# Mercari Price Suggestion

Shanthan Reddy. G
IMT2018522
Shanthan.Reddy@iiitb.org

Madhav. P
IMT2018524
sai.madhav@iiitb.org

Sriman Reddy. P
IMT2018525
sriman.reddy@iiitb.org

***Abstract*—Through this project, we are presenting a model that can automatically suggest the right product prices based on the product's given features. We have used Regression models such as Ridge regression LGBM Regressor and Linear Regression. The main goal of our project is to find the most reasonable price for a given product.**

***Index Terms*—Linear Regression, Ridge Regression, Feature Engineering, Label encoding, One-hot encoding, LGBMClassifier,XGBoost, Ensembling, TF-IDF, Bag of words, Sentimental Analysis**

## I. Introduction

In the modern world, people are more inclined towards purchasing products online rather than shopping in-person. Millions of products are available for purchase on online shopping websites. A major issue in online shopping is the pricing of products. Giving a price to each one of them manually is very slow and expensive. Hence this process is automated. Since the prices are automatically given, it is important to consider all the features and properties of a product to provide an accurate price. A highly accurate price predicting model can give any seller a great advantage in the competitive market. This is the motivation behind our project.

## II. DATASET

For this project, we have used the Kaggle dataset provided by Mercari for Mercari Price Suggestion challenge. There are over 1.5 million products in both test and train data combined. Each product has its own id as a label. Each product has a *name, item_condition_id, category_name, brand_name, shipping, item_description and price*.

**Feature description:**

- ***name:*** Name of the product (object)
- ***item_condition_id:*** The status or condition of the item. The condition is described through a number from 1 to 5. (int64)
- ***category_name:*** Category the item belongs to. There are 3 different categories in hierarchical order. (object)
- ***Brand_name:*** Brand the item belongs to. (object)
- ***shipping:*** Determines if shipping cost was involved. (int64)
- ***Item_description:*** The description of the item. (object)
- ***price:*** The price the item was sold for. This is our target variable. (float64)

## III. Exploratory Data Analysis (EDA)

The first step in solving any data science problem is to thoroughly check and analyze the given data. This step will help us a lot in providing insights and hidden patterns from the data. This is known as Exploratory Data Analysis (EDA). A good EDA will help us in identifying and creating new features which can be later used for our model training.

Before we start analysing anything, we must ensure that there are no null values.

### A. Checking for null values

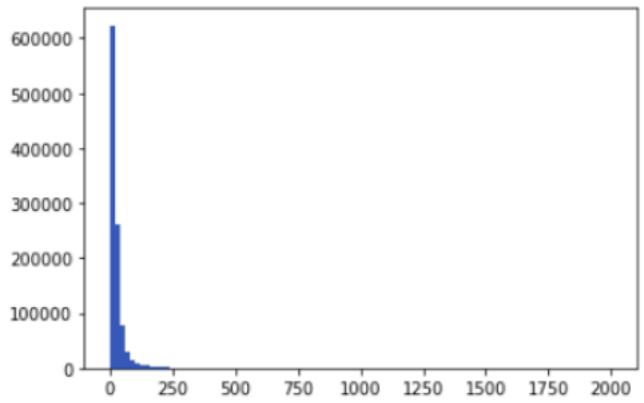
We expected our data has null values. It is also important to check for the percentage of the missing data so we can decide how to fill the missing values.

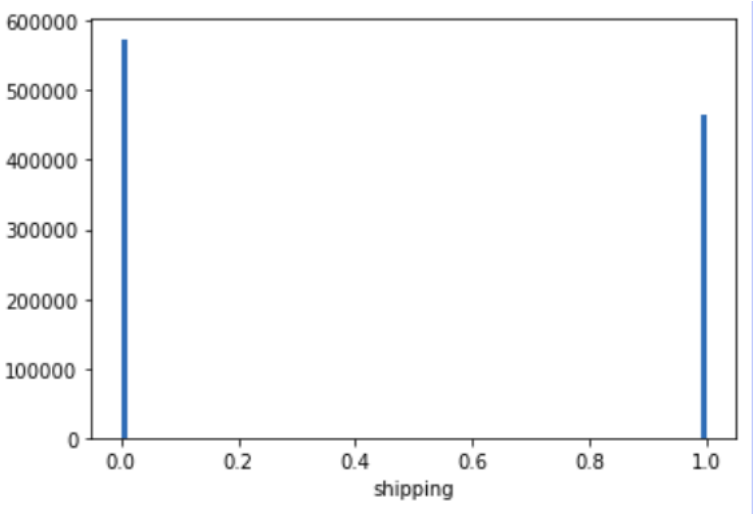
| | Feature | Unique_values | null count | Percentage of missing values | type |
|---|---|---|---|---|---|
| 0 | train_id | 1037774 | 0 | 0.000000 | int64 |
| 1 | name | 877889 | 0 | 0.000000 | object |
| 2 | item_condition_id | 5 | 0 | 0.000000 | int64 |
| 3 | category_name | 1258 | 4382 | 0.422250 | object |
| 4 | brand_name | 4413 | 442947 | 42.682414 | object |
| 5 | price | 760 | 0 | 0.000000 | float64 |
| 6 | shipping | 2 | 0 | 0.000000 | int64 |
| 7 | item_description | 904740 | 2 | 0.000193 | object |

As we can see, *category_name, brand_name and item_description* have null values in their columns. None of them has a very high percentage of missing data hence we need not drop any feature. In the case of *brand_name*, 40% of the data is missing. Hence instead of filling it with mode, we decided to create a new brand name *'missing'*. Also, the NaN values in the remaining two features are replaced with *'missing'*.

### B. Price

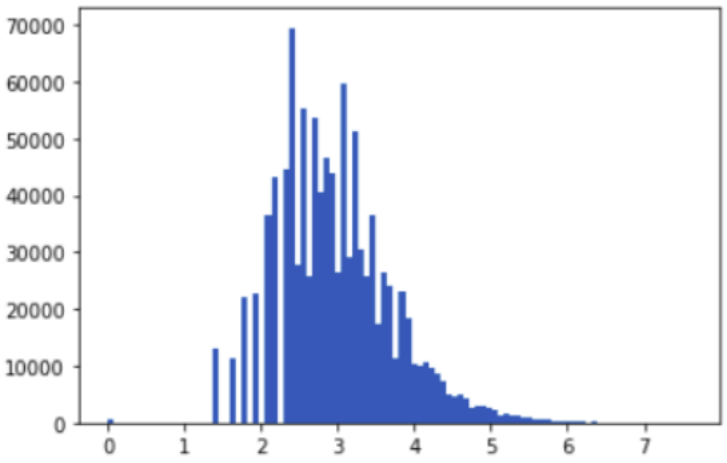
We then tried to check the distribution in the target variable i.e, *price*. We have found that the *price* was highly skewed. I.e, there are very few items with a very high price and the majority of them are in a lower range.



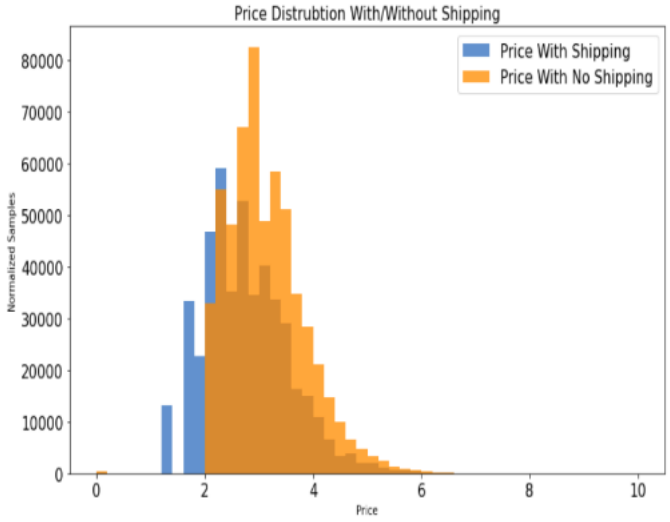




### C. Removing Skew

Skewness can be defined as the degree of distortion from a normal distribution. From the above plot, we can see that the price column is right-skewed. In order to remove right skewness, we used logarithm transformation. After applying this transformation, the skewness got removed and we got the below distribution.




As we can see from the above histogram, the buyers paid the shipping more than the sellers did.



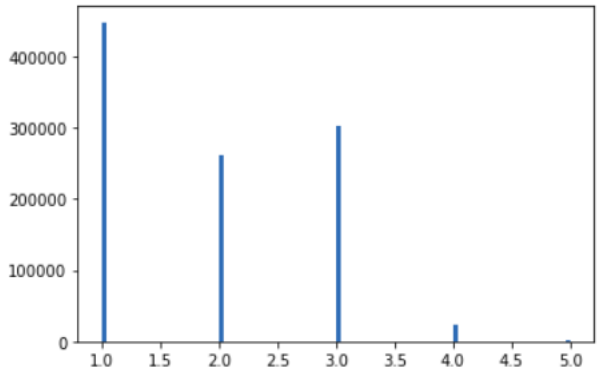

From this graph, we can infer that at higher price ranges, the shipping cost is being excluded. In other words, if the cost of the item is low, the shipping fee is included for profits. This is something we observe too when we buy online products. If the price of the item doesn't cross a fixed price, the shipping fee is incurred automatically.

### D. Shipping

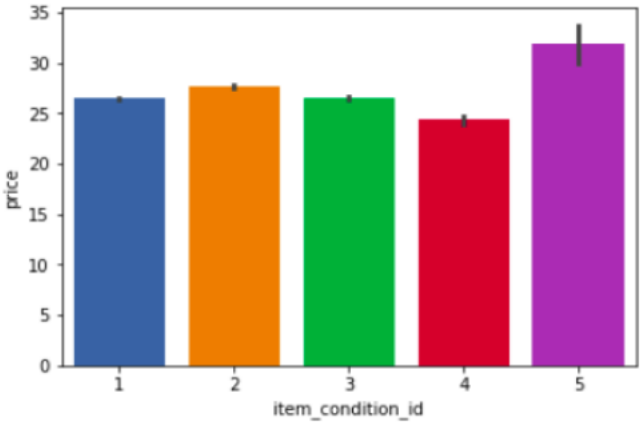
- If *shipping* is 1, it indicates that the shipping cost is not included or the seller paid the shipping fee.
- If the *shipping* is 0, it indicates that the shipping cost is included or the buyer paid the shipping fee.

### E. item_condition_id

This feature indicates the condition of the product. This is scaled from 1-5 where 1 is the least quality and 5 being the best quality.




From the above histogram, we can see that there are very few items with the best quality (4,5) and most of them are in condition 1-3.



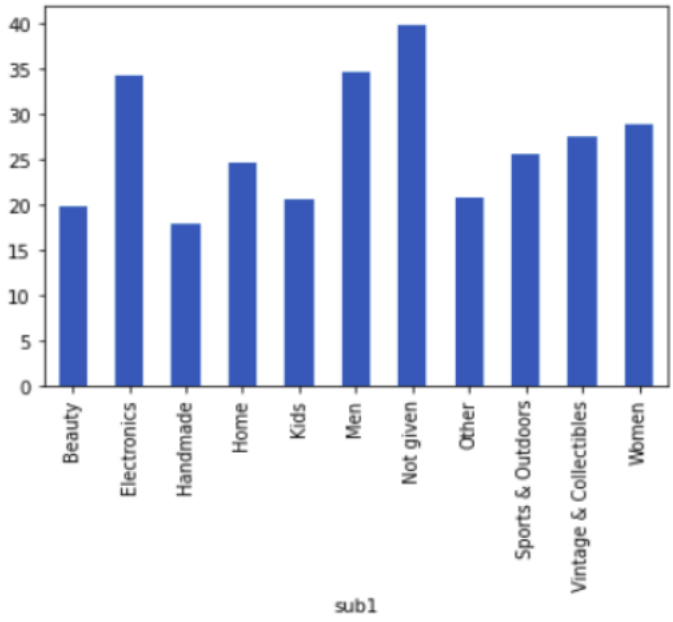

Above is a bar plot between price and item condition. HIgher the *item_condition_id*, better the condition of the product. As we can see, items with higher condition ids are having higher prices.

### *F. Category_name*

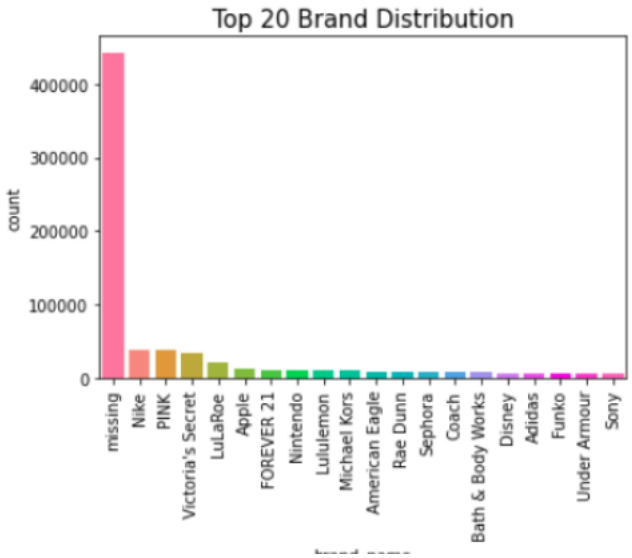
While we were looking through the data, we found that the *category_name* has three different levels of categories in the *category_name*, we have separated them into three different features i.e, sub1, sub2, sub3 which indicate the respective subcategories inside the *category_name*. An example of sub-category 1 is given below.

Sub-category 1:
As we can see, most of the products are either from Men/Women or from Electronics (apart from Not given). This trend extends with the increase in hierarchical depth.

- sub1 has a total of 11 unique values.
- sub2 has a total of 114 unique values
- sub3 has a total of 851 unique values.

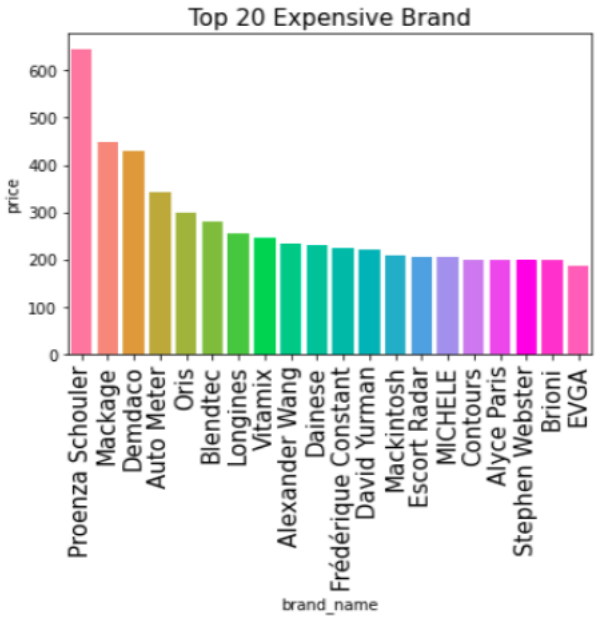



### *G. Brand*




As the data set has 40% of missing values hence the highest total sales for missing is as expected. Apart from them, Nike, PINK and Victoria's Secret are the 3 most common brands in the data.

We classified expensive brands as the mean of the price of their all products Even though Nike, PINK and Victoria's Secret had a lot of frequency, none of them was among the top 20 expensive brands.




## IV. Pre-Processing

Data Preprocessing is the next important step before building the model. It is a process in which we convert the raw data into a clean and formatted data which is suitable for a machine learning model to learn.

**Splitting into Train and Test dataset:** The very first step in pre-processing is splitting the data. We divide the given data into training data and testing data. The training data is the one that is used by our algorithm to learn from.

**Text Pre-processing:**

- First, we started off by decontracting contracted strings. That is, converting strings such as " aren't " to "are not".
- Then we removed unnecessary characters i.e, characters which are neither numeric nor alphabetic. We also removed punctuations.
- Then we replaced \n, \t etc. with empty strings.
- We have also converted all the words to lowercase.

**Stemming:** We then applied stemming on the words. Stemming is an important operation in any NLP related project. Stemming is basically converting or reducing a given word into its root-base form. We have imported *PorterStemmer* from nltk.stem.porter to perform this operation.

**Removal of stop words:** In any text data, we find stop words which are only useful for providing meaning to the sentences(Ex.: 'a', 'an', 'in'). Since they do not contribute anything useful for our regression model, we have to remove stop words.
We imported *stopwords* from *nltk.corpus* to remove these stopwords.


We tried to visualize the *item_description* feature after preprocessing. Hence after implementing all the above preprocessing steps, we used the **Worldcloud** to identify the frequency of words in the descriptions. From the above Wordcloud, we can see that 'brand', 'new', 'free' etc. are some of the most frequently used words (bigger word size indicates higher frequency). This indicates that the sellers are using these words to market their products.
All the above steps are performed on *item_description, name* features.

## V. Feature Extraction

After cleaning and analysing the data, the next and most important step before building our model is Feature engineering.

Feature engineering is the process of using domain knowledge of data to create features that make our models work. To put it simply, Feature engineering is the process where we create new or meaningful features from the existing data. The data given to us consists of categorical, textual and numerical features. But since our model only accepts numerical data, we need to extract the relevant features.

We have used Term-frequency and inverse document frequency (TF-IDF) and Bag of Words for vectorizing our textual data. We used one-hot encoding for categorical data.

### A. Vectorization of categorical features:

**Label Encoding:** One way to encode categorical values is by label encoding. In this process, we will be assigning numbers to respective unique elements of features. We have seen in EDA how different brands vary for different values of evaluation. So we decided to try out label encoding with *sub1, sub2, sub3, brand*. We have implemented encoding

with help of python dictionaries. We have done encoding with respect to mean and median. Median has produced a better result when compared to mean. Accuracies obtained from this are not up to the mark. So we chose to shift from label encoding as mean or median is the best way possible where the better way being one-hot encoding.

**One-Hot Encoding:** After ruling out Label encoding, we decided to use One-hot encoding. One-hot encoding basically creates new columns which contain binary values indicating the presence of all the unique values in the data.

We have imported *preprocessing.OneHotEncoder* from *sklearn* in order to convert *sub1, sub2, sub3 and brand* into One-Hot Encoded form.

### *B. Vectorization of textual features:*

Since the textual features can not be fed into our models, we need to convert them into numerical form or vectors.

**Bag of words:** A Bag of Words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of presence of known words.

The output will be a fixed length vector of numbers and each number indicates the number of times a word is repeated.

We have imported *CountVectorizer* from *slkearn.feature_extraction.text* in order to apply this vectorization. We have used *max_features* = 150000. Our *ngram_range* = (1,2) i.e, the vectorizer is generating both single and two word phrases from the given corpus.
We have implemented the Bag of Words model on *name* feature.

**TF-IDF:** One of the problems of using Bag of Words is that it doesn't take into consideration the order of words. Since the name feature contains only nouns, it was not a big problem there. The *item_description* feature contains textual description which has a context and meaning, so if we use Bag of Words, a high count of less meaningful words will not provide much value to our learning model. Hence we used the efficient TF-IDF vectorization to vectorize the *item_description* feature.

In TF-IDF, the Term Frequency (TF) gives the count of words. On the other hand, the Inverse Document Frequency (IDF) provides less weightage to the words which repeat across all the documents.

We have imported *TfidfVectorizer* from *sklearn.feature_extraction.text* to implement the TF-IDF vectorization. We have used *max_features* = 500000. Our *ngram_range* = (1,3) i.e, the TF-IDF generates both 2 and 3-word phrases along with the single words from the given corpus. Also, *min_df* = 10 i.e, during the building of vocabulary, all the terms that have a document frequency lower than 10 are ignored.

We implemented this Vectorization on *item_description*.

### *C. Sentiment Analysis:*

We have implemented a Sentiment score for *item_description* feature. We imported *SentimentIntensityAnalyzer* from *nltk.sentiment.vader* to implement this. This function reads the data and analyses this in four different perspectives namely *negative, positive, neutral* and *compound*. How is this analysis helpful?

- Most of the time, a positively scored description may implicate a high price item.
- Similarly, a negatively scored description may implicate a low price item
- Hence, this formulates a correlation between *item_description* and our target variable *price*.

The evaluation results are not binary instead we get them as float format. This function returns a dictionary consisting of all these four values. Later we have split this dictionary into four different features. Also, note that we have applied this before preprocessing the data because the function results in two different values for GOOD and good. The results confirm the same i.e, better accuracy is obtained if we applied it before preprocessing than the processed text.

### *D. Item_description length and name length:*

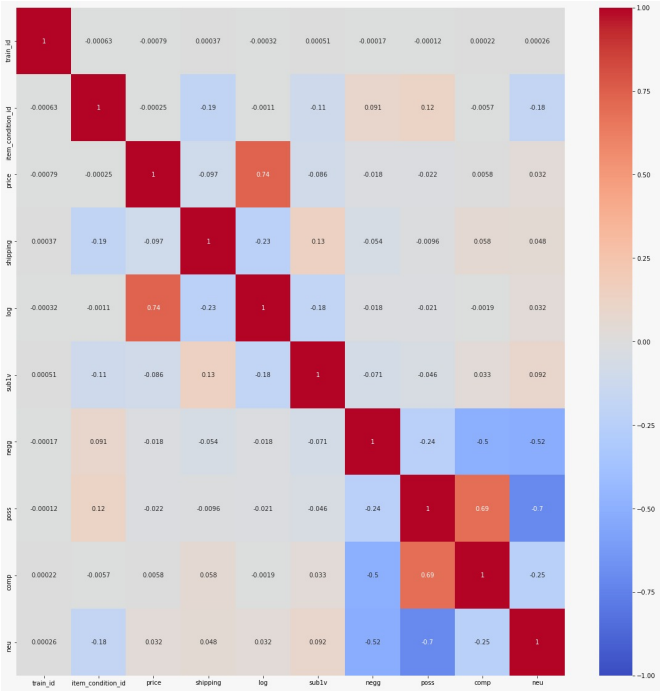
After preprocessing the *name* and *item_description* features, we felt that the length of these texts would also impact the price of the items. Hence we extracted both the *name* length and *item_description* length and created two new features for our model.

### *E. Removing Highly correlated features*

When two or more features contribute almost the same value of information in predicting the target variables, they are considered to be highly correlated. It is essential to keep only one feature and remove the other highly correlated features. If we do not perform this step, the results we obtain from the regression will be inaccurate.

We can identify such features using the correlation matrix.

From the below plot, we can see that none of the features has a correlation higher than 0.95 with any other feature. Hence we can proceed to regression.




## VI. Model Selection

After having done the cleaning and pre-processing of data, using the Feature extraction steps we created all the required features for building our model. We created a sparse matrix using the numerical format of all the encoded features obtained through through one-hot encoding. We created another sparse matrix using the numerical features. Using hstack we horizontally stacked this sparse matrix with the previous sparse matrix that we got after one-hot encoding. In simple words, all the features are concatenated into a sparse matrix which is our final matrix. This prepared matrix contains a lot of features. This matrix is used to feed various machine learning regression models that we have tried.

There are plenty of machine learning algorithms that we can try out depending on the usage of memory, time, type of problem, accuracy etc. So, we started experimenting by training the data with linear regression model with the aim to improve the root mean squared logarithmic error (RMSLE) I.e., to get prediction prices closer to the actual prices.

We have experimented the following linear regression models and boosting models:

- Linear Regression.
- Ridge Regression.
- XGBoost
- Light Gradient Boosting Machine (LGBM).

**Linear Regression:** *Linear Regression fits a linear model with coefficients w = (w1, . . . , wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.* [1]

Easy and simplest out of all the regression models is a linear regression model for prediction. As a beginning, we trained our model using linear regression. A linear regression model is initially trained on label encoded data which did not give us good accuracy, so we then shifted to one-hot encoding which gave us better results. We trained the model by tuning hyperparameters by trying different values of Alpha. Tested the model to make it more efficient by tuning the hyperparameters and comparing RMSLE function values after each try to obtain the prediction price closer to the actual price. The Final RMSLE functional value obtained for when trained with linear regression is 0.61264 on the public leaderboard.

**Linear Regression RMSLE on public leaderboard – 0.61264**

**Linear Regression RMSLE on private leaderboard – 0.61548**

**Ridge Regression:** *Ridge Regression is a type of regression that use L2 Regularization, which aims for low training error while balancing against model complexity.*[2]

L2 Regularization is about:

- Penalizing really big weights
- Reducing model complexity
- Shrinking coefficients, but not eliminating them (unlike L1 Regularization)

L2 Regularization Loss Function:

- L: aims for low training error
- Lambda: A scalar value that controls how weights are balanced
- W: weights are balanced against complexity

$$L(\boldsymbol{w}, D) \; + \; \lambda \, || \, \boldsymbol{w} \, ||_2^2$$

Fig. 1. Cost Function

As a process of experimenting and improving the accuracy, we trained the model using Ridge regression which reduces the complexity of coefficients which results in avoiding the overfitting. In order to apply ridge regression, we used ridge_regressor from ski-kit learn and the hyperparameters were tuned using GridSearchCV. We tried to make the model more efficient by tuning the hyperparameters and comparing the obtained RMSLE functional values. After several trials, we obtained a good fit for alpha = 5.5. The ridge regression gave better results compared to linear regression. We further modified some preprocessing functions by adding and replacing new features like TF-IDF, sentimental score, etc. to understand which works better for our model.

**Boosting the Model**

**XGBoost:** *XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the*

*Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way*. [5]

**LGBM:** *LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the advantages like Faster training speed and higher efficiency, Lower memory usage, Better accuracy. [4]*

We initially boosted the obtained ridge regression model by using XGBoost which is an implementation of the gradient boosted decision trees. When used GridSearchCV we directly obtained the best fit that suits our model. But the obtained results were not satisfactory. So, we then shifted to Light Gradient Boosting Machine (LGBM) which is known for its less memory usage and a good model to use on big data sets. This gave us better results compared to xgboosting. We kept continuing the process of adding new features, tuning the hyperparameters till we obtained a better RMSLE score. After several trials, we obtained a good fit for alpha = 2.0. The Final RMSLE functional value obtained when trained with ridge regression and boosted with LGBM is 0.43411 on the public leaderboard.

The following parameters were used for implementing LGBM-Classifier:

- n_estimators=4000
- max_depth=9
- num_leaves=64
- learning_rate=0.4

ENSEMBLING: An ensemble method is a technique in which we use more than one model to make a better prediction or increase accuracy or even boost the performance. Every model's prediction is influenced by bias, variance and noise. In order to combat these disadvantages, we are using ensembling.

**Ridge Regression (boosted with LGBM) RMSLE on public leaderboard – 0.43411**
**Ridge Regression (boosted with LGBM) RMSLE on private leaderboard – 0.43378**

## VII. Table on Models and accuracy

| Regression Model | RMSLE score |
|---|---|
| LR[1] | 0.61548 |
| RR[2] | 0.60721 |
| RR[3] | 0.62280 |
| RR[4] | 0.46564 |
| RR[5] | 0.45468 |
| RR[6] | 0.43378 |

Fig. 2. Table depicting various methods and the corresponding RMSLE scores on private leaderboard

*1. Linear Regression, Sentimental Analysis, Label Encoding,Bag of words*
*2. Ridge Regression, Sentimental Analysis, Label Encoding,Bag of words*
*3. Ridge Regression,TF-IDF, Sentimental Analysis, Label Encoding, Bag of words*
*4. Ridge Regression, TF-IDF, Sentimental Analysis, One-Hot Encoding, Bag of words*
*5. Boosted the existing Model with XGBoost*
*6. Boosted using LGBM, Best value of Alpha that fits our model after tuning Hyperparameters*

## Conclusion

We would like to conclude that we were able to come up with an efficient model that can automatically suggest product prices to online sellers. Our model has obtained an RMSLE score of 0.43378 on private leaderboard. These project has potential scope to save a lot of time to online sellers across the globe by predicting the prices for a large set of products to the onine sellers efficiently within no time.

## CHALLENGES AND FUTURE SCOPE

After participating in this challenge, we got a better understanding of the commonly used approaches to solve the Mercari Price Suggestion Challenge. As a future scope, we would like to try other regressions like Lasso regression, non-linear regression models and also we want to try Convolutional Neural Networks (CNN) to this problem. We want to try other possible ML algorithms to solve such problems. We will focus on making our model efficient for both time and space consumptions. We will also focus on tuning the hyperparameters for the faster convergence of the model.

## Acknowledgement

We would like to thank Professor G. Srinivas Raghavan, Professor Neelam Sinha and all our Machine Learning Teaching Assistants for giving us the opportunity to work on the project. Special thanks to Teaching Assistant Amitesh Anand for helping us whenever we were struck by giving us ideas and resources to learn from. We had a great learning experience while working on this project and leaderboard constantly motivated us to try new stuff to improve our model to obtain better results.

## References

[1] sklearn. linear_model.LinearRegression [online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html [Accessed: October 2020]
[2] sklearn. linear_model.Ridge [online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html [Accessed: October 2020]
[3] "Mercari Price Suggestion Challenge", Mercari, [online]. Available: https://www.kaggle.com/c/mercari-price-suggestion-challenge [Accessed: October 2020]
[4] "Features", LightGBM Documentation. [online] Available: https://lightgbm.readthedocs.io/en/latest/ [Accessed: October 2020]
[5] "XGBoost Documentation", XGBoost, [online]. Available: https://xgboost.readthedocs.io/en/latest/ [Accessed: November 2020]

[6] A. Natekin and A. Knoll, “Gradient boosting machines,” [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full/ [Accessed: November 2020].
[7] The editors of Wikipedia, “Exploratory data analysis – Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Exploratory_data_analysis [Accessed: November 2020].
[8] S. Chatterjee, “Good Data and Machine Learning - Towards Data Science,” [Online]. Available: https://towardsdatascience.com/data-correlation-can-make-or-break-your-machine-learning-project-82ee11039cc9/ [Accessed: November 2020]
[9] https://machinelearningmastery.com/gentle-introduction-bag-words-model/ [Accessed: November 2020]