Product Price Recommendation Using Product Descriptions

Contents

Section 01 – Introduction 2

1. Introduction 2
2. Problem statement 2
3. Business Value 2

Section 02 – Project Lifecycle 3

1. Exploratory data analysis 3
   a. Preliminary analysis 3
   b. Price column 3
   c. Shipping column 4
   d. Categories column 5
   e. Item condition column 6
   f. Brand column 6
   g. Item description 7
2. Data Pre-Processing 7
   a. Treat missing values 7
   b. Text pre-processing 8
   c. Consolidate all features to a sparse matrix 8
4. Model development 9

Section 01 – Introduction

1. INTRODUCTION

Mercari, Inc. is a Japanese e-commerce company founded in February 2013 and currently operating in Japan and the United States.

Their main product, the Mercari marketplace app, which allows users to buy and sell items quickly from their smartphones. This app is known for its ease of use and unique shipping system, which allows users to ship items anonymously from local convenience stores through agreements with various shipment vendors. The app has become Japan's largest community-powered marketplace with over JPY 10 billion in transactions carried out on the platform each month.

Product pricing gets even harder at scale, considering just how many products are sold online. Clothing has strong seasonal pricing trends and is heavily influenced by brand names, while electronics have fluctuating prices based on product specs. Our goal is to predict the sale price of a listing based on user provided information for the listing. We have used various Data Mining techniques to build the required algorithm.

2. PROBLEM STATEMENT

To build an algorithm that automatically suggests the right product prices based on the user provided textual product descriptions, and other details like product category, brand name, and item condition.

The dataset is part of the Mercari Price Suggestion Challenge on Kaggle (https://www.kaggle.com/c/mercari-price-suggestion-challenge).

3. BUSINESS VALUE

This algorithm can help sellers judge the prices of their products and list the “best” possible selling price to increase profits

Section 02 – Project Lifecycle

1. EXPLORATORY DATA ANALYSIS

a. Preliminary analysis

We start by looking at the first few rows of the dataset and by looking at the column types

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

Figure 1 First 5 rows of the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1482535 entries, 0 to 1482534
Data columns (total 8 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   train_id           1482535 non-null  int64
 1   name               1482535 non-null  object
 2   item_condition_id  1482535 non-null  int64
 3   category_name      1476208 non-null  object
 4   brand_name         849853 non-null   object
 5   price              1482535 non-null  float64
 6   shipping           1482535 non-null  int64
 7   item_description   1482531 non-null  object
dtypes: float64(1), int64(3), object(4)
memory usage: 90.5+ MB
```

Figure 2 Column types

b. Price column

Next, we look at our response variable, that is '***price***'

```
count    1.482535e+06
mean     2.673752e+01
std      3.858607e+01
min      0.000000e+00
25%      1.000000e+01
50%      1.700000e+01
75%      2.900000e+01
max      2.009000e+03
Name: price, dtype: float64
```

Figure 3 Distribution of values in price column

We can notice that the price of items is right skewed, vast majority of the items are priced in the 10-20 range. However, the most expensive item is priced at 2009. So, we will take a log transformation of price.

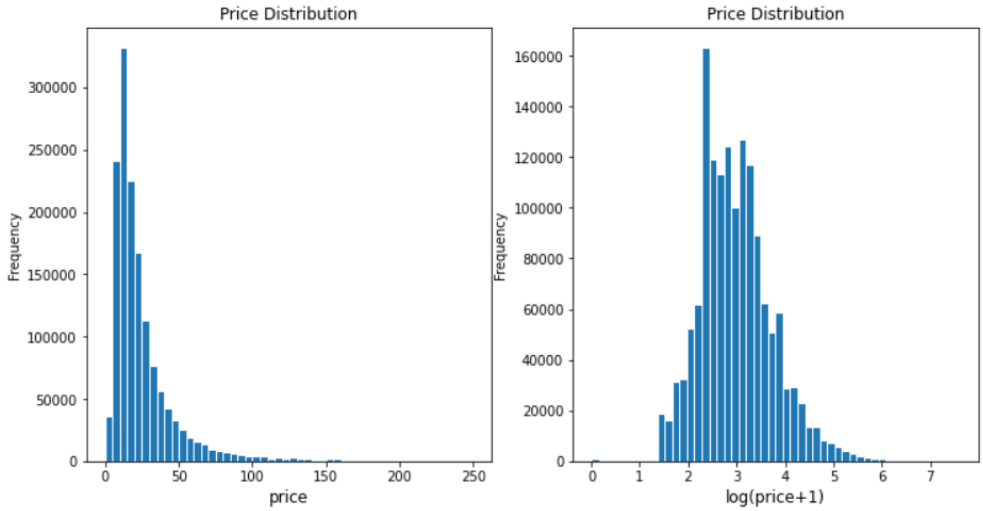

Figure 4 Histogram of price and log(price+1)

c. Shipping column

```
0    0.552726
1    0.447274
Name: shipping, dtype: float64
```

We can observe that for over 55% of items, shipping fee was paid by the buyers.

Let us check how shipping is related to price

Figure 5 Price distribution by shipping type

It is obvious that the price is slightly higher if buyer pays for shipping.

d. Categories column

These are the top 10 most common category names:

```
Women/Athletic Apparel/Pants, Tights, Leggings                60177
Women/Tops & Blouses/T-Shirts                                 46380
Beauty/Makeup/Face                                            34335
Beauty/Makeup/Lips                                            29910
Electronics/Video Games & Consoles/Games                      26557
Beauty/Makeup/Eyes                                            25215
Electronics/Cell Phones & Accessories/Cases, Covers & Skins   24676
Women/Underwear/Bras                                          21274
Women/Tops & Blouses/Blouse                                   20284
Women/Tops & Blouses/Tank, Cami                               20284
Name: category_name, dtype: int64
```

As a next step, we segregate the categories column into three separate sub-category columns and look at how average price per category varies

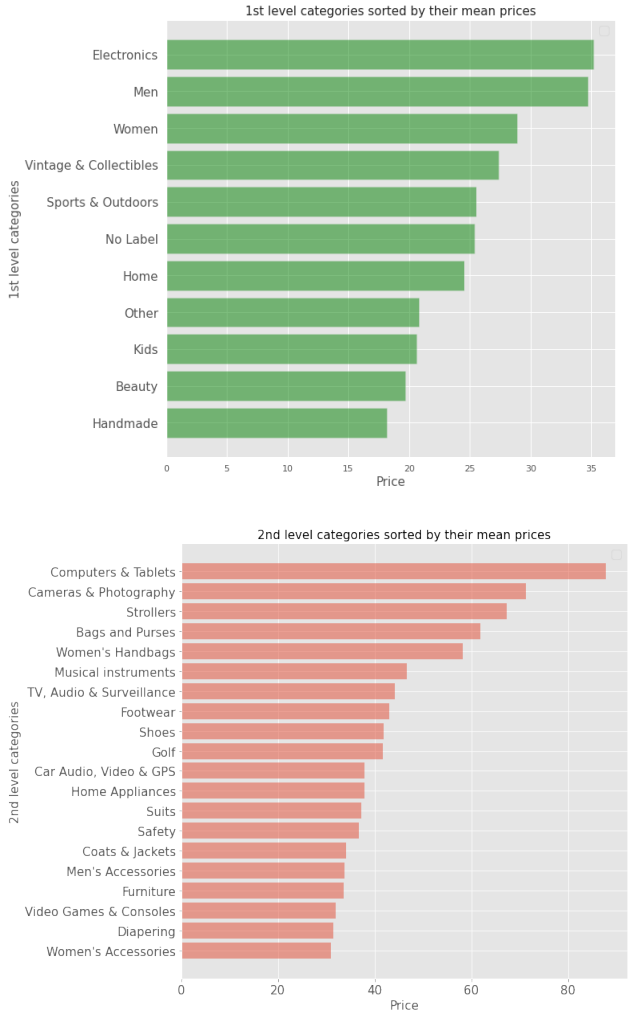

Figure 6 Average price by product sub-categories

We can observe that the average product price varies a lot with the product sub-categories. Product categories should be a key predictor in our model for predicting price.

e. Item condition column

Let us look at how price varies with item condition. We hypothesize that a products condition should impact its selling price posted on the Mercari platform

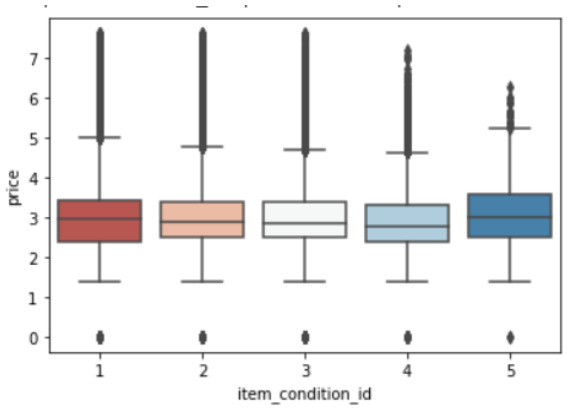

Figure 7 Distribution of price with item condition

f. Brand column

Let us look at frequency of different brands in data using wordcloud.

PINK	56687
Nike	56491
Victoria's Secret	49519
LuLaRoe	38572
Apple	19155
Nintendo	15792
FOREVER 21	15193
Michael Kors	14971
Lululemon	14922
Rae Dunn	14664

Name: brand_name, dtype: int64

g. Item description

We tried to observe the frequency of words in description through word cloud

2. DATA PRE-PROCESSING

Our dataset had many columns with text data and a lot of missing values as well. We have conducted pre-processing to achieve the following objectives:

1. Treat missing values
2. Convert text data into a form suitable for modeling
3. Consolidate all features to a sparse matrix

a. Treat missing values

Below is an output from our code highlighting the number of null values in our dataset:

```
There are 6327 items that do not have a category name.
There are 632682 items that do not have a brand name.
There are 4 items that do not have a description.
```

Figure 8 Number of missing values

We took the following steps for treating missing values:

1. For brand_name column: We computed ngrams (1, 2, 3, 4) on the product names for each missing brand name value. We then compare these ngrams with the list of all brand names and replace the missing brand if there exists a match.

2. For category_name and item_description we have replace null values with the string 'missing'

b. Text pre-processing

1. Converting categorical features to numbers: One-hot encoding

One-hot encoding is a scheme of vectorization where each category in a categorical variable is converted into a vector of length equal to the number of data points. The vector contains a value of 1 against each data point that belongs to the category corresponding to the vector and contains 0 otherwise

We have converted the 'brand_name' column to its one-hot encoded vector using the LabelBinarizer function. Here is a snippet of the code:

```
lb = LabelBinarizer(sparse_output=True)
X_brand = lb.fit_transform(merge['brand_name'])
```

2. Converting text features to numbers: TF-IDF and Count vectorizer

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. We have encoded item_description into TF-IDF vectors of uni-grams, bi-grams and tri-grams.

Additionally, we have also used Scikit-learn's CountVectorizer. CountVectorizer is used to transform a corpus of text to a vector of term/token counts. We have used CountVectroizer to transform 'name' and 'category_name' columns

c. Consolidate all features to a sparse matrix

Matrices that contain mostly zero values are called sparse, distinct from matrices where most of the values are non-zero, called dense. It is computationally expensive to represent and work with sparse matrices as though they are dense, and much improvement in performance can be achieved by using representations and operations that specifically handle the matrix sparsity.

We stack dense feature matrices with categorical and text vectors and then convert the dense matrix into a sparse matrix using Python's scipy library. We finally feed our model an input matrix, which contains all the features that we have extracted in the previous section, and an array of corresponding target values.

4. MODEL DEVELOPMENT

a. Choosing Model

Just based on the dataset we can observe that the price prediction of the products would largely be dependent on the Category--> Sub-category-->Brand name-->description, etc.

This is very similar to how a tree model works. Hence, for regression tree we will be using Light Gradient Boosting Model (LightGBM regression).

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks. Since it is based on decision tree algorithms, it splits the tree leaf-wise with the best fit whereas other boosting algorithms split the tree depth-wise. So, when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the depth-wise algorithm and hence results in much better accuracy. Also, it is surprisingly very fast, hence the word 'Light'.

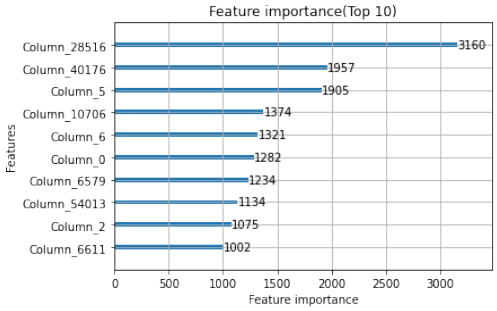

Figure: Top 10 Features

b. Using 3-fold Cross-validation for hyperparameter tuning

To find the best model paraamters, we have implemented grid search using 3-fold cross-validation (using the RandomizedSearchCV function in Python).

We subsampled 90% of our training data to prevent the trees from overfitting. Subsampling also forces the trees to make use of the features more evenly, even if they're not as strong. Having weak contributions from many features is often more robust than having strong contributions from few features.

We then defined a grid for the following tree parameters:

1. Learning rate - determines the impact of each tree on the outcome
2. Number of iterations – High number of iterations result in higher accuracy

3. Number of leaves - To control the complexity of the tree model by limiting the minimum number of leaves
4. Maximum depth – To limit the maximum depth of a tree model
5. Minimum child weight - stops splitting after sample size goes below this value

PARAMETERS	DEFAULT VALUE	TUNED VALUE
Learning Rate	0.75(Default value-0.1) *	0.24102
Number of Iterations	100	975
Number of Leaves	31	194
Maximum depth	3(Default value-infinite) *	13
Minimum Child weight	0.001	1.219
RMSE	0.4598	0.4111

*Updated the learning rate and maximum depth to accelerate model training

The above table provides us with the new tuned parameter values. The tuning resulted in the improvement of our RMSE on the test data.

Section 03 – Results

Our Final model is a Gradient Boosted Regression Tree Model having a learning rate of 0.24 with a **RMSE (on the test dataset) of 0.4111**.

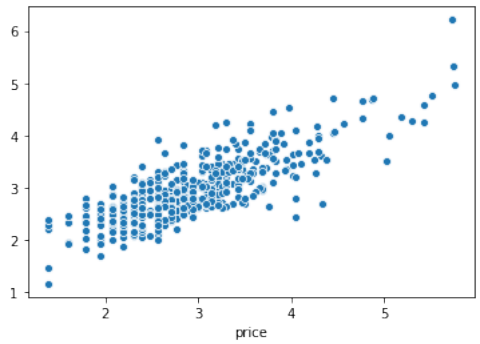

Figure 9:Scatter plot of Actual price vs Predicted Price(500 values)

Note: For Benchmarking purposes, the winning model at Kaggle for this competition's RMSE was 0.37.

Section 04 – Future Work

- Using deep learning was productive and yielded a very good score on test data. More complex models such as MLP, LSTMs, Convolutional Neural Nets can be tried.
- Other vectorization schemes such as Wordbatch, word2vec can be experimented with ML models.
- Regression models like Ridge, FTRL and FM_FTRL can also be tried.

Section 05 – Conclusion

Based on the above report and results, we can state that the Light GBM model performed up to our satisfaction.

We learnt that the data preparation part is more challenging than any other part of the modeling process. It was a very challenging project for us, and we enjoyed doing it thoroughly.

Unfortunately, we could not provide the list of best features and the final plot of the tree due to the large number of the variables in which we divided each feature. We are still researching on how we can interpret such a model so that we can extract some business insights from it.

Section 06 – References

- https://www.kaggle.com/c/mercari-price-suggestion-challenge
- https://www.kaggle.com/ceshine/fork-second-place-solution