# Mercari Price Suggestion

## Team: Pirates

Kalyan V. Nadimpalli
*IIIT, Bangalore*
kalyan.varma@iiitb.org

Shashank Reddy
*IIIT, Bangalore*
shashank.reddy@iiitb.org

R.V.S. Ajith
*IIIT, Bangalore*
rvs.ajith@iiitb.org

***Abstract*—Price prediction of common commodities is a non-trivial task. Small details can make a world of difference, for example the season can determine the value of a garment. In an online retail store sellers can choose just about any price they please. Websites do their best to suggest appropriate prices for which the item will be sold, but doing this manually is an impossible task as it is simply unscalable.**
**In this project we use classic NLP models in combination with various traditional Machine Learning algorithms to automate this process of predicting prices.**

***Index Terms*—component, formatting, style, styling, insert**

### I. Introduction

In a world where online retail is now the go-to method of shopping, there is an exponential rise in the number of purchases made online. Sellers have the power to set the price as they please, but this leads to great inefficiency. A seller could either set the price much too high to reduce the number of sales they could make or the seller could set it too low which would reduce their profits. It is near impossible to manually find that perfect price right in the middle simply due to the number of parameters that influence the price. There is an urgent need to solve this problem efficiently with an automated solution.

The challenge of performing such a task automated is both in the number of products that have to be dealt with, i.e, there is a huge dataset that must be handled to understand the market and the fact that most of this data is in English that is human parsable text not in a form that a computer can parse.

This report presents a methodology for an analysis of the market with the help of traditional NLP and Machine Learning methods. The dataset used to train this model was provided by Mercari which is a Japanese e-commerce company. Our model given certain features like name of product, item description, item condition, etc. can determine and suitably provide a price for it to be sold at.

The rest of the paper proceeds as follows: in [2] we describe the dataset, in [3] we show visualisations of the data. In [4] we explain the pre-processing steps we took and in [5] we explain our choice of models and the performance of each. Finally, we conclude in [6] and provide areas of improvement and the challenges we faced in [7].

### II. Dataset

The dataset used to understand and solve this problem was the publically available Mercari dataset from a competition hosted by Mercari on Kaggle.

Each data point in the training dataset contained 7 features, of which price was the target variable, i.e, what we predicted after training of the model. There were 4 features which were strings which were the name of the product, a description of the product, brand and the category of the product. Apart from this there was also a feature called item-condition which ranked the condition of the product from 1 to 5. Finally there was a boolean variable which mentioned if shipping was free or not.

Looking at the data and the target variable it is evident that this is a regression problem, i.e, we've to predict a real valued number given the data of a product.

### III. EDA

The given dataset has a large number of null values in the brand-name and category-name columns. The price is heavily left-skewed as shown in figure 1. We observe that most of the

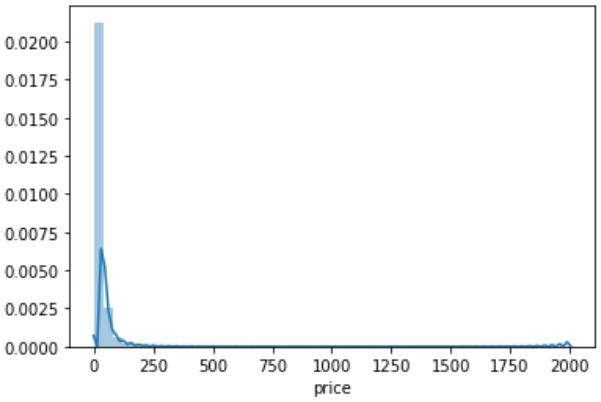


Fig. 1.

item-descriptions have a relatively short length that is length less than 200. The plot of the value-counts is shown in figure 2. We notice that there is a linear relation between the length of the item description and the price for products with shorter lengths which is most of them.

The mean price for products where shipping is free is 30 dollars while it is 22 dollars for products where the shipping isn't free. This shows that customers are willing to pay more for their products if they don't have to pay an additional

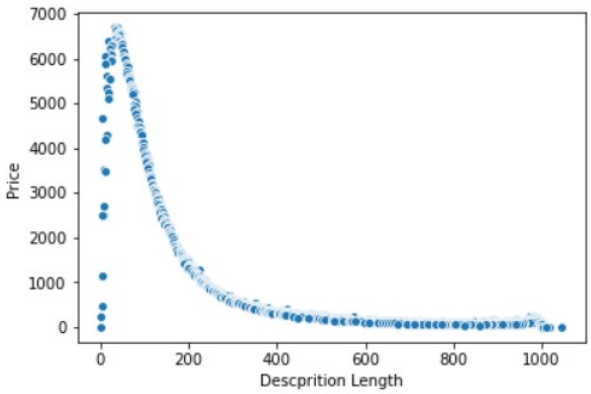


Fig. 2.

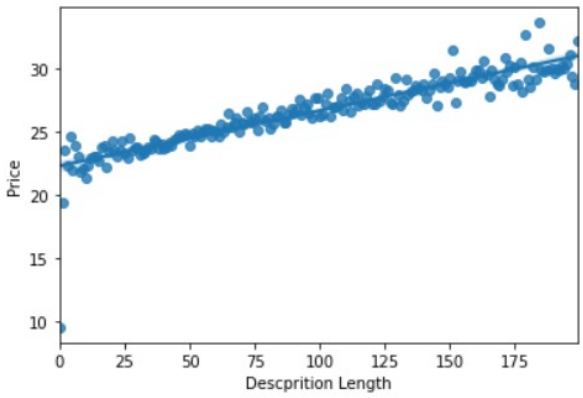


Fig. 3.

shipping cost. We split the category into three sub-categories. The number of main categories in our data is 11.
While the number of subcategory1 in our data is 114 and finally the number of unique values in subcategory3 is 858.
Word clouds are visual displays of text data, they show the most frequent and most important words from a corpus. Below are the word clouds of various features.

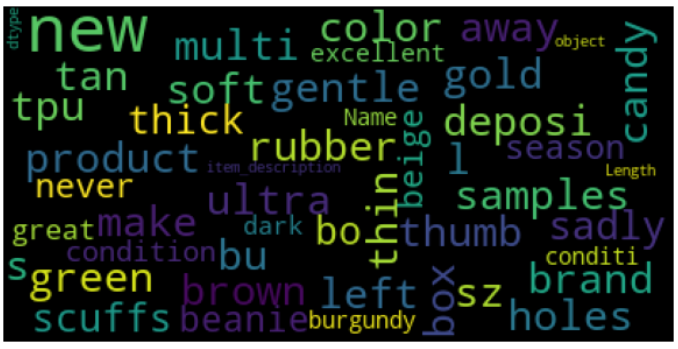


Fig. 4. Item-Description Word Cloud

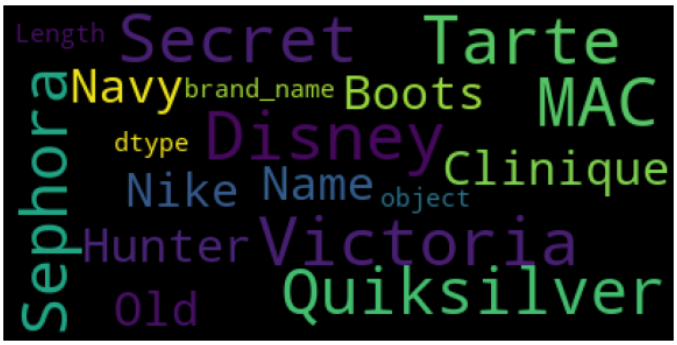


Fig. 5. Brand-Name Word Cloud


Fig. 6. Name Word Cloud

## IV. Data Preprocessing and Feature Extraction

To improve the speed and accuracy of training the model we must modify the data into a form that the model can work with best. Models trained on this transformed data always outperform models trained on the raw data itself and need less data to achieve similar results.
In the given dataset there were a large number of null values in both the brand name and the category name. One interesting observation about the dataset was that sometimes the brand-name could be deciphered from the name of the product and the item description of the product. For example an iphone entry could have no brand-name in it but the name field could be ”Apple Iphone” or the item description could be ”Brand new Apple Iphone”. So we collected a list of all the known brand names, and checked if any of them matched with a word in the name or in the item description, if they did we filled in the brand name. This significantly reduced the number of null values, for the remainder we simply replaced them with an empty string.
Categories were in the form of a tree hierarchy, like electronics/phone/smart phone. That is there was a hierarchy category, subcategory and sub-subcategory. For the null values we simply replaced them with //. That is saying their category, subcategory and sub-subcategories are empty. For convenience from here as we did in our code we shall call category as category1, subcategory as category2, sub-subcategory as category3. After filling in the null values,

we split this string into 3 new features namely category1, category2 and category3 based on the /.

The price feature was heavily skewed as shown in the EDA. We applied a log transformation to make it closer to a gaussian distribution which enhances learning in several models.

After this we used standard NLP pre-processing techniques to improve the format of the strings before converting them into word embeddings. We tokenized the strings that is made them into lists of words. We then lemmatized the words that is brought them into their roots forms, eg: running to run. After this we removed all stopwords as they don't provide any significant meaning to the sentence and unnecessarily confuse the model. Stopwords are general words like a, an, this, that, etc. In the item-description of various products there were emojis which we removed.
From here we began feature-engineering. Notice we haven't properly handled all the null values in the brand values and simply replaced them with empty strings. To overcome this issue and also to generate columns of semantically different meanings we generated new features which were combinations of the previous features. That is we introduced the features name-description, name-brand and brand-description which is simply the concatenation of the respective features. We applied the appropriate pre-processing described above on these features. After which we removed the duplicates from each of these features as words could repeat which are common in both the original features.

There were 2 main NLP embeddings that we used to convert these string features into equivalent integer ones. The first was TF-IDF and the second was a pre-trained Word2Vec model found[ref1]. The issue with Word2Vec for this particular problem is that many of these words aren't commonly found and hence aren't trained correctly in the Word2Vec model. For example, brand names like Underarmour, Nike aren't handled and Word2Vec returns a random vector instead. So for our final works we switched to TF-IDF. More details on our experiments and performances for each is in the Model Selection section below.
Finally, for the category columns we applied label encoding as it provides a better performance on tree based models which we used.

## V. Model Selection

Initially we used TF-IDF as our word embeddings and one-hot encoding for our brand-name and category-name on top of which we applied Truncated SVD a form of PCA to reduce the dimensionality. Our initial selection of models on this transformed data was just an exhaustive search over various linear models like Linear Regression, Ridge Regression, Lasso Regression, ElasticNet, etc.

We then moved on to a better embedding Word2Vec and applied the same models. We spent a lot of time tuning hyperparameters but our score was peaking at 0.6. We then realised Word2Vec doesn't work that well with proper nouns. Our dataset has a large number of such proper nouns: brand names, company names, etc.

At this point we switched back to TF-IDF but instead of applying Truncated SVD we leveraged the fact that TF-IDF returns a sparse matrix. While this Sparse Matrix can not be used to train linear models, it could be used to train tree based models such as xgboost, catboost and lightgbm. As we decided to use tree-based models, we switched all the columns where we used one-hot encoding to label-encoding as tree-based models perform better on label-encoding. Here we experimented with creating new features, we observed that when columns were combined to form larger masses of text our models performed better. Our score dropped to 0.53 with our new approach.

We attempted to use stacking an ensemble method to combine various models like xgboost, catboost and lightgbm but it didn't help with our score. This is probably due to the fact that typically stacking is done with hundreds and even thousands of models while we used only 3-5 at a time.

On tuning hyperparameters we finally achieved our best score of 0.51 when we used catboost as our model.

| Model | Embedding | Log Mean Squared Error |
|---|---|---|
| Linear Regression | TF-IDF | 0.64547 |
| Ridge Regression | TF-IDF | 0.59889 |
| Lasso Regression | TF-IDF | 0.70261 |
| ElasticNet Regression | TF-IDF | 0.63544 |
| Ridge Regression | Word2Vec | 0.56457 |
| Lightgbm | Word2Vec | 0.61440 |
| xgBoost | Word2Vec | 0.54146 |
| xgBoost | TF-IDF | 0.53996 |
| Catboost+xgBoost+Lightgbm (Stacked) | TF-IDF | 0.53781 |
| Catboost | TF-IDF | 0.51405 |

Fig. 7. Model Performance

## VI. Conclusion

We were able to come up with an efficient model to predict prices of various goods based on a human written description of the item. Such projects have a potential scope of suggesting meaningful prices to sellers which in turn will improve the efficiency of the market.

## VII. Challenges and Future Scope

One significant challenge was the number of null values in brand-name and category-name, accurately filling these in based on the other fields wasn't easy at all.

Another challenge was the types of words we had to deal with, we had to deal with various uncommon companies and brands. Due to this pre-trained embeddings like Word2Vec weren't suitable and we had to stick with TF-IDF though Word2Vec model better captures the meaning of a sentence. This can be solved in future attempts by training our own Word2Vec neural network on this dataset.

In this attempt we used classical machine learning models to solve this problem. Neural networks could possibly improve our solution as they eliminate the challenge of feature engineering and are in general much more versatile.

## VIII. Acknowledgements

We would like to thank Professor G. Srinivas Raghavan and Professor Neelam Sinha along with our numerous TA's, primarily our project head Amitesh Anand for giving us the oppurtunity to work on this project and for helping us whenever we got stuck.

We thank all the TA's for the effort they put in to conduct insightful tutorial sessions held every Friday. Many of them inspired us and taught us new techniques which we incorporated and experimented with in our project.

We would also like to thank many of our helpful peers who have posted valuable resources in the Slack discussion forum. Overall this was a great learning experience and we had a lot of fun in the process.

## References

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[2] Array-Programming: with Numpy, Charles R. Harris and K. Jarrod Millman, et al., Nature Journal, 2020

[3] E. Jones, T. Oliphant, P. Petersonet al., "SciPy: Open source scientific-tools for Python," 2001–, [Online; accessed ¡today¿]. [Online].Available: http://www.scipy.org/

[4] . S. Hendrik Jacob van Veen, Le Nguyen The Dat, "Kaggleensembling guide," 2015, [Online; posted 6-February-2018]. [Online].Available: https://mlwave.com/kaggle-ensembling-guide/

[5] Word2Vec embeddings, Stanford, GoogleNews-vectors Available:https://drive.google.com/file/d/0B7XkClNUT21pQmM/edit

[6] NSS, Analytics Vidhya https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

[7] Kaitlyn, "Calculating tf-idf with python," 2017, [Online; posted 3-June-2017]. [Online].
Available: https://methodi.ca/recipes/calculating-tf-idf-python.