Machine Learning Engineer Nanodegree

Price Prediction for ecommerce products

MANCEÑIDO, AGUSTIN
March 14th, 2019

I. Definition

Project Overview

Using the official application of an e-commerce site, a historical sales database corresponding to a product category has been produced. By using this dataset containing all the information related to the products and their corresponding price, it is possible to train an algorithm capable of predicting the optimal competitive price for the publication of a new product. The motivation for pursuing this project is due to the fact that e-commerce companies require a holistic understanding of prices, and they must establish it intelligently as a marketing weapon.

Problem Statement

Given a dataset of historical sells corresponding to a product category of an e-commerce site, it is necessary to develop an algorithm to predict the optimal competitive price for a new product to be published. The algorithm needs to determine what is the relevant data in order to get the best prediction for the price taking into account seasonal prices, market fluctuations, economic trends, better sellers, etc. The final algorithm can be part of the publication service of the e-commerce site, using the information provided by the seller and returning the suggested price for the publication. The predicted price can also be used as a starting point for additional pricing services such as price optimization and dynamic prices.

Metrics

The measure of performance for this project will be the Mean squared logarithmic error (MSLE) [1] between the predicted and actual values of the product price for a test subset of values extracted from the dataset.
This metric is particularly convenient for this problem because it penalizes to an under-predicted estimate greater than an over-predicted estimate, which is appropriate to avoid losses due to lower prices.

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e(1+y_i) - \log_e(1+\hat{y}_i))^2.$$

Where $\log_e(x)$ means the natural logarithm of x. This metric is best to use when targets having exponential growth, such as population counts, average sales of a commodity over a span of years etc.

II. Analysis

Data Exploration

Datasets and Inputs

The dataset to be used on this project has been collected from an Argentine ecommerce site [2] using the official API [3]. It consists 795295 records of sells with with 8 features each, all taken between 03/17/2017 and 05/10/2018.

	title	sellerid	state	city	pos	sqty	price	created_at
795290	Mampara De Baño Fija Vidrio Blindex Laminado 6...	315455008	Capital Federal	Villa Urquiza	781.0	1	1770.0	2018-05-10 03:14:44
795291	Bureta De Vidrio De 25 Cc Robinte De Vidrio	241479696	Buenos Aires	Don Torcuato Este	532.0	1	260.0	2018-05-10 03:20:52
795292	Turbina Odontologica Nsk Pana Max	196185272	Capital Federal	Villa Devoto	4132.0	1	1455.0	2018-05-10 03:20:52
795293	Olivas Littmann. Original. Nuevas!!	189552013	Capital Federal	Abasto	4138.0	1	569.0	2018-05-10 03:20:52
795294	Solución Iodo Povidona Tópica O Jabonosa	316678373	Buenos Aires	Sáenz Peña	4782.0	1	83.0	2018-05-10 03:20:52

Table 1 - Dataset sample

The following features are included:

- **title**: name of the product provided by the seller.
- **sellerid**: unique id for each seller.
- **state / city**: seller location.
- **pos**: position where the product was found in the search.
- **sqty**: number of products selled.
- **created_at**: timestamp for the sell.
- **price**: price of the product.

Some statistical values are presented for the `price` feature, which the algorithm will try to predict:

count	795295.000000	25%	463.000000
mean	1849.462588	50%	1023.000000
std	2723.852485	75%	2099.000000
min	1.000000	**max**	256000.000000

In the exploratory visualization it is possible to notice that this characteristic is positively skewed and it will be necessary to correct it in order to obtain better prediction results. In addition, 953 missing values were found for the `pos` characteristic that will also need to be corrected.

Finally, these are the number of unique categorical features:

sellerid:	18038 unique values.	city:	2210 unique values.
state:	24 unique values.	state_city:	2307 unique values.

A full exploration of the dataset is performed in the `capstone_exploration.ipynb` file.

Exploratory Visualization

By analyzing the data set it is possible to verify that the average number of records varies with time, showing a clear decrease in sales on Fridays and Saturdays, as well as in the evening hours.

Figure 1 - Record density by time

In addition, it is possible to observe that the value we are trying to predict is positively biased and that it can be easily corrected by applying a logarithmic function.

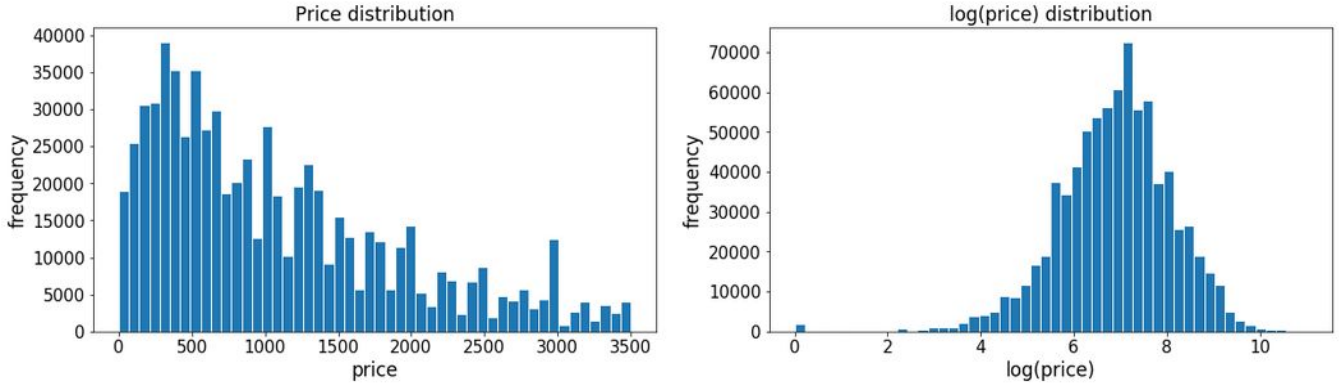

Figure 2 - Price and log(Price) distribution

Finally, two graphs showing the most common words used in the title of the product for the best and worst sellers can be seen below.

Figure 3 - Word cloud for best and worst sellers

Algorithms and Techniques

A Neural Network is a computational model that is inspired by the way biological neural networks in the human brain process information. Neural Networks have generated a lot of excitement in Machine Learning research and industry, thanks to many breakthrough results in speech recognition, computer vision and text processing. In this blog post we will try to develop an understanding of a particular type of Artificial Neural Network called the Multi Layer Perceptron [4].

The basic unit of computation in a neural network is the perceptron, often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs.

A Multi layer perceptron (sometimes referred to as “Fully Connected Layers”) contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non–linear functions. Every hidden layer tries to detect patterns and when pattern is detected the next hidden layer is activated and so on. The more layers in a neural network, the more is learned and the more accurate the pattern detection is. By applying Neural Network techniques a program can learn by examples, and create an internal structure of rules to classify different inputs [5].

A Neural Network has been defined and trained to solve the prediction problem because of its proven effectiveness in similar areas such as price forecasting and stock prediction. This technique can also infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data, and are quite simple to implement.

Benchmark

For this project, two benchmark models were selected. First, a Persistence model [6] will be used as a really simple and fast benchmark model. Then, a second and more advance model will be used to get a better idea of the performance using all the features available.The second one is XGboost [7], that is a quick and efficient and implements machine learning algorithms under the Gradient Boosting framework that is easy to use and is also enormously flexible. Over the past year and half 50% of the Kaggle competitions have used XGboost as a key part of the winning solution according to Ben Hamner, CTO of Kaggle [8].

The complete benchmarking process can be found in the `capstone_benchmark.ipynb` file.

III. Methodology

Data Preprocessing

After a complete analysis and exploration of each feature of the dataset, which can be found in the `capstone_exploration.ipynb` file, the final dataset looks like the following:

	price	weekday	day	week	hour	title_len	state_city	seq_title
0	6.685848	4	17	11	16	9	1038	[41, 1, 57, 59, 18, 4, 159, 224]
1	6.685848	4	17	11	16	9	1038	[41, 1, 57, 59, 18, 4, 159, 224]
2	6.685848	4	17	11	18	10	1038	[41, 1, 57, 59, 18, 4, 159, 940]
3	6.685848	4	17	11	20	9	1038	[41, 1, 57, 59, 18, 4, 159, 224]
4	6.685848	4	17	11	23	10	1038	[41, 1, 57, 59, 18, 4, 159, 940]

Table 2 - Processed dataset

The following transformations have been applied:

Feature creation

A date-time feature contains a lot of information that can be difficult for a model to take advantage of in its native form, that it is why it is decomposed into constituent parts that may allow models to discover and exploit relationships. Using this approach, the following features are created from the original “created_at”:

- `'weekday'`: the day of the week as an integer, where Monday is 0 and Sunday is 6.
- `'month'`: between 1 and 12 inclusive.
- `'day'`: between 1 and the number of days in the given month of the given year.
- `'week'`: the week ordinal of the year.
- `'hour'`: the hour of the datetime.

After a correlation analysis, the feature "`month`" was eliminated because it was considered that it does not add value to the prediction.

Also, a `title_len` feature is created from the length of the `title` feature.

Feature aggregation and label encoding

The `state` and `city` features are combined to form a single `state_city` feature that combines the localization information into a single label. After the aggregation step, each unique category value is assigned an integer value through Label Encoding that simply converts each value in a column into a number.

Text processing

For the `title` feature, that is supposed to contain the most valuable information to predict the `price`, the text data must be encoded as numbers to be used as input for the neural network model.
Keras library provides the Tokenizer class for preparing text documents for deep learning. Once the Tokenizer has been fit on training data, it can be used to encode documents in the train or test datasets. This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf.

Transforming Skewed Data

A large portion of the field of statistics is concerned with methods that assume a Gaussian distribution, but in this case, the `price` feature (to be predicted) distribution is positively skewed. It is recommended to fix the skewness to make good decisions by the model and in this case, the skewness is fixed performing a log transform of the data.

Feature extraction

Once the analysis is completed and the functions are fixed, a reduction in dimensionality is made by eliminating all highly correlated functions and all functions that are not available to a user who is about to publish a new product. This is the case of the `pos` and `sqty` features that are related to actual sales and their corresponding search, but are not available to a seller.

Implementation

Using the Keras Deep Learning library a Neural Network model is constructed and trained to be able to predict the price of a product. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation.

Define Model

Models in Keras are defined as a sequence of layers. The final model defined is the result of trying different combinations of layers to overcome the MSLE score of the benchmark models.
The layers used in the model are:

- **InputLayer**: Layer to be used as an entry point to instantiate a Keras tensor.
- **Embedding**: Turns positive integers (indexes) into dense vectors of fixed size. [9]
- **Flatten**: Flattens the input. Does not affect the batch size.
- **Conv1D**: Creates a convolution kernel that is convolved with the layer input over a single dimension to produce a tensor of outputs.
- **GlobalMaxPooling1D**: Global max pooling operation for temporal data.
- **concatenate**: Concatenates a list of tensors alongside the specified axis.
- **BatchNormalization**: Normalize the activations of the previous layer at each batch.
- **Dense**: Regular densely-connected NN layer.
- **Dropout**: Applies Dropout to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

The last layers of the neural network model are shown in the following image. The complete model can be found in the file `model_plot.png`.

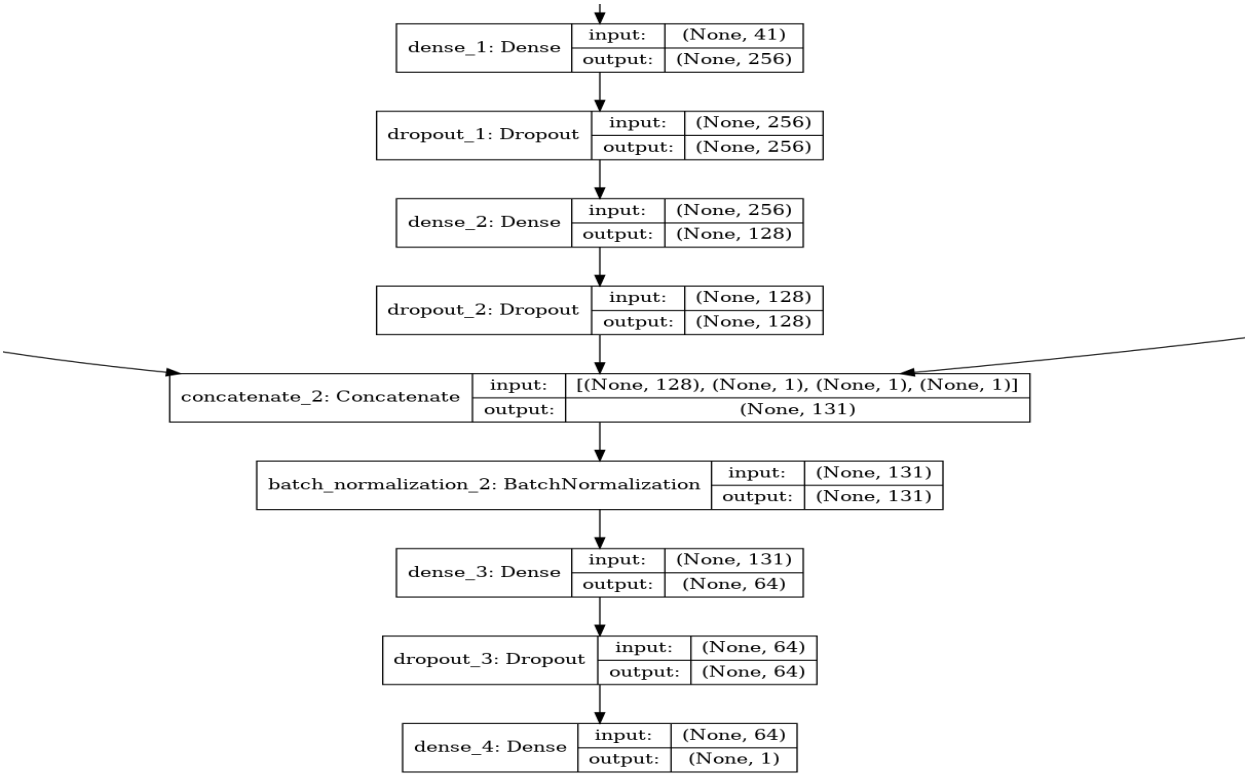

Figure 4 - Last layers of the neural network model

Some settings and parameters that resulted critical and has proven to be useful to get successful results were the dropout rates, the activation layers and the sequence of hidden layers. The activation layer with best results was the Rectified Linear Units (`relu`) and for the dropout layers, the network showed the best results with `0.6` ,`0.5` and `0.3` rates.

Compile Model

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on the hardware.
When compiling, we must specify some additional properties required when training the network. Training a network means finding the best set of weights to make predictions for this problem. We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training.
In this case, we will use `mean_squared_logarithmic_error` loss and the efficient gradient descent algorithm `adam`.

Fit & Evaluate Model

Finally, we have defined our model and compiled it ready for efficient computation. The next step is to execute the model on some data.
The training process will run for a fixed number of iterations through the dataset called epochs, specified using the `epochs` argument. We can also set the number of instances that are evaluated before a weight update in the network is performed, called the batch size and set using the `batch_size` argument. For this problem, the following arguments were defined:

- BATCH_SIZE = 100
- EPOCHS = 5

Refinement

The refinement of the model was a difficult and complex process of choosing different layers, testing different sequences and a deep investigation of the models and techniques used in similar cases. Different layer combinations and sequences the model was tested until the test results were better than the benchmark models. The two techniques that resulted in large improvements for the model were the use of a `BatchNormalization` and the addition of some key numerical entries before the last `Dense` layer.

The first attempts, using only 2 `Dense` hidden layers, without `BatchNormalization` and without the addition of the numerical entries before the last layer, resulted in a poor model with scores around 1.137. There was a slight improvement after adding a third `Dense` hidden layer, where the score increased to 0.609. The addition of the `BatchNormalization` layers resulted in a great improvement, with scores around 0.218. The final improvement was the addition of a `concatenated` layer with all the numerical characteristics just before the last dense layer, which resulted in the final score.

Model	Score
First models: (2) `Dense` + (2) `Dropout` + `Embedding` + `Conv1D` + `GlobalMaxPooling1D`	1.137
+ Adding a `Dense` layer	0.609
+ Adding a `BatchNormalization` layer	0.218
+ Adding a `concatenate` with numerical values (Final)	0.109

Table 3 - Score summary for refinement

IV. Results

Model Evaluation and Validation

After evaluating different layer sequences and adjusting the parameters, the final model resulted in a robust solution to predict prices with an acceptable score. The refinement process, shown in the previous section, showed the benefits of adding different layers and how the model responded in terms of score. It is also important to notice that after some epochs the model stopped the improvement with respect to the MSLE score.

To verify the robustness of the model, a K-Fold cross validation was performed with the following results:

Fold	Score	Fold	Score
0	0.114	5	0.166
1	0.111	6	0.130
2	0.115	7	0.115
3	0.107	8	0.129
4	0.120	9	0.114

Table 4 - K-Fold cross validation results

Justification

The benchmark process provided a baseline on performance, giving an idea of how well the final model actually performed on the problem. These were the benchmark results:

- Benchmark I - **Persistence Model** score = **2.220**
- Benchmark II - **Xgboost model** score = **0.244**

After an iterative process of adjusting the sequence of layers, the neural network began to overcome the Xgboost model, until obtaining less than half the score.:

- **Final score - Keras model = 0.109**

V. Conclusion

Free-Form Visualization

It is possible to see the training history for the neural network model and verify the improvements over each epoch:

```
Train on 585566 samples, validate on 5915 samples
    Epoch 1/5
        585566/585566 [==============================] - 183s 312us/step - loss: 0.0442 - val_loss: 0.0075
        Epoch 00001: val_loss improved from inf to 0.00749, saving model to weights.best.hdf5
    Epoch 2/5
        585566/585566 [==============================] - 180s 308us/step - loss: 0.0112 - val_loss: 0.0048
        Epoch 00002: val_loss improved from 0.00749 to 0.00484, saving model to weights.best.hdf5
    Epoch 3/5
        585566/585566 [==============================] - 180s 308us/step - loss: 0.0079 - val_loss: 0.0050
        Epoch 00003: val_loss did not improve from 0.00484
    Epoch 4/5
        585566/585566 [==============================] - 180s 307us/step - loss: 0.0063 - val_loss: 0.0048
        Epoch 00004: val_loss improved from 0.00484 to 0.00479, saving model to weights.best.hdf5
    Epoch 5/5
        585566/585566 [==============================] - 180s 307us/step - loss: 0.0054 - val_loss: 0.0040
        Epoch 00005: val_loss improved from 0.00479 to 0.00405, saving model to weights.best.hdf5
```

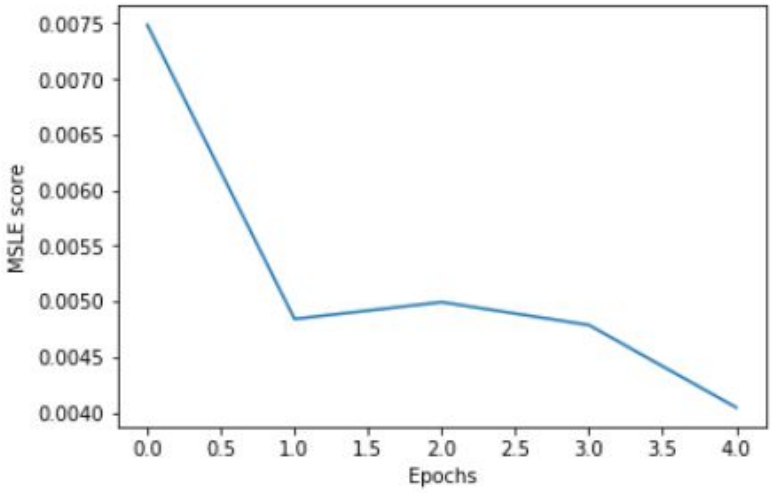

Figure 5 - MSLE score for each epoch

Reflection

Optimal pricing is as much art as it is science, and it can mean the difference between an avalanche of orders flowing out or a mountain of excess stock sitting somewhere. Pricing products and services online is one of the most exciting and complex exercises and, as an online business, finding the right formula is one of the most important questions to solve.

The model created and tested for this project works great to predict the price for new products to be published but it is also a great starting point for other pricing services as price optimization and dynamic pricing. The main difference is that dynamic pricing is a particular pricing strategy, while price optimization can use any kind of pricing strategy to reach its goals. [10]

Future improvement

Additional improvements can be added to this work to improve the accuracy of price prediction. Two of the most important to consider are:

Description feature

For this project only the `title` field was collected from the oficial API, but with the addition of the `description` field, additional information of the product that may be present that results in better predictions.

Currency exchange

Most of the products present in the data set are imported and, therefore, prices are heavily influenced by the change of currency from Argentine pesos to US dollars (ARS to USD). By adding an exchange feature, better predictions can be obtained during seasons with high inflation rates.

VI. References

[1] scikit-learn: Mean squared logarithmic error
[2] Mercado Libre, Inc. (https://www.mercadolibre.com.ar/)
[3] Mercado Libre API. (https://developers.mercadolibre.com.ar/)
[4] The data science blog. A Quick Introduction to Neural Networks.
[5] Chatbots Magazine. Machine Learning, Neural Networks and Algorithms.
[6] Machine Learning Mastery. How to Make Baseline Predictions for Time Series Forecasting.
[7] XGboost developers. XGBoost Documentation.
[8] Udacity Interview with Ben Hammer.
[9] Machine Learning Mastery. How to Use Word Embedding Layers for Deep Learning with Keras.
[10] https://tryolabs.com/blog/price-optimization-machine-learning/