Università degli studi di Milano-Bicocca

Advanced Machine Learning

Final Project

Mercari Price Suggestion

Authors:

Alessandro Pigato - 852482 - a.pigato@campus.unimib.it

Davide Banfi - 806539 - d.banfi@campus.unimib.it

Giacomo Cesareo - 805716 - g.cesareo@campus.unimib.it

2020-2021

Abstract

The project focuses on the prediction of prices knowing properties associated to a set of products reachable on Mercari, a Japanese online marketplace founded in 2013. The high heterogeneity of variables, spacing from textual to numeric classes, required particular attention operating-wise and this is the reason why preprocessing phase covers itself the first half of the entire work. The cleaned data is then fed into three unique ANN to apply a prediction. Each is characterized by some unique layer-wise architectures: for instance, the first embeds simple dense-only layers to handle every kind of variable, while the second and the third respectively make use of recurrent and convolutional layers, or a combination of both. Since the test data made available by Mercari does not have the dependent variable, the evaluation of models has been carried out on a partition of the training data and eventually applied to map the distribution of the yet unknown prices.

Contents

1 Introduction **3**

2 Dataset **3**
- 2.1 Dataset exploration 4
- 2.2 Preprocessing 5
  - 2.2.1 Handling missing values 5
  - 2.2.2 Handling textual variables 6
  - 2.2.3 Labels encoding 7

3 Methodological approach **7**
- 3.1 Models 8
- 3.2 Evaluation measures 10

4 Results and evaluation **10**

5 Discussion **13**

6 Conclusions **13**

List of Figures

1 Price distribution 4
2 Example of dirty text form 5
3 Histograms of textual variables 6
4 Basic model structure 8
5 LSTM model structure 8
6 CNN/GRU model structure 9
7 Performance basic model 10
8 Performance LSTM model 11
9 Performance CNN/GRU model 11
10 KDE of the LSTM model 12

List of Tables

1 Performance of each model on both sets 11
2 Performance on test set 12

1 Introduction

Every day billions of items are put up for sale through e-commerce platforms as IoT makes possible to address to an audience as wide as never before. Goods are often listed in a catalogue having short but effective "descriptions" whose aim is to induce the customers to buy. However, in such platforms there is no real contact between the involved seller and buyer and these details, collected in variables, slowly assume a key role in an IoT society. It is safe to say that the volume of sales in e-commerce platforms now depends on the ability of each variable to be senses appetizing as a unique product can be perceived differently by the same client.
In particular, the item price embeds a strict connection with the item properties and having either one of these gives a rough idea about the other.
The Mercari challenge relies on this connection to advise sellers on the prices their listed goods can have, and this is done by accomplishing a regression where items' characteristics are known.

2 Dataset

The dataset can be found on Kaggle[1] and is divided into two files being the testing data, where the price is not available and has to be predicted, and the training data. Each item is described by a set of six relevant explanatory variables.

1. *name*: the title of the listing.

2. *item_condition_id*: the condition of the items provided by the seller, it is made of numeric values about the quality of the item, with 1 being "new" and 5 being "very poor".

3. *category_name*: it represents the set of categories and/or subcategories which the item belongs to. It can be either one or many, with multiple subcategories being divided by a slash.

4. *brand_name*: name of the product brand.

5. *shipping*: it is a binary variable indicating the person in charge of shipping fees. If 1 the costs are paid by the seller, otherwise by the customer.

6. *item_description*: a brief summary on the item properties such as the manufacture quality.

7. *price*: this is the target/dependent variable.

It is easy to spot the diversity of the available variables with some being purely textual and others being numeric, either binary or quantitative. In the preprocessing phase yet another class is created as some textual labels were converted into numeric encodes.

2.1 Dataset exploration

It is immediate and mandatory to visualize the distribution of the dependent variable as it will be useful to better understand the output returned by the models, this is represented in *Figure 1*.

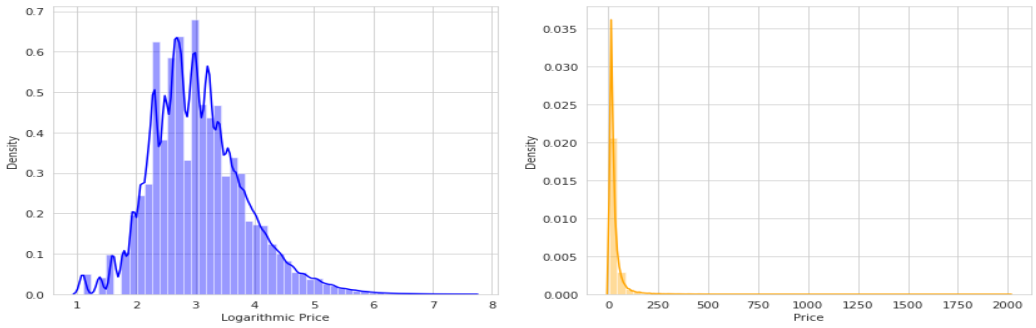

Figure 1: Price distribution

Chart spaces within the range [0, 2009] dollars with the 95% of prices falling in the interval up to 86$. An improved visualization can be achieved by considering a logarithmic scale.
It is worth spotting that 50% of outliers is associated to products designed for women, probably high-quality manufactures, and that some items have a listing price of 0$. Even though outliers were retained during the model training phase, the same did not apply to 0$ items as they are not useful to predict a consistent real value.
An in-depth analysis on how the prices are distributed across categories can be found in the python notebook *Exploratory analysis.ipynb*.
For what concerns textual columns, these give birth to a long list of problems

due to the randomness arising from the nature of the person that produced the text. The absence of a standard structure and the extreme use of special characters and symbols, as shown in *Figure 2*, required an accurate application of advanced NLP techniques.

Figure 2: Example of dirty text form

The presence of many incomplete informations (NaN) mainly located in brand and category name attributes required a solution as well.

2.2 Preprocessing

The very high heterogeneity in the data makes the preprocessing a crucial phase which can have huge worse counterparts on the model development if not done correctly. The preprocessing relies on three key points:

- The removal of NaN values;
- The encoding of categories in unique integers;
- The standardization of textual objects.

2.2.1 Handling missing values

The estimate of the price was established to consider all the six explanatory variables in order to be as accurate as possible. Records where the information in at least one field is not available were removed, decreasing both the test and the train datasets to half their original size. Nonetheless, the loss of roughly 700.000 records is well compensated by a better performance achieved by models.

2.2.2 Handling textual variables

Textual variables are very dissimilar and filthy in their structure and is not possible to map them to numbers at the current state.
Texts therefore underwent a series of handcrafted ad-hoc functions which sequentially split by punctuation and special characters, removed the latter, expanded the contracted forms, uniformed accents to a set of standards, converted everything to lower case, removed the stop words and lastly tokenized the sentences in cleaned lists of words.
By iterating on every list and token it was possible to come up with a vocabulary containing 101.154 unique words from 1.692.852 total strings. The *Figure 3* below shows how the lists of tokens distribute in length for the preprocessed variables *name* and *item_description*.

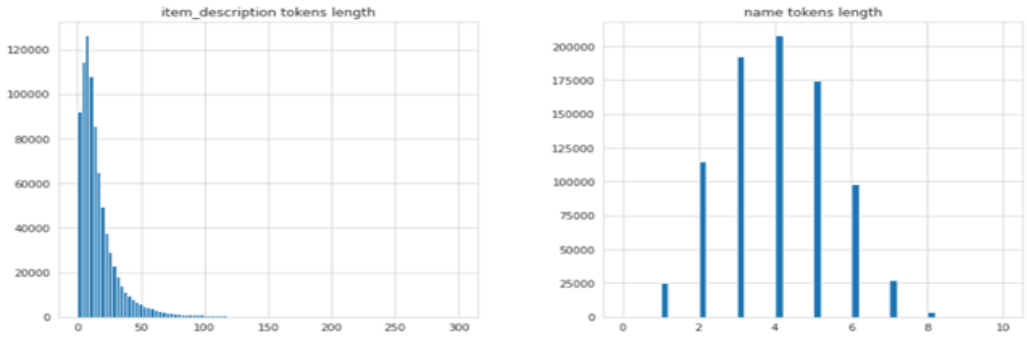

Figure 3: Histograms of textual variables

From this vocabulary, a dictionary of relations words-indexes has been eventually built.

Example of the preprocessing phase:

"I'm selling all my Pokémon cards! I'll ship the-cards-on-Monday."

↓

"am selling all pokemon cards will ship cards monday"

↓

[1, 2, 3, 4, 5, 6, 7, 5, 8]

As the use of embedding layers is strongly advised for textual variables, an extra task that has been accomplished was the mapping of unique terms in vectors of 300 floats. This operation has been carried out thanks to the algorithm Word2Vec and its ability to capture the semantic similarity of words by simply considering a context window of 5.

2.2.3 Labels encoding

When feeding a neural network, all the inputs have to be numeric for the model to "understand". While pure textual objects require more complex methods to achieve the conversion in numbers, there are attributes which are constituted by single words just like in the case of *brand*, and for which the transformation is done by simply associating every unique term to a unique integer. This procedure can be done as well with *category_name*, but first some preparation is needed. This field can have up to three labels separated by a slash, hence a function that splits the values in different columns is handy to later apply an encoding on newly created sub-columns.

3 Methodological approach

The test set provided by Kaggle does not have the target variable and does not permit to validate the results. The suggested approach was therefore splitting the train dataset, where the target variable is known, in an effective training batch and a new 20% sized artificial test batch. This made possible the evaluation of the results as the prediction could be compared to effective values.
A total of three models have been designed by providing a distinctive architecture: in particular, the root idea was to extract features following a column-specific criterion and then accomplishing an aggregation to predict the final target.

3.1 Models

A total of three models have been proposed to predict the target variable. In the first one, the basic model (*Figure 4*), numbers and categories have been handled strictly with dense layers and then aggregated at a deeper stage.

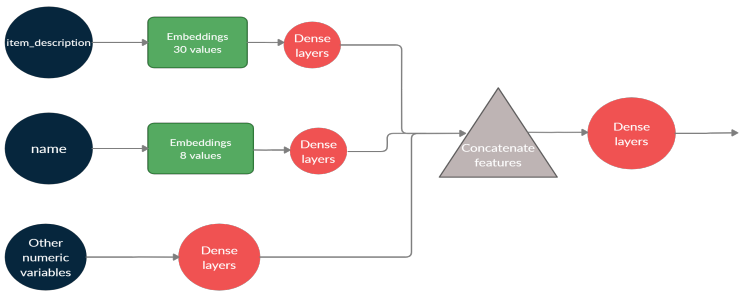

Figure 4: Basic model structure

The second architecture (*Figure 5*) still makes use of dense layers for numeric variables, however it exploits a LSTM layer to handle the flows of words.

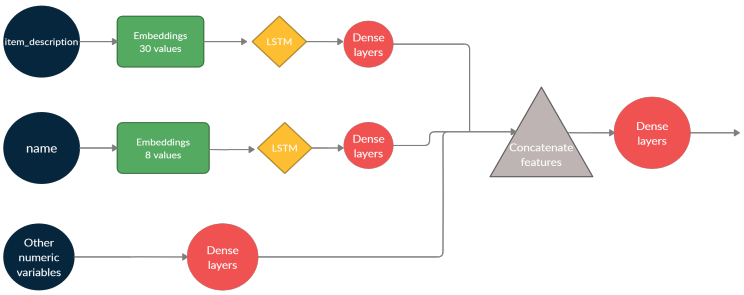

Figure 5: LSTM model structure

At last, the third model (*Figure 6*) has been built by combining the power of the convolution and a gated recurrent unit layer just after the embedding stage.

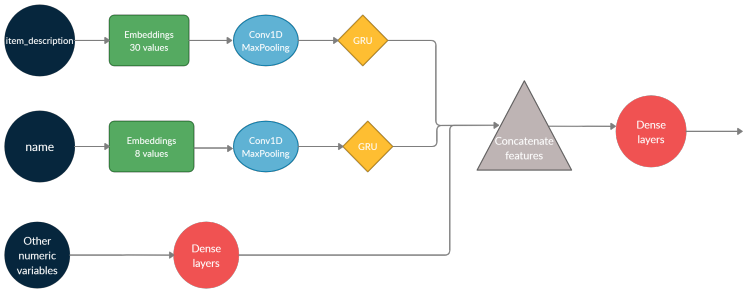

Figure 6: CNN/GRU model structure

Architectures are designed to handle in a unique way textual inputs, yet they share some common aspects. In fact, each has been provided with an embedding layer to give Keras the possibility to find its own best encoding vectors, while numeric variables have been mainly handled by fully connected neurons as embeddings are not suitable for these types of objects.
The length of each embedding array has been chosen by considering the mean/median length of the variable that it maps: with reference to *Figure 3*, arrays mapping *item_description* were set to be 30 elements long as it is the average length of tokens lists in the relative column.
The non-linearity was provided by ReLU at all levels as theory suggests it works better than most of the alternative activation functions.
Each ANN also shares an equal collection of hyper parameters to be able to confront performances when running in similar conditions. For instance, the chosen loss function has been set to be the mean squared error, to be minimized in up to 15 epochs using Adam as optimizer (learning rate equal to 0.01) and with a batch size of 128 observations.
Nonetheless, these parameters and the complexity of networks initially entailed overfitting which has been limited consistently by introducing dropout layers and L2 regularization.
The performance behaviors, illustrated in the following paragraphs, also required an early stopping method to return the best parameters.
For a deeper insight on the number of neurons and intensities of regularizing factors please address to the models' notebooks.

3.2 Evaluation measures

The suggested metrics for a regression task are *mean squared error* (1), *mean absolute error* (2) and *mean absolute percentage error* (3) defined as follows[2].

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad (1)$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \qquad (2)$$

$$MAPE = \frac{100}{N}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}| \qquad (3)$$

The decision to monitor the mean squared error on the validation data rather than the other measures took into account the downsides of the alternatives. In fact, MAE is not scaled to the average true values while MAPE suffers an exploding behavior when the actual value is very close to zero (price is a positive continuous variable and can therefore be very small).

4 Results and evaluation

Evaluation has been accomplished on a 20% partition of the training data. Below are reported the performances of the base (*Figure 7*), LSTM (*Figure 8*) and CNN/GRU (*Figure 9*) models across the epochs.

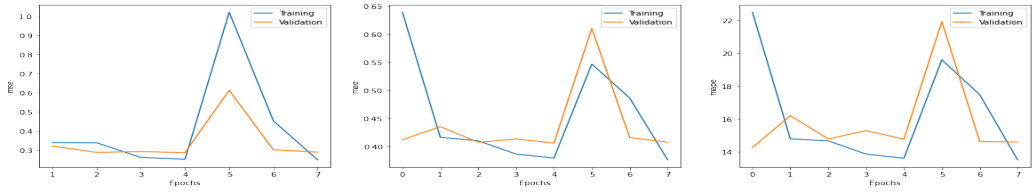

Figure 7: Performance basic model

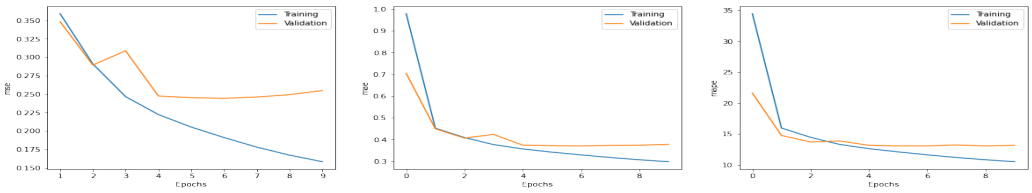

Figure 8: Performance LSTM model

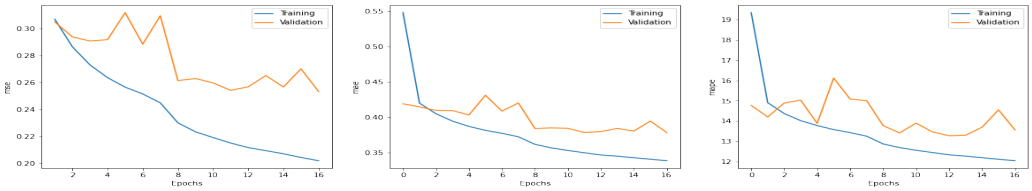

Figure 9: Performance CNN/GRU model

It is noticeable how the validation error of the third model seems to have a decreasing trend which differs from the swinging behavior of the first and the stabilizing behavior of the second. This is also the reason why it was necessary to introduce an early stopping criterion[3] if a variation of at least 5% had not been accomplished within 5 epochs.
Even though measures strongly depend on the selected random observations, the average final score is situated approximately around the values reported in *Table 1.*

Table 1: Performance of each model on both sets

	Basic Model		LSTM		CNN/GRU	
	Train	Validation	Train	Validation	Train	Validation
MSE	0.3427	0.2870	0.2218	0.2474	0.2131	0.2541
MAE	0.4095	0.4072	0.3558	0.3737	0.3480	0.3784
MAPE	14.6212	14.7797	12.5894	13.1584	12.3881	13.4615
Loss	0.6535	0.4772	0.2218	0.2474	0.2316	0.2721

MAPE and MAE are still made available for the reader to freely choose a metric that suits its tastes.

Every model has then been applied to predict the prices of the artificially created test set and produced good results, as can be seen in the *Table 2*.

Table 2: Performance on test set

	MSE	MAE	MAPE	Loss
Basic Model	0.2875	0.4081	14.7714	0.4777
LSTM	0.2472	0.3732	13.1279	0.2472
CNN/GRU	0.2517	0.3771	13.4214	0.2697

The kernel density estimation plot is a useful tool to visualize the concentration of scattered points having coordinates actual and predicted[4].

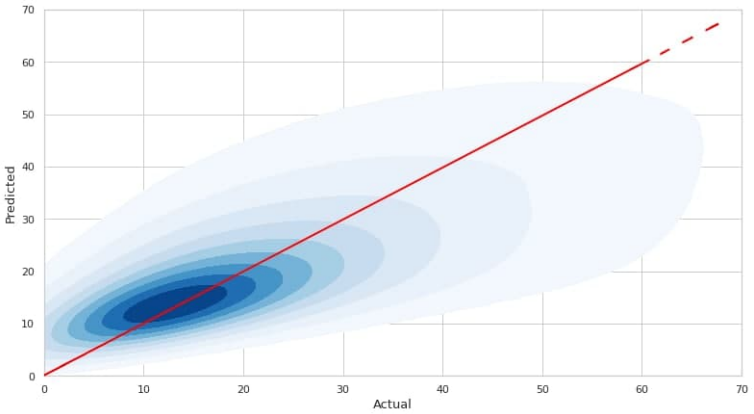

Figure 10: KDE of the LSTM model

Figure 10 compares the predictions returned by LSTM model with the expected values, however all three KDE plots diverge imperceptibly from each other. Having verified models' goodness, the last step was their employment so as to predict the price of the original test set not having the target variable. In this case, the goodness of predictions could not be explicitly measured but it could be inferred anyway by looking at the distribution of predicted values being very similar to all those encountered up to this point.
For a deeper insight on this last part please refer to the notebooks.

5 Discussion

Even though all the models turned out to be satisfactory as they achieve a relatively low logarithmic MSE, LSTM seems to be slightly better than the others as it achieves a prediction error of just 1 dollar when evaluating on a non-logarithmic scale.

$$RMSE = \sqrt{e^{0.2472}} = 1.13\$$$

The best alternative does not change even when considering different measures or criteria for the selection (MAE, MAPE, loss). This may be caused by the fact that a model with mainly dense layers, which performs worst in all the metrics, is too generic and is not able to extract specific key details that help in lowering the prediction error.
During the development of the project work, many experiments were made even though only the top three results are reported here: initial ones demonstrated that without any sort of regularization the difference between performances on train, validation and test sets was too wide to not be negligible. The highest importance has therefore been given to the measures computed when predicting test's target rather than train's.

6 Conclusions

All these results are just an approximation as they may slightly vary due to the randomness by which observations are selected. Future applications resort to enhancing the neural networks by employing more advanced techniques inherent to NLP, such as the lemmatization and stemming, or by cleaning the data more accurately. As well as that, architectures could be made more complex and eventually simplified through model compression or gaining a boost by augmenting the data. It is worth mentioning that less intense regularizations were attributed to the LSTM model even if this is the best in all metrics. Looking at Kaggle's scores leaderboard, this project achieved an MSLE that is 33% lower than the top one on the platform. Nothing can be said for sure about the correctness of the methodologies as they might differ, or whether Kaggle's leading project has been strongly limited by the absence of newer advanced libraries (since it was made 3 years ago), however despite the applied procedures and results, the proposed models are still valid to be applied in concrete contexts.

References

[1] Kaggle dataset,
https://www.kaggle.com/c/mercari-price-suggestion-challenge/data

[2] Evaluation measures,
https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d

[3] Early stopping,
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[4] Kernel density estimation,
https://jakevdp.github.io/PythonDataScienceHandbook/05.13-kernel-density-estimation.html