Mercari Price Suggestion

(Team Name: MLiens)

Chetan Gulecha
IIITB (MT2020052)
Chetan.Gulecha@iiitb.org

Pratik Pawar
IIITB (MT2020121)
Pratik.Pawar@iiitb.org

Ankit Sahoo
IIITB (iMT2018502)
ankitkumar.sahoo@iiitb.org

Abstract *- **Mercari is Japan's biggest community-powered e-commerce company. The company has developed a e-commerce market application 'Mercari' that allows individuals to easily and safely buy and sell goods. Mercari's challenge is to build an algorithm that automatically suggests the right product prices to sellers on its app. Predicting the price of a product is a tough challenge since very similar products having minute differences such as different brand names, item condition, category demand of the product, etc. Using the data given in competition, we have to come up with a solution that predicts the price of a product listed on Mercari as accurately as possible.***

Index Terms— Exploratory Data Analysis, normalization, Feature engineering, Tfldf vectorizer, Grid search, Train-Test split for cross-validation, Important feature extraction, Bag of words, One hot encoding, stopwords, Ridge regression, LGBM, SVR, Model combination, stacking, RMSLE

I. INTRODUCTION

Artificial Intelligence has become major part of all major e-commerce companies today and machine learning being one of the ways to achieve it is showing great scope. Many of the IT companies have started to explore the ways to automate various activities using state of the art Machine Learning algorithms and Deep Neural Networks.

First, let us explain why price suggestion is important in Mercari platform. On the Mercari app, users can choose whatever price they want when listing an item. Here are some examples which show why this becomes a problem:

-Some items on Mercari cannot be sold because their listing prices are too high compared to the market price.

-Conversely, if the listing price is lower than the market price, customers lose out.

As a solution, users can search Mercari for an item they plan to list, effectively performing market research. However, this is a lot of effort for the user and this idea may not occur to new Mercari users. Therefore, listing becomes easier if we automatically display a suitable price for users when they list an item.

The rest of the paper proceeds as follows: Sec. 2 we have described our dataset and evaluation scheme. In Sec. 3 covers our visualizations, EDA and their inferences. Sec. 4 will discuss about data pre-processing and feature extraction Sec. 5 will cover model - building, various approaches and their output. We conclude in Sec. 6 and outline challenges with our method and possible avenues of future work.

II. DATASET

train.csv has size of (1037774, 8) and test.csv has (444761, 7) size.

train_id or **test_id** - the id of the listing
name - the title of the listing of the item.
item_condition_id - the condition of the items provided by the seller
category_name - category of the listing
brand_name – brand of th item
price - the price that the item was sold for.
shipping - 1 if shipping fee is paid by seller and 0 by buyer
item_description - the full description of the item.

```
Shape of train data:  (1037774, 8)
Shape of test data:  (444761, 7)
```

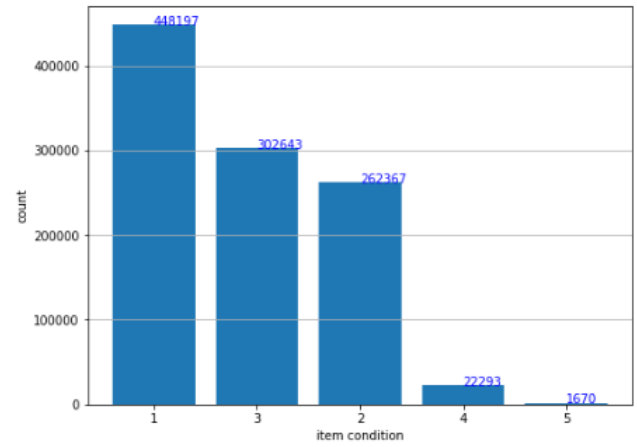
	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	1436222	Tarte rainforest after dark colored clay	1	[Beauty, Makeup, Makeup Palettes]	Tarte	36.0	1	A brand-new, unused, unopened, undamaged item...
1	402989	Mac mineralize skin finish	3	[Beauty, Makeup, Face]	MAC	15.0	1	Color is gold deposit, about 85% of product left
2	638275	White Case Phone 6/6s	1	[Electronics, Cell Phones & Accessories, Cases...	NaN	3.0	1	New Ultra thin Candy TPU Silicone Rubber Soft
3	1113629	Victorias Secret push-up plunge	3	[Women, Underwear, Bras]	Victoria's Secret	18.0	1	VS sexy little thing multi way bras both size ...
4	328823	Disney Princess Toddler Boots Size 10	2	[Kids, Girls 2T-5T, Shoes]	Disney	13.0	1	New with out box!

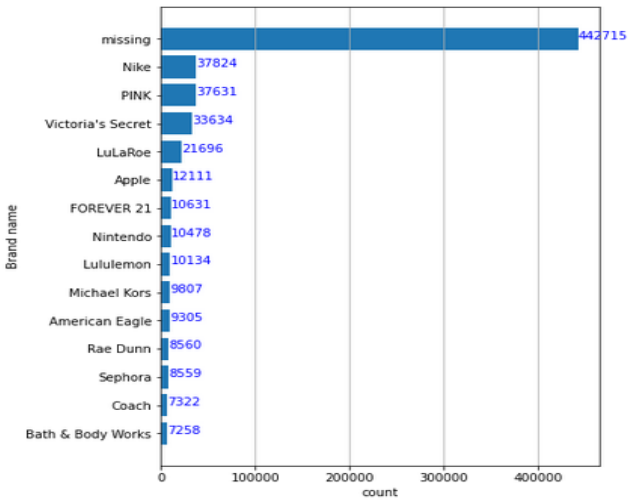
We have to predict the missing column which is 'price' in test dataset. After predicting output value, we should have some way to determine how good or bad our model has done. For this competition in Kaggle they have used RMSLE (Root mean square logarithmic error) score.

III. EXPLANATORY DATA ANALYSIS

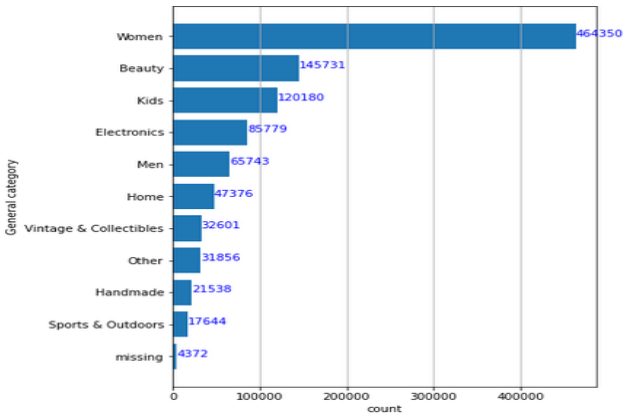
Before getting into pre-processing and feature extraction, it is very important to get to know the distribution of data in order to get better insights while feature selection. We are presenting a few of those here:

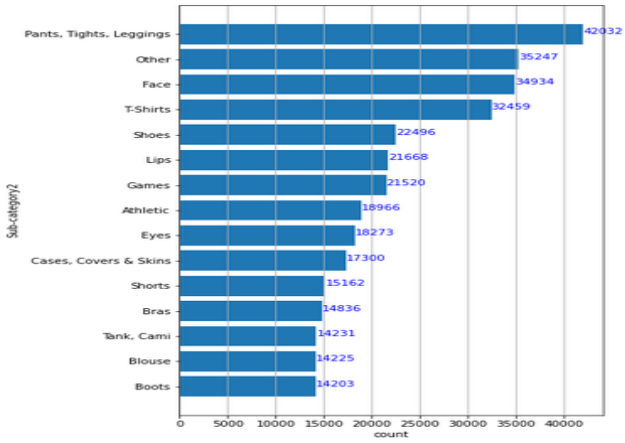
1)Distribution of data over different features.





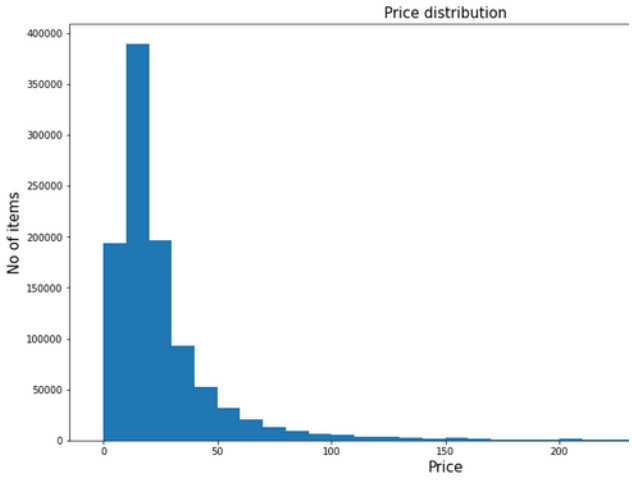
For most of the items brand is null and it is replaced by ‘missing’ value.



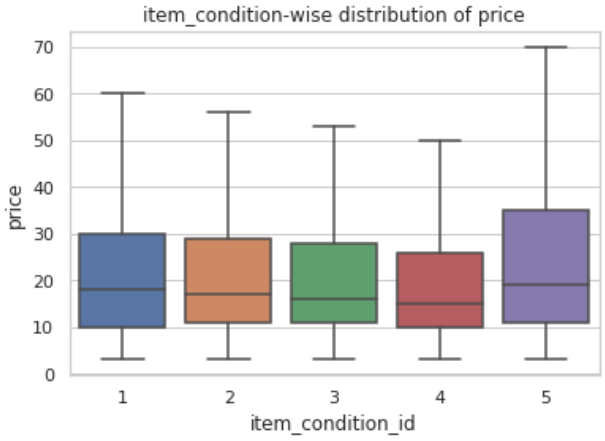


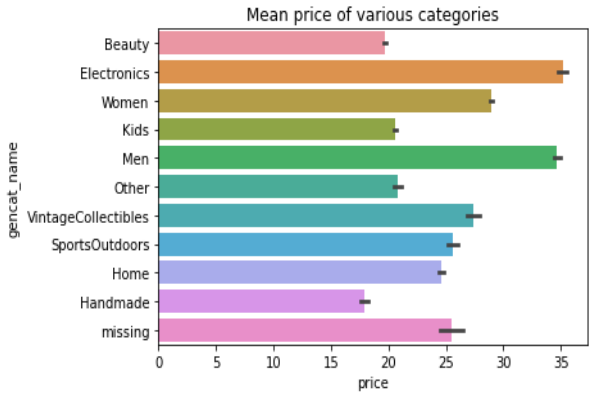
2) Distribution of price over various features

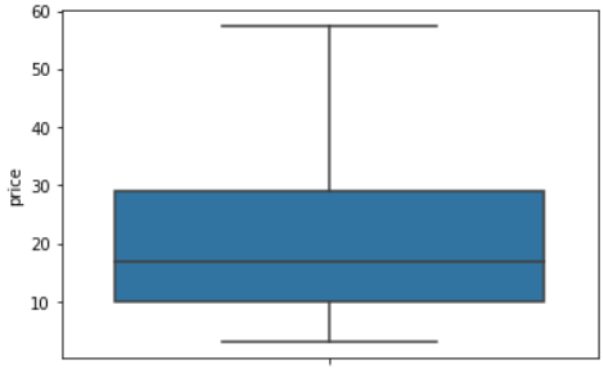
We have removed those rows which have price less than 0.

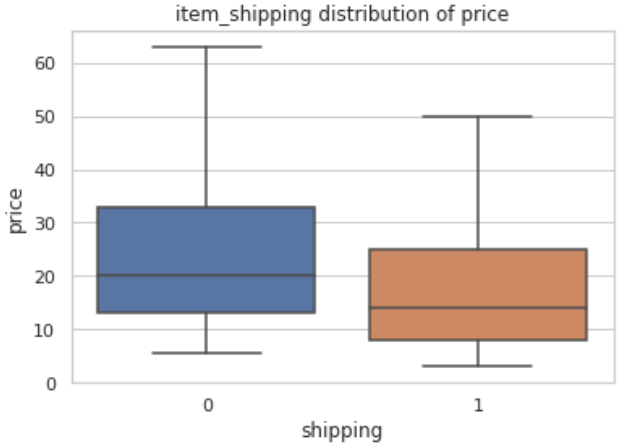


As it is clear from above graph that price is left skewed in our dataset, so we have applied log function over price column to remove the skewness.









As from above graphs it is clear that features like item condition, brand name, general category, subcategory etc. play major role in determining price of item.

IV. DATA PREPROCESSING AND FEATURE EXTRACTION

To make the training better we need to modify the data, which essentially means that we need to convert the raw data we were given to more sophisticated data which would then be ready to undergo subsequent processing. Basically, we need to extract meaning out of raw data before we can use it to train our models. Models trained on meaningful data always have upper hand to those trained on raw/less meaningful data. Hence, pre-processing the data is a very crucial step in training our model. Note that these steps are also done on test dataset as well because we want both train and test feature set and their corresponding values to be in sync while applying them on model.

1) Basic text pre-processing

a. Remove stop words from name and description column.
b. Convert text columns to lower case.
c. Replace non alphanumeric characters from string with space.

2) Add feature engineered columns

After doing some trial and error we added features like:

Length of item name,
Length of description,
Number of words in name
Number of words in description etc.

As we know features like brand, general category, subcategory etc. plays vital role in determining price of item. We have added below features:

Brand mean price
General category mean price
Subcategory1 mean price
Subcategory2 mean price
Item_Condition mean price
Brand mean price
General category median price
Subcategory1 median price
Subcategory2 median price
Item_Condition median price
Brand median price
General category mean price
Subcategory1 mean price
Subcategory2 mean price
Item_Condition mean price
Concatenated brand, name and description

These features are computed by grouping the values in those corresponding features. For test dataset we have mapped those values from corresponding train dataset as test dataset doesn't have price column. This is known as test dataset transformation. These added features improved performance of individual models.

3) Apply Bag of Words for text features

Models don't understand text values while computation. So, we have converted our text features like brand name, general category, subcategory1 and subcategory2 to numeric values using one-hot encoding.

4) Conversion of name, description and concatenated column to vector using TF-IDF Vectorizer

There are different methods for converting text string to vector like word2Vec, TfIDF, counterizer (0/1) or numerical etc.
Each of above method has their pros and cons.

Why have we used TF-IDF vectorizer?

From problem statement we can infer that what we have to predict price based on description and name given by user. Intuitively we can guess that what we have to do is not that much related to semantic meaning of sentence (in which case word2vec would be better choice). We simply have to check the importance/presence of the word in determining price. TFIDF performs better in such scenarios. In this method we have to convert words in corpus as feature and assign them value mentioned by algorithm below. TFIDF is computationally better than word2Vec as in word2Vec each word is determined by long vector representing its's value and relationship with other words, while in TF-IDF it is in our hand up to how many features we want to choose.

About TF-IDF Vectorizer: It is a numerical statistic that is intended to reflect the importance of a word in a document of a collection or corpus. TF-IDF vectors are calculated in the following way:

a. Calculate term frequency (TF) in each document.
b. Calculate the inverse document frequency (IDF): Take the total number of documents divided by the number of documents containing the word.
c. Iterate each document and count how often each word appears.
d. Calculate TF-IDF: multiply TF and IDF together.

Output is sparse matrix as the feature will get significant value only if it is present in that name or description.

We have encoded name, item_description and concatenated feature into TF-IDF vectors of uni-grams, bi-grams and tri-grams. Note that using 1,2,3-grams together would result in a huge number of words in the dictionary of TF-IDF vectorizer and using all of them would result in very high dimensional vectors. To avoid this, we have limited the number of dimensions to 250k for name and 500k for item_description vectors.

5) Column normalization of numeric features

The primary purpose of normalization is to scale numeric data from different columns down to an equivalent scale so that the model doesn't get skewed due to huge variance in a few columns. We have used min-max normalization here.

Normalized columns- Feature engineered columns like mean, count, median of brand, category, Length of item name, Length of description etc.

6) Horizontal stacking of features created like output of TFIDF vectorizer, one hot encoding output, Boolean features like (price in name, price in description, shipping) and normalized columns.

After all these steps the output is used in various models mentioned below.

V. MODEL SELECTION

After doing all feature engineering, cleaning data and extraction of important features we also added some extra features as mentioned in EDA. We also handled all categorical, text and numerical columns.

We have experimented with following models

1. Linear Regression
2. Lasso Regression
3. Ridge Regression
4. Support Vector Regression
5. Light Gradient Boosting Machine
6. Stacking
7. Ensembling

1. **Linear Regression**

Linear Regression model is a basic model used when we want to predict value of dependent variable using other variables. It will find the best line that fits the datapoints.

RMSLE of Linear Regression
Train data :- 0.4812
CV data :- 0.4893
There is no hyperparameter tuning in Linear Regression.
Linear Regression is simple model and to improve score we have tried some other models that perform hyperparameter tuning.

2. **Lasso Regression**

Lasso Regression is a type of linear regression which penalizes the sum of absolute values of the coefficients (L1 penalty). It is used to prevent over-fitting which may result from simple linear regression.

Cost function of Lasso Regression

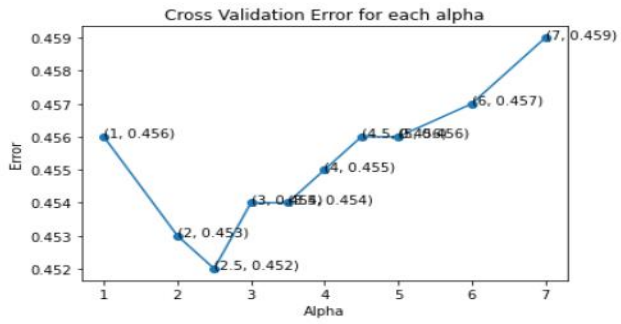
$$\text{Min} (||Y - X(\text{theta})||^2 + \lambda ||\text{theta}||)$$

Lambda is the penalty term. λ given here is denoted as alpha in lasso. After performing grid search, we have got alpha = 1e-06. And we have trained the model with this alpha.

RMSLE of Lasso Regression
Train data :- 0.4742
CV data :- 0.4799

After using lasso, errors are almost similar to linear regression. So, we have ridge regression.

3. **Ridge Regression**



Ridge Regression is a type of linear regression which penalizes sum of squared coefficients (L2 penalty). It is a model tuning method that is used to analyze any data that suffers from multicollinearity.

Cost function of Ridge Regression

Min (||Y – X(theta)|| ^ 2 + λ ||theta|| ^ 2)

Lambda is the penalty term. λ given here is denoted as alpha in ridge function. Ridge has two hyperparameters in it {alpha, solver}. After performing hyperparameter tuning using grid search we have got alpha = 2.5 and solver = 'sag'.

RMSLE of Ridge Regression
Train data :- 0.3572
CV data :- 0.4520

Here, result is slightly better than above lasso and linear.

4. **Support Vector Regression**

To try a new model, we went ahead with SVR. In linear, lasso, ridge the aim is to reduce the error of the test data. In SVR we can define how much error is acceptable in model and it will find best fit line (hyperplane in higher dimensions).

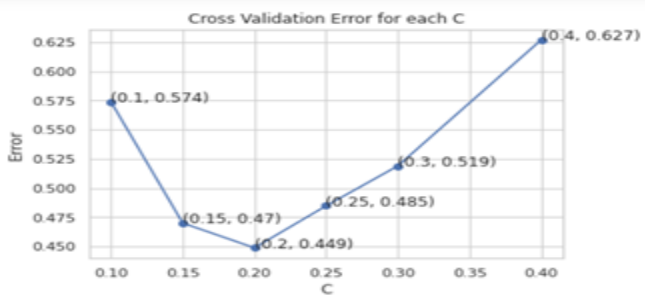
Cost function of SVR

Min (½ || w || ^ 2 + C (i=1 to n)∑ | £ᵢ |)

Constraints

| y_i - $w_i x_i$ | <= ε + | £ᵢ |

Gris search is used to find better C value with epsilon error value 0.1. We have got C = 0.2. Again, we build our model with C = 0. 2 and ε = 0.1.



RMSLE of SVR

Train data :- 0.4085
CV data :- 0.4488

This model improves accuracy.

5. **Light GBM**

We have tried couple of regression models. So, to try a boosting model LGBM. LGBM is a gradient boosting algorithm which uses decision tree and expands in vertical fashion. There are many hyperparameters in LGBM to tune. Following are the generated parameters after hyperparameter tuning:

{
learning_rate=0.5
max_depth=8
n_estimators=500
num_leaves=80,
sub_sample=0.9
colsample_bytree=0.8,
min_child_samples=50
boosting_type='gbdt'
}

RMSLE of LGBM
Train data :- 0.3135
CV data :- 0.4564

LGBM has performed better than all models that we have implemented.

6. **Stacking**

Till now we have implemented multiple regression models. It may happen that these models learn some part of the problem but not all and then predicting. To solve this, we have decided to go with stacking. In stacking we used predictions of ridge, LGBM and SVR. We build a new model which learns from these predictions. And this improved our overall performance a bit.

RMSLE of stacking
Train data :- 0.4533
CV data :- 0.4537

7. **Ensembling (Merging all models)**

Ensembling is technique that combines several base models to produce one optimal model. At first we ensemble our three best models and that are ridge, SVR and LGBM. This ensembling improves our model a lot and we reached to error of 0.43518. Formula that we have used for this ensembling is as follows

Weighted result = 0.4 * (Ridge prediction) + 0.3 * (SVR prediction) + (LGBM prediction)

To move forward, we thought that stacking also gave us better results so why not try ensembling of stacking and above mentioned weighted result. After trying many combinations we have got the best results for following combination.

Final_output = 0.3 * (Stacking prediction) + 0.7 * (Weighted result)
Public error after ensembling is 0.42760.

Model Scores

Algorithm	Public Score	Private Score
Linear Regression	0.48056	0.48017
Lasso Regression	0.47907	0.47879
Ridge Regression	0.44772	0.44720
SVR	0.44788	0.44961
LGBM	0.45703	0.45611
Stacking	0.45068	0.45033
Ensembling(Weighted Merging)	0.43496	0.43518
Final Result	0.42760	**0.42778**

VI. Conclusion

It was fun as well as a great learning experience doing this project. We would like to conclude that we were able to come up with an efficient model to predict the price of the item. We got private score of 0.42778 which is pretty good using classical models only.

Challenges and Future scope

Predicting the price of a product is a tough challenge since very similar products having minute differences such as different brand names, additional specifications, quality, demand of the product, etc. can have very different prices. Another thing is that we have to properly use name and description given to enhance our performance of models. There were lot of things which have to perform on basis of trial and error like getting best possible weights for combining models, feature engineering etc. Going further we can explore different vectorization schemes like 'wordBatch' etc. Regression models like FTRL and FM-FTRL can also be tried. We can experiment with neural network models, those were not allowed during competition.

Acknowledgment

We would like to thank Professor G. Srinivas Raghavan , Dr. Neelam Sinha, our mentor Amitesh Anand and all Machine Learning Teaching Assistants for giving us the insights in ML field and help us whenever we were struck by giving us ideas and resources to learn from. We would also like to thank team SuperSaiyans, three musketeers for being a great competitor and setting a benchmark time by time for the rest of us which acted as a driving fuel for us to constantly work hard and surpass them.

References

I. https://www.mygreatlearning.com/blog/what-is-ridge-regression/
II. https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2
III. https://lightgbm.readthedocs.io/en/latest/Parameters.html
IV. https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
V. https://www.geeksforgeeks.org/stacking-in-machine-learning/
VI. https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
VII. https://www.kaggle.com/c/mercari-price-suggestion-challenge
VIII. https://engineering.mercari.com/en/blog/entry/2018-12-05-183000/