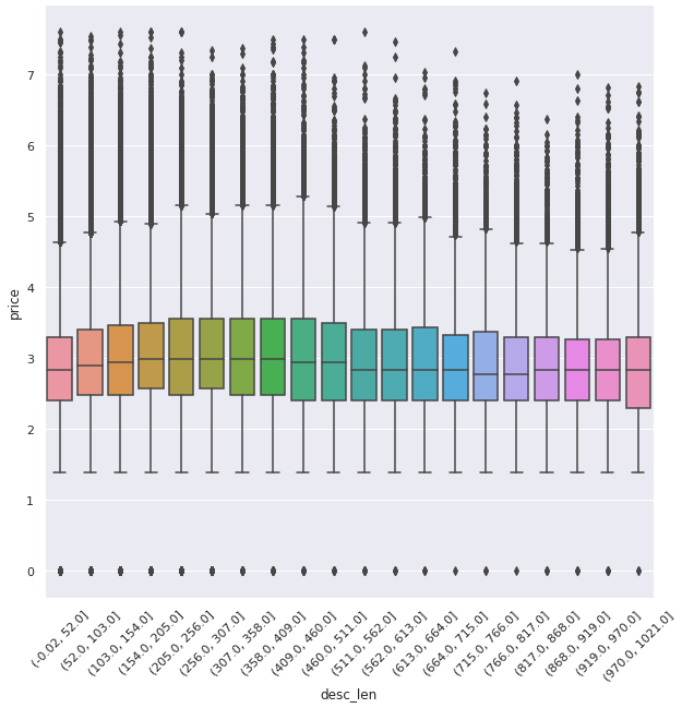

Figure 2.3: Distribution of price over description length, at intervals of 50 characters at a time.

2.2 Preprocessing

- deleted train_id.
- split category feature into 3 different ones. There are now 11 category1, 114 category2 and 859 category3.
- logarithm of price, as shown before.
- replace missing values and "no description yet" with "missing", in order to avoid problems training neural networks.

2.2.1 New features

Two new features have been created because they proved to be correlated in some way to the price. The first is *description length*; the second is *is brand*

in name, which is 0 if the brand doesn't appear in the name feature, 1 if it does.

2.2.2 Encoding

For string features that are actually categorical, their values have been encoded into integers through a LabelEncoder [2]. This applies to the brand name and the three categories. After that, every integer feature will be normalized into [0,1] For the other string features, different form of embedding will be discussed.

2.2.3 Textual features

For string columns, such as name and description, a deeper pre-processing has been performed:

- lowercase.
- removed links.
- removed emojis.
- removed stopwords.
- removed punctuation.
- removed excessive whitespaces.

2.2.4 Validation and test data

Since we had no access to the test data, the train dataset has been split into validation and test, both 10% of the training data. This size appears to be enough to have valid and consistent results.

3 The Methodological Approach

3.1 Model 1

The very first model has been designed to learn only on non-textual features (which are obviously easier to manage). This way, we have a score to use as a baseline.

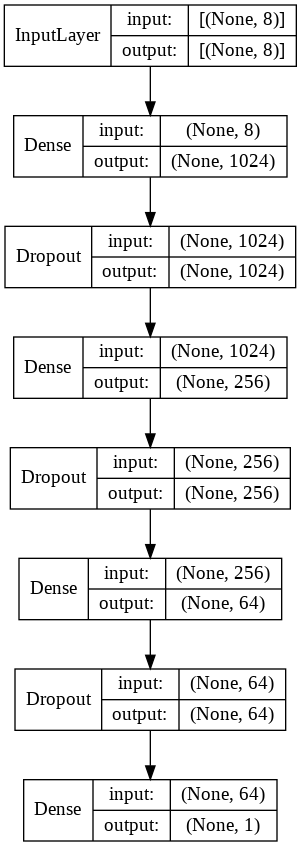

Figure 3.1: The model is structured with just Dense layers with ReLU activation function, alternated with Dropout layers. The final layer is a Dense of 1 neuron with a linear output function.

The training is characterized by a batch size of 512 and *adam* as optimizer,

for 100 epochs, but using an Early Stopping (patience = 3) and a ModelCheckpoint callback (which only saves the best model regarding loss on validation).

The model reaches an rsmle of 0.59514 on the test set.

3.2 Model 2

First approach to textual features has been to use Keras embeddings. Keras provides an Embedding layer that creates word embeddings inside the network, after converting words to tokens previously [3]. The maximum length of *name* and *description* token sequences are picked by looking at their distribution, in such a way that most outliers are not considered; in this case, 10 for the *name* and 75 for the *description*.

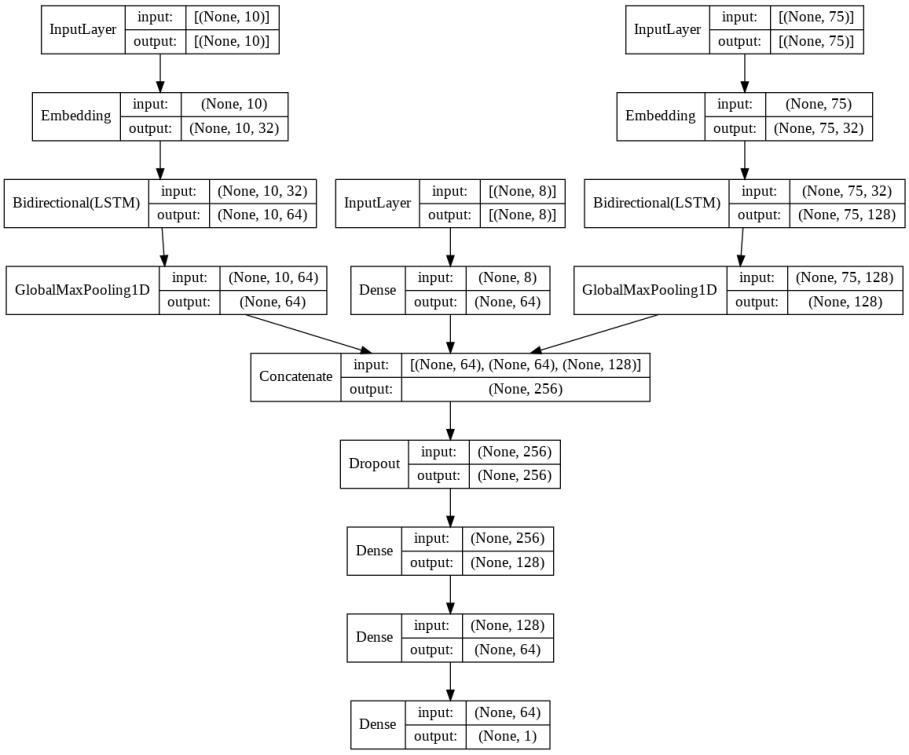

Figure 3.2: Model2 structure.

In figure 3.2 we see the textual features on the top center; after an attempt of freezing model1 and putting it inside model2, the most performing way

is to actually re-learn from scratch. The textual inputs are linked to an Emmbedding layer and, after that, a bidirectional Long Short-Term Machine. Finally, after a concatenation, some final dense layers.

The training is performed in the same way as model1, but with a batch size of 1024. The model reaches an rsmle of 0.45613 on the test set.

3.3 Model 3

Second approach to textual features has been to use Word2Vec, a neural network based architecture for computing continuous vector representations of words from very large data sets [4]. The *Google-News-300 model* [5] has 300-dimensional pre-trained vectors on a part of the Google News dataset (about 100 billion words) and contains vectors for 3 million words and phrases.

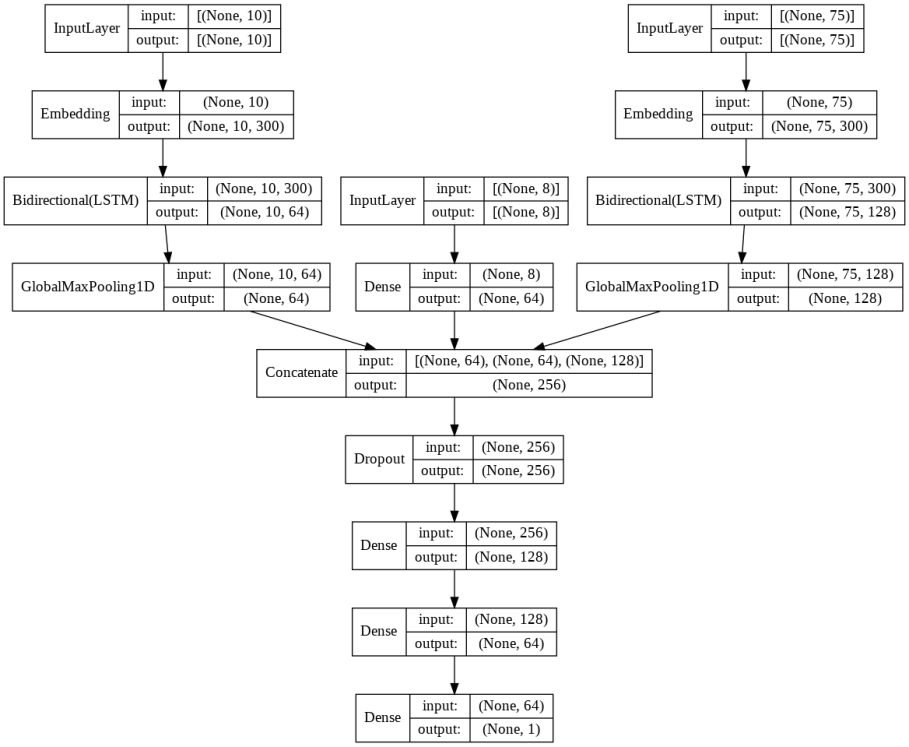

Figure 3.3: Model3 structure.

Like before, both the textual features have been tokenized and then mapped, creating a dictionary from all the words from the sentences, converting each

input to a list of fixed length of dictionary keys. Next, an embedding matrix is created and each word of the dictionary is associated to its vector in the Word2Vec model, if it exists. This matrix is then used to updated the weights of the Keras Embedding Layer.

The structure of the model is exactly as model2. But not as expected, this model perform worse, with an RMSLE of 0.46677.

3.4 Model 4

The final approach to the textual features has been BERT, an architecture based on the encoder-decoder connected through an attention mechanism [6]. More precisely, DistilBERT has been used: a small, fast, cheap and light Transformer model based on the BERT architecture obtained through knowledge distillation [7] [8].

Same as for the previous model, both textual features have been tokenized, this time using DistilBERT's specific fast version [9]. This time size of 100 has been used for both *name* and *description*.

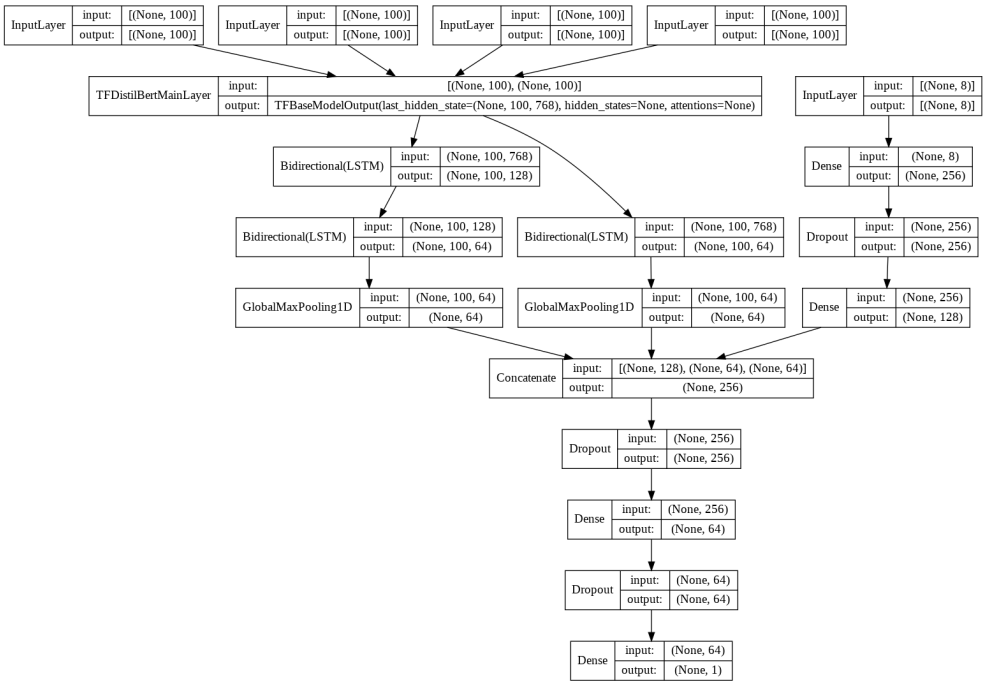

Figure 3.4: Model4 structure.

The first layer of the model creates the embedding of the input and feeds them to a Bidirectional LSTM layer with some dropout, followed by a GlobalMaxPooling.

Due to the higher computational cost of training with BERT, a TPU environment is necessary.

Again, the result was worse than expected, with an RMSLE of 0.47012.

4 Results and Evaluation

In the following table there is a summary of the models' performances and their overall training characteristics.

Model	TotalParams	TrainingParams	TrainingTime	RMSLE
Without text	288,129	288,129	12 min	0.59514
Keras	12,170,113	12,170,113	42 min	0.45613
Word2Vec	113,395,921	313,921	60 min	0.46677
DistilBERT	67,087,361	724,481	151 min	0.47012

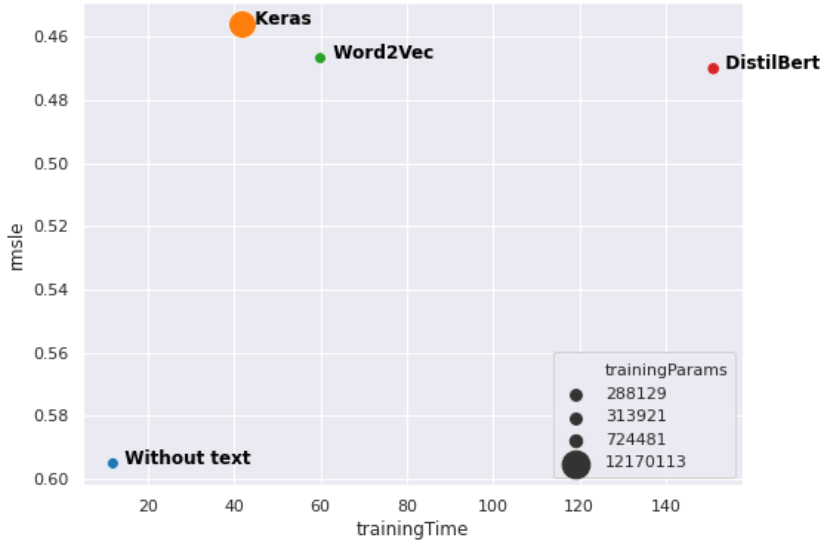

Figure 4.1: Scatterplot comparing results of these models.

5 Considerations

5.1 On data

Missing values play probably an important role in correctly predicting the price: 42% of brands not available are a lot. One additional improvement to the preprocessing could be to create a model that predicts the brand name from the textual features. Obviously this process can introduce noise and would need to be done carefully.

In some cases, the price label has a value of 0, which doesn't seem an intuitive price. Those products might be auctioned. Anyways, an improvement can certainly be to understand this phenomenon.

5.2 On methodology

While LSTMs should learn much better when the source sentences are reversed [10] (not the case with BERT), unfortunately this didn't improve the performances of the models on which it has been tested; a reason may be the composition of sentences, which are generally summarized in few words not well written.

Stacked LSTMs [11] brought improvements to the results only in the last model.

The non textual model has been developed to actually understand how much *name* and *description* would affect the prediction. Word2Vec has been chosen due to his reliability and efficiency without being too expensive computationally, Keras Embedding as an alternative to Word2Vec, to avoid pre-training a model, serving the same purpose but in a more comfortable way. BERT has been chosen because it is the state-of-the-art for text based tasks.

6 Conclusions

The first model, without textual features, has the worst performances of them all (as expected).

The comparison between Keras and Word2Vec embeddings slightly tends towards Keras, as counterintuitive as it could be. Word2Vec, in fact, is pre-trained and already has solid information about words; Keras, on the other hand, has to learn only on text input. The hypothesis is that, this way, it learns on weird words that are not present in W2V dictionaries.

Regarding transformers, so DistilBERT, it performed slightly worse compared to the previous models despite being on paper the most powerful, on the other hand the long training duration did limit its use.

In conclusion, the results are not directly comparable with those on the Kaggle leaderboard, because the train set has been reduced in order to create validation and test set (which is different from Kaggle's).

Bibliography

[1] *Mercari Price Suggestion Challenge*. URL: https://kaggle.com/c/mercari-price-suggestion-challenge (visited on 01/20/2022).

[2] *Sklearn.preprocessing.LabelEncoder*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[3] *Tokenizer*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer (visited on 01/20/2022).

[4] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. eprint: arXiv:1301.3781.

[5] *Word2Vec*. URL: https://code.google.com/archive/p/word2vec/ (visited on 01/20/2022).

[6] Ashish Vaswani et al. *Attention Is All You Need*. 2017. eprint: arXiv:1706.03762.

[7] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. eprint: arXiv:1910.01108.

[8] *DistilBERT*. URL: https://huggingface.co/docs/transformers/model_doc/distilbert (visited on 01/20/2022).

[9] *DistilBertTokenizerFast*. URL: https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertTokenizerFast (visited on 01/20/2022).

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. eprint: arXiv:1409.3215.

[11] *Stacked Long Short-Term Memory Networks*. URL: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/ (visited on 01/20/2022).