Machine Learning Nanodegree
Capstone Project

Mercari Price Suggestion Challenge

Quan Yuan
March 24th, 2018

1. Definition

1.1 Project Overview

It can be really hard to know how much something's really worth. Clothing has strong relation with seasons and brand names, while electronics' price mostly based on product specs. Mercari, Japan’s biggest community-powered shopping app, have the same trouble about offering the reasonable pricing suggestions to sellers and put this problem in the recent Kaggle competition Mercari Price Suggestion Challenge[1]. In this competition, Mercari provided detailed data including text descriptions of their products, product category name, brand name, and item condition[2]. For reducing the training time, only 'train.tsv' data will be used in this project. The challenge thing I do is to build an algorithm that automatically suggests the product prices.

In the past, many researches have done for predicting price, like B Park and JK Bae try to predict the house price in Fairfax Country with machine learning algorithms such as C4.5, Adaboost[3], Hezhong Dai try to predict stock index future based on SVM[4]. The approaches those works use are worth learning. However, few works directly relate to predict the general products' price like the furniture price, clothes price, which make this problem quite challenge.

1.2 Problem Statement

In this problem, I need to build an algorithm for helping automatically suggests the product prices. Since the object variable is continuous, the solution for this problem is to use regression algorithm to fit the price based on the features of the product. Therefore, there are two necessary steps to tackle this problem. First, we need to construct features. The most difficult thing in this part is to process txt data and I will try my best to find more information from txt based on Tf-idf model. Then we use those features and try to find a suitable algorithm to do regression work making the best result in the test dataset. In this part, many machine learning models like Ridge, LightGBM will be employed to do the prediction work and we will compare the prediction results based on cross validation for finding the best model of

[1] https://www.kaggle.com/c/mercari-price-suggestion-challenge

[2] https://www.kaggle.com/c/mercari-price-suggestion-challenge/data

[3] Byeonghwa Park, Jae Kwon Bae, Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data, Expert Systems with Applications, Volume 42, Issue 6, 15 April 2015, Pages 2928-2934, ISSN 0957-4174

[4] Y. Zhang, H. Dai and D. Wang, "Price Prediction of Stock Index Futures Based on SVM," 2011 Fourth International Conference on Business Intelligence and Financial Engineering(BIFE), Wuhan, Hubei China, 2011, pp. 54-57.doi:10.1109/BIFE.2011.96

this problem. After modeling work, some evaluation jobs will be done to assess the model performance, I hope the Root Mean Squared Logarithmic Error in the test dataset for this algorithm will less than the benchmark result.

1.3 Metrics

The evaluation metric for this problem is Root Mean Squared Logarithmic Error (RMSLE)[5]. RMSE is a very common evaluator and it makes an excellent general purpose error metric for numerical predictions. Compared to Mean Absolute Error, RMSE amplifies and severely punishes large errors, which is suitable for this problem since the majority products' price around mean value. If the large error is not been punished severely, we cannot get the reasonable score since the model can just predict the price around the mean value instead of considering the expensive product price suggestion. In this view, we could get a model that cannot be generalized well.

The RMSLE formula shows below.

$$\varepsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Where:
ε is the RMSLE value (score)
n is the total number of observations in the (public/private) data set
p_i is the prediction of price
a_i is the actual sale price for x

2. Analysis

2.1 Data Exploration

There are seven columns in the dataset including product name, item condition, category name, brand name, txt description, shipping and product price. The following **Table 1** shows the sample raw data.

Table 1: Sample of raw data

Feature	*Value*	*Feature*	*Value*
Train_id	0	Brand name	Target
Name	AVA-VIV Blouse	Price	10.0
Item_condition_id	1	Shipping	1
Category_name	Women/Tops & Blouses/Blouse	Item_description	Adorable top with a hint of lace and a key hole ...

For the object variable *price (continuous, numerical)*, the following **Table 2** shows the basic description about the price data.

Table 2: Description of price data

[5] https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError

Data	*Mean*	*Std*	*Min*	*Max*	*Q 25%*	*Q 50%*	*Q 75%*
Price	26.69	38.34	0	2000	10	17	29

After some analysis, we could also get some observations about the price data.

- 0.052% products price equal to zero
- 1% products price lower than $ 4; 90% products price lower than $ 52; 99%; products price lower than $ 171; 99.9% products price lower than $ 451; 99.99% products price lower than $1060

The product price is not normal distributed based on the **Figure 1** (set price lower than 171 for better visualization), but after log transformation, the product price follows the normal distribution. Therefore, we use log function to transform the price data for getting better result. Besides, generally the products' price is not likely being zero. Therefore, the 0.052% products which price equal to zero have been eliminated. **Figure 2** shows the distribution of the product log price.

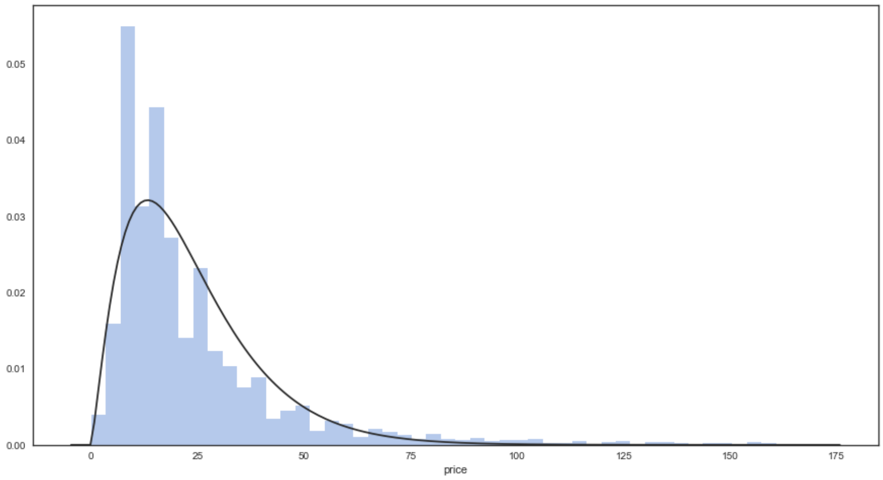

Figure 1: The distribution of the product price

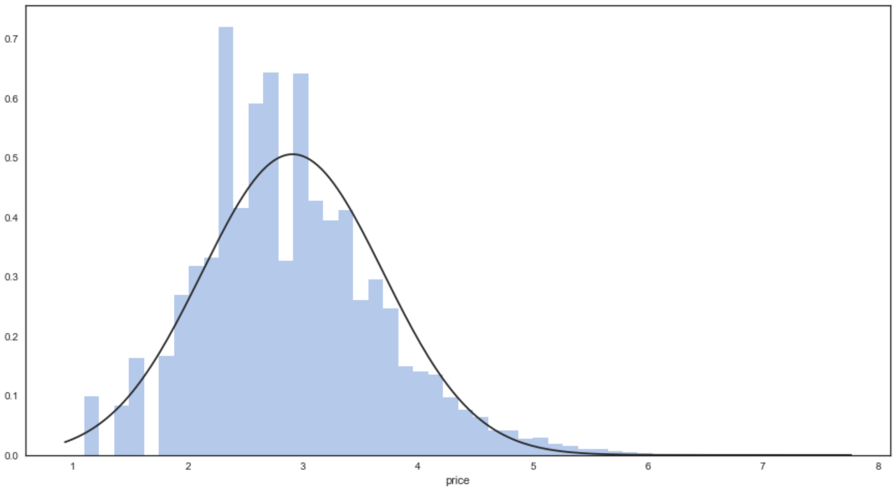

Figure 2: The distribution of the product log price

For the *item condition (category, numerical)*, there are five types of condition in the dataset (1, 2, 3, 4, 5) and the most common one is 1. We are interested in the relationship between the product price and item condition. The following **Table 3 and**

Figure 3 describe the price distribution when the products in different item condition.

Table 3: Description of price data and item condition

item condition	1	2	3	4	5
Percentage	43.2%	25.3%	29.1%	2.2%	0.2%
Mean	18.26	18.84	18.32	16.80	20.68
Median	18.00	17.00	16.00	15.00	19.00
Min	3.00	3.00	3.00	3.00	3.00
Max	1909	1815	2000	1106	360

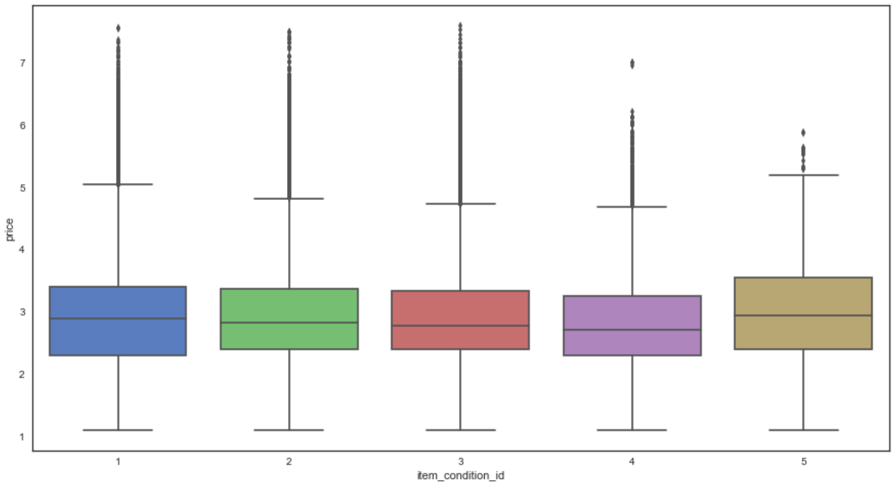

Figure 3: The relationship between log price and item condition

From Table 3 and Figure 3, we could draw a conclusion that when the item condition is equal to 5, the products more likely have a higher price. When the condition is 4, the price seems lower.

For the 0-1 *shipping* feature, the mode of it is 0, the description of this feature shows below (**Table 4**), Figure 4 shows the price distribution when the products have different shipping condition.

Table 4: Description of price data and shipping condition

Shipping	1	2
Percentage	55.4%	44.6%
Mean	30.07	22.52
Median	20.00	14.00
Min	5.5	3
Max	1909	2000

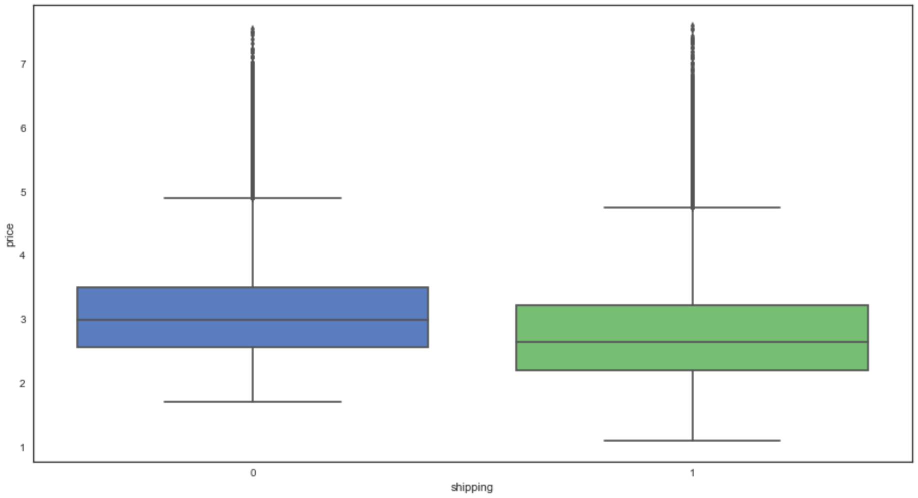

Figure 4: The relationship between log price and shipping

From Table 4 and Figure 4, we could see that the products with shipping equal to 1 have higher price than the products with shipping 0.

For the *category name (category, string)*, the first thing we do is to spite each string to three separate categories. There are 11 first order classes, 114 second order classes and 871 third order classes. Notice that the first order ⊃ the second order ⊃ the third order. The following **Figure 5** shows the price distribution when products in different first order classes.

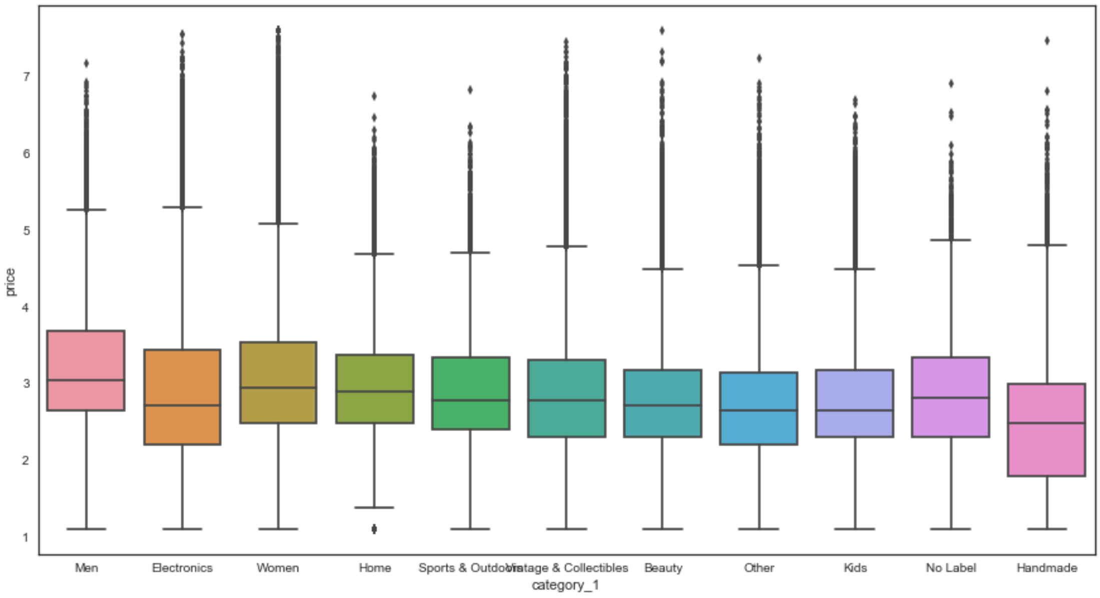

Figure 5: The relationship between log price and category name

Form Figure 5, we could conclude that different category have different price distribution and class like handmade equipped with lower price. However, products belong to man class have higher price.

For the *brand name (txt, string)*, there are 3751 different brand names, but the nan rate is 42.64%. For better evaluate the relationship between price and brand name, we use word cloud to show the expensive (price above 171) products' brand name **Figure**

6 and the cheap (price below 4) products' brand name **Figure 7**.

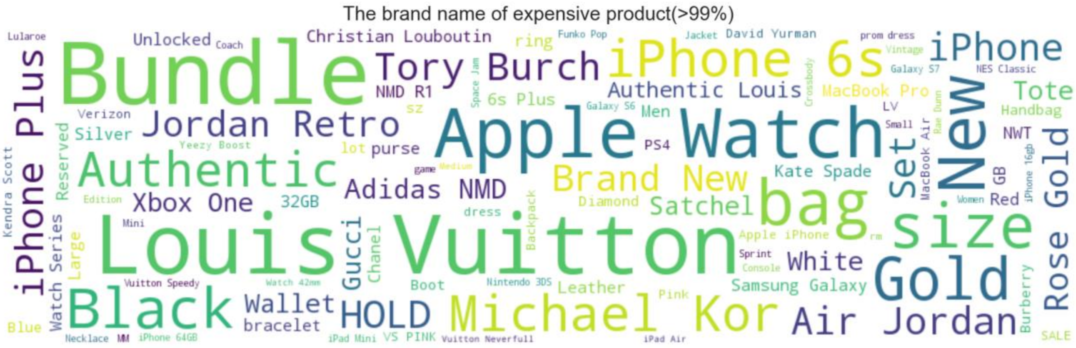

Figure 6: Word Cloud for expensive products' brand name

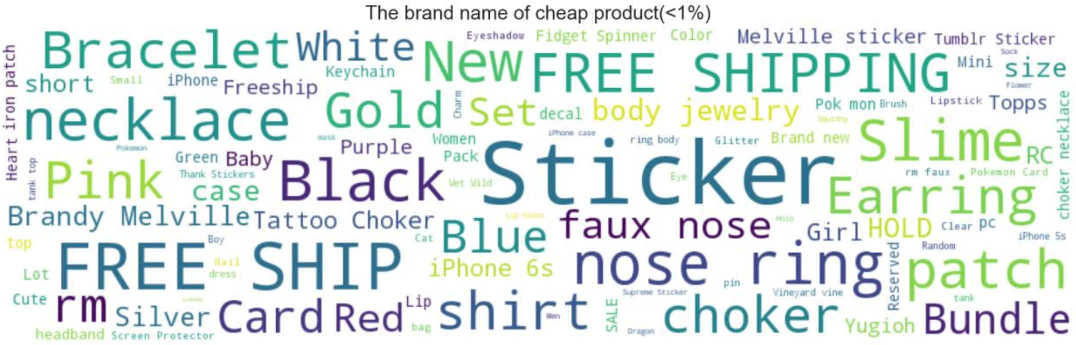

Figure 7: Word Cloud for cheap products' brand name

From Figure 6 and Figure 7, we could see some expensive brand names like Louis Vuitton, Gold, Apple Watch and also see some cheap products like Sticker, Free shipping.

For the *item description* and *products' name (txt, string),* both feathers are string type and we need some nlp methods like Tf-idf to process the txt data, we will do that later. Notice that 5.57% of products don't have item description.

2.2 Algorithms and Techniques

For processing txt data, Tf-idf model in Sklearn package will be used to convert txt data to numerical matrix since this method is really popular in NLP field. Notice that this method won't be used in the benchmark model part, the easier way that calculating the length of txt will be employed for simplifying the calculation. The parameters in Tf-idf will be illustrated in the next part (3.2).

Let's go through Tf-idf algorithm with an example. Suppose we have three simple documents:

- This is a pencil.
- This is a car.
- This is a banana.

We want to find information from those three sentences. The first thing we do is to count the word frequency since that will help us know their common and difference. After that, we got a matrix of word frequency (6×6).

$$\begin{array}{lcccccc} & This & is & a & pencil & car & banana \\ document\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ document\ 2 & 1 & 1 & 0 & 0 & 1 & 0 \\ document\ 3 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

The next thing we do is to calculate tf and idf value. For word 'this', tf is just the frequency of the "this" for each document. So $\mathrm{tf}('\mathrm{this}', \mathrm{d1}) = 0.25$, $\mathrm{tf}('\mathrm{this}', \mathrm{d2}) = 0.25$, $\mathrm{tf}('\mathrm{this}', \mathrm{d3}) = 0.25$. idf is the ratio of documents that include the word "this". So, $\mathrm{idf}('\mathrm{this}', \mathrm{D}) = \log(3/3) = 0$. Thus we got $\mathrm{tf} - \mathrm{idf}('\mathrm{this}', \mathrm{d1}) = 0.25 \times 0 = 0$. Every words' tf-idf in the document can be calculated by this way.

For reducing the dimension of features, we will use Truncated SVD[6] method since it can work with sparse matrices efficiently and always used for processing Tf-idf matrix.

SVD[7] method suitable for general matrix while eigen value decomposition only applied when data is a square array. We want to use this method to find the important features from matrix. The SVD of an M by N rectangular matrix A has this form:

$$A_{m\times n} = U_{m\times n} \times S_{m\times n} \times V'_{m\times n}$$

Where U is an orthogonal matrix, whose columns are the left singular vectors; S is a diagonal matrix, whose min(m, n) diagonal entries are the singular values; V is an orthogonal matrix, whose columns are the right singular vectors; V is the matrix that we want. If we want top r features from A, the from can be rewritten by Truncated SVD form, which satisfied our need.

$$A_{m\times n} \approx U_{m\times r} \times S_{r\times r} \times V'_{r\times n}$$

In the modeling part, many different models have been tried like LightGBM, GradientBoosting, Ridge. We choose LightGBM because it has better performance and faster training speed than Xgboost which has been widely used in many Kaggle competitions. In the benchmark model part, the random forest model perform well which motivated me to try Gradient boosting method since this method handle error data better. We use Ridge methods because it always has good performance in large dataset and the training speed is good. I don't try to use SVM since many papers have illustrated that SVM performs good in small dataset but not good in large data, and we also don't use NN methods considering the extreme long training time and messy parameters.

We trained random forest with random state equals to 0, Gradient boosting with random state equals to 1. For LightGBM, there are many parameters which need to be defined. We set n_estimators equals to 1000, learning rate equals to 0.1, num_leaves equals to 31, subsmaple equals to 0.9, colsample_bytree equals to 0.8,

[6] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
[7] http://people.sc.fsu.edu/~jburkardt/m_src/svd_truncated/svd_truncated.html

min_child_samples equals to 50, n_jobs equals to 2. We will adjust some parameters in LightGBM with Grid Search method later. For Ridge regression, we use RidgeCV[8] to find the best alpha parameters, but the score is not sensitive to alpha.

For Random Forest Algorithm, it operates by constructing a multitude of decision trees at training time and outputting the mean prediction of the individual trees, which help avoid overfitting. Decision Tree is a popular method. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. Finally, we could get a tree with decision nodes and leaf nodes. The decision node helps decide which category the data belong to. The Leaf node tells us the exact value of this category. There are lots of algorithms of getting the tree model like ID3 (Step1: calculating information gain; Step2: split the set into subsets using the feature with max IG).

Gradient Boosting method can be interpreted as an optimization algorithm on a suitable cost function. This algorithm needs to train the weak leaner (decision tree) one by one since the later learner needs to fit the error from the former for minimizing the error. LightGBM is a better gradient boosting framework, which also based on Tree model. The important difference between LightGBM and GBDT is LightGBM keep the data with large error and sampling remain data before iterating, which both reduce the training data and not change the distribution. Some memory optimization and parallel processing support also done in LightGBM framework.

Ridge is a linear regression method. For ordinary least square method, our object is to minimize the cost function:

$$||X\theta - y||^2$$

But this method is not stable since the matrix $X'X$ becomes close to singular when terms are correlated and the columns of the design matrix X have an approximate linear dependence. The solution to this problem $\theta = (X'X)^{-1}X^Ty$ becomes highly sensitive to random errors in the observed response y, producing a large variance. There for we add L1 regularization to eliminate this effect. The solution becomes $\theta = (X'X + kI)^{-1}X^Ty$.

In the last part, some ensemble learning methods will be used to get better results. First, we will try to average each predict results. Second, we will try to weighted each predict results with their RMSLE score. Finally, we do stacking to get better model with mlxtend StackingRegressor.

Stacking regression involves combining multiple regression models via a meta-regressor. First, each regression models are trained based on the complete training set; then, the meta-regressor is fitted based on their outputs. The Stacking process shows below.

[8] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html

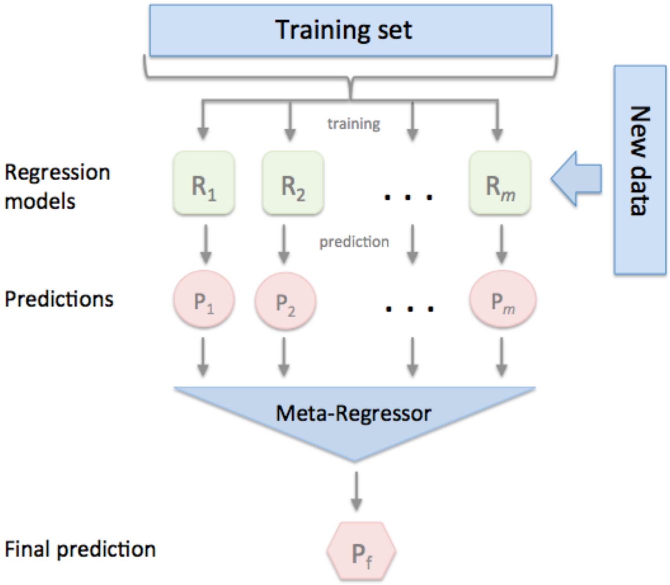

Figure 8: Stacking process in regression

Source: http://rasbt.github.io/mlxtend/user_guide/regressor/StackingRegressor/

For evaluating the model, we will use cross validation to test whether the model generalizes well and various input will be tried to test the robust. Cross validation is a popular method in ML field. Every time we choose different train set and test set to calculate the score which help us test the robustness of the model.

2.3 Benchmark

Random Forest with default parameters is the benchmark model in this project. I want to use an easiest way to get a model, so I took the length of item description and product name as the substitution for txt data. Category features have been transformed by Label Encoder[9]. After fitting the txt dataset, we get RMSLE equals to 0.1663. In order to test the robust, we use cross validation method and the mean squared error in five folds are -0.41217251, -0.40989434, -0.41463082, -0.41357113, -0.41064602, which represents the model is not overfitting. (RMSLE is not acceptable in cross validation, so we use negative mean squared error)

3. Methodology

3.1 Data Preprocessing

We made some changes in the origin dataset; the first thing is to transform price data with log function since it has long tail which is not good to input into model. The second change was to remove the product data with price equals to 0 (0.052%) since we don't believe any product have zero price.

After that, we also separated the category name into three single classes for

[9] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

constructing features easier. Since we cannot input the string into model (Sklearn only support numerical features), so we use Label Encoder to encode the category features (DataFrameMapper[10] has been used). For missing data in product name or item description (including no description yet), we use 0 to represent their status.

3.2 Implementation

For processing txt data, we have two methods. The first method is easy to calculate the length of txt which we used in the benchmark model.

Another is better. First we do filtration since there are lots of symbols in the context. After that, we use Tf-idf method to generate numerical matrix (actually sparse matrix) for training model. In this part, we are confronted with the problem that the parameters' range is very large and we cannot decide which one is good, but some results from EDA gave us enlightens. From former data exploratory analysis, we knew 99.9% 3rd category contains at least 1.0 products, 99.9% 2nd category contains at least 13.339 products. 2nd category seems to be a good feature, so we set the min_df equals to 13 because we don't want to miss any 2nd category when we train the model. However, some 3rd categories contain only one product, which cannot be considered in Tf-idf. We also set the max_df equals to 0.99 since if a word in 99.9% product descriptions, we believe this item description don't contain any special words. The stop words is English for filtering and the max features is 50000 since the curse of dimensionality (actually, it doesn't surpass the max features line). After Tf-idf method, we got a sparse matrix.

Then we want to train the model, but the feature dimension is really large which takes really long time to train and easy to overfit the model since the higher model complexity. So we decided to reduce the dimension for training model easier, truncated SVD is a good method. The parameters n_components (the number of features you want to get) is 10 and we use 'arpack' algorithm since it works on sparse matrix. Finally, we selected the top three features because they almost explained 100% variance.

The last thing is to choose a good model. In this part, we try to use ridge, LightGBM, Gradient Boosting, Random Forest to predict the product price. The reasons about why we choose those models and what parameters we use have been written in 2.2 Algorithms and Techniques. The scores (RMSLE) of those model have been displayed in **Figure 8** and **Table 5**.

During the whole process, we are confronted with many problems. When we don't understand some parameters, we went to the official website of Sklearn, mlxtend, Lightgbm, which provide detailed explanations. The website like stackoverflow is also helpful.

[10] https://pypi.python.org/pypi/sklearn-pandas/0.0.3

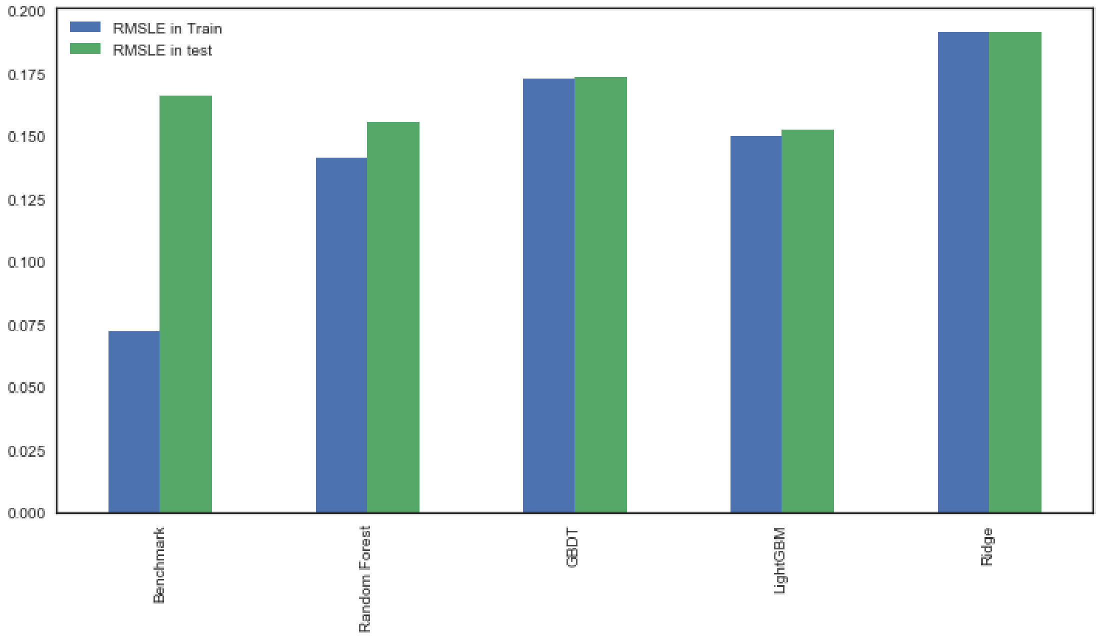

Figure 9: RMSLE in different models

According to the Figure 8, we could see LightGBM is better than the benchmark model and Ridge don't perform well. Except Benchmark model, all algorithms have not been overfitted based on the RMSLE on train and test dataset, which means the Tf-idf and SVD methods are effective for processing txt data.

Table 5: RMSLE in different models

Model	Train	Test	Time
Benchmark	0.0727	0.1663	60.606
RandomForest	0.1414	0.1560	54.406
GBDT	0.1732	0.1735	175.749
LightGBM	0.1500	0.1529	27.833
Ridge	0.1916	0.1917	0.503

The following **Table 6** shows the cross validation results (negative mean squared error) of different models, which proves that all of them can be generalized well.

Table 6: Cross Validation Results of five different models

Fold	1	2	3	4	5
Benchmark	-0.412	-0.410	-0.415	-0.414	-0.411
RandomForest	-0.358	-0.357	-0.359	-0.360	-0.355
GBDT	-0.457	-0.456	-0.459	-0.458	-0.454
LightGBM	-0.345	-0.346	-0.347	-0.347	-0.344
Ridge	-0.566	-0.563	-0.567	-0.567	-0.564

3.3 Refinement

In the former part, we find the best model LightGBM. In this part, we try to get a better model through adjusting parameters and stacking.

In order to getting better results, we used grid search method to adjust two important parameters in LightGBM, the colsample_bytree, learning_rate and subsample. Considering the training time is really long, we don't try to adjust all the parameters in this model and the range of parameters value is also limited. After grid search, we get the best learning rate is 0.15, the best subsample is 0.9 and the best colsample_bytree is also 0.9. The RMSLE score of this model is 0.1524 which is better than before.

In the last part, we try to use stacking method to refine the result. The simplest way is to calculate the average of each model's predictions, but the result is not good. We think it might be the bad model (Ridge) has negative effects on the whole performance, so we try to eliminate the Ridge model and recalculate the average. Although the score became better, it cannot bit the best RMSLE.

We also try to weight the results of each model with their RMSLE score, the results became better but still cannot bit the best LightGBM. Therefore, we try to use stacking method with mlxtend package. This time we also get better score but still cannot bit the best LightGBM model. Thus we believe the best LightGBM is the best model to do price suggestion job.

The following **Table 7** displays all the attempts for refining model.

Table 7: Model Refinement

Model	RMSLE
LightGBM	0.1529
LightGBM (better parameters)	**0.1524**
Average	0.1597
Average (no Ridge)	0.1549
Weighted average	0.1585
Weighted average (no Ridge)	0.1546
Stacking	0.1568
Stacking (no Ridge)	0.1563

4. Result

4.1 Model Evaluation and Validation

Finally, we get the best model, LightGBM. The n_estimators is 1000 which is not small or large, num_leaves equals to 31 (default, since it perform well, we don't adjust it), min_child_samples equals to 50 (not very small for avoiding overfitting and not large for getting better score), learning rate equals to 0.15 (adjusted by Gridsearch), colsample_bytree equals to 0.9 and subsmaple equals to 0.9 (both reasonable since we we use some features and some samples to train the base learner).

In order to evaluating the model, the first thing is to do cross validation, we get five negative mean squared error scores (-0.343, -0.344, -0.344, -0.345, -0.342), which means this model is robust.

We also change the train dataset by sampling. Every times we sampling 80% data and train the model, then evaluate the performance. The following **Table 8** shows each time RMSLE score, which don't have much chaanges.

Table 7: Sampling Train data for Validation

Sample	RMSLE
1	0.15269
2	0.15278
3	0.15276
4	0.15282
5	0.15286

According to the Table 8, I believed the results from this model can be trusted.

4.2 Justification

Comparing the model performance with benchmark model, I believe this model is much better. The best LightGBM model has 0.1487 RMSLE in train dataset and 0.1525 RMSLE in test dataset. However, the benchmark model has 0.0727 RMSLE in train and 0.1663 in test, which been regarded as overfitting. Besides, the Time of RMSLE is 36s compared with benchmark model 60s. Thus, we believe this model is better than benchmark and enough to solve the price suggestion problem based on the performance and the last validation part results.

5. Conclusion

5.1 Reflection

In this project, we tried to do price suggestion job with the dataset provided by Mercari Company in the Kaggle platform.

First we did data exploratory analysis work for understanding the characteristics about data and made some changes according to the EDA results. Since the distribution of price data has really long tail, we did log transformation for price in order to reducing the extreme values' effect. Then we found a small number of price have zero price value which doesn't make sense. So we removed those data points for avoiding the negative influence (zero price also cannot do log transformation). After all the efforts, the distribution of price seems really good. Therefore, we started to do feature engineering job.

The most difficult and interesting thing in this project was extracting txt features. At first, the length of txt was used to train the benchmark model. Then we used some advanced techniques like Tf-idf. However, after Tf-idf, we got a sparse matrix and nearly 30000 features. For simplifying the calculation, we used truncated SVD to reduce the features dimension which helped us to train the model with less time.

In the modeling part, we try to use different models, LighGBM, Gradient Boosting and Ridge. The training time of ridge is really short but it doesn't perform well. The Best Model is LightGBM with RMSLE equals to 0.15286. Then we try to refine the LightGBM model with adjusting parameters. After that the RMSLE drop down to 0.15246, which is better. Notice that we didn't adjust all the parameters considering the training time.

The Last refinement is to stack the models. Some simple attempts have been applied first like taking the average of each model's result and taking the weighted average. However, the model is still not good enough. Therefore, we tried more complicated method stacking with mlxtend package. Finally, we got a satisfied result, which perform better than the benchmark model.

In the validation part, we try to use cross validation to test the robustness and try to input different train to see whether this model's result change drastically. According to the results, the model is not overfitting and can be generalizes well. The following **Figure 10** shows the solution to this price suggestion problem.

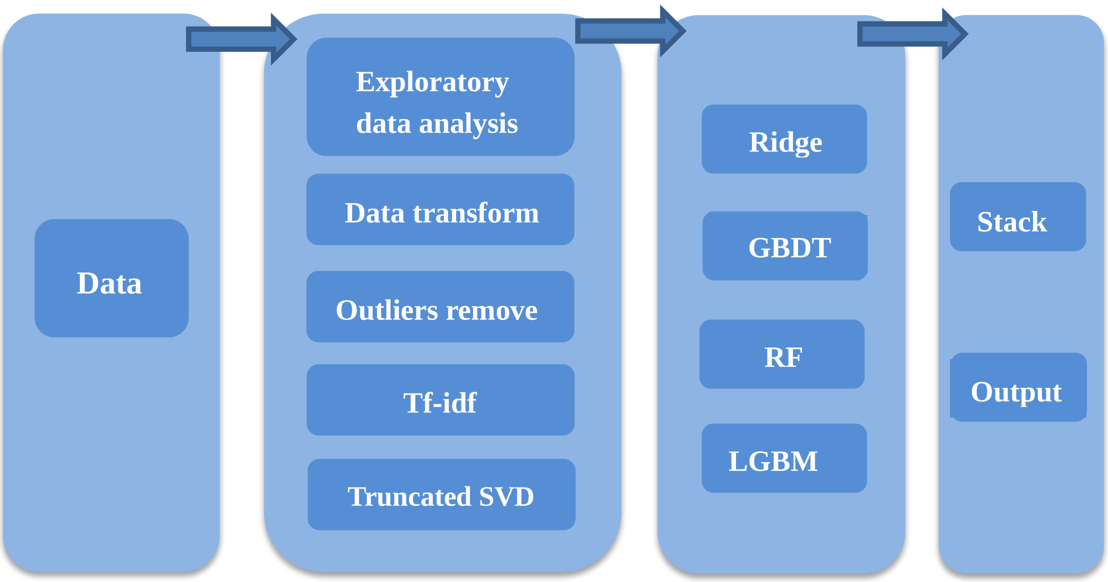

Figure 10: Solution for the price suggestion

5.3 Improvement

In conclusion, although we get good result of this question, some improvements can be done for achieving better results. We believe after those modifications, the model will perform better.

1. Try to use Neural Network methods

2. Try to adjust more parameters

3. Try to do better txt features extraction (some spelling error in txt content)