**ISOM 676 – Machine Learning II**

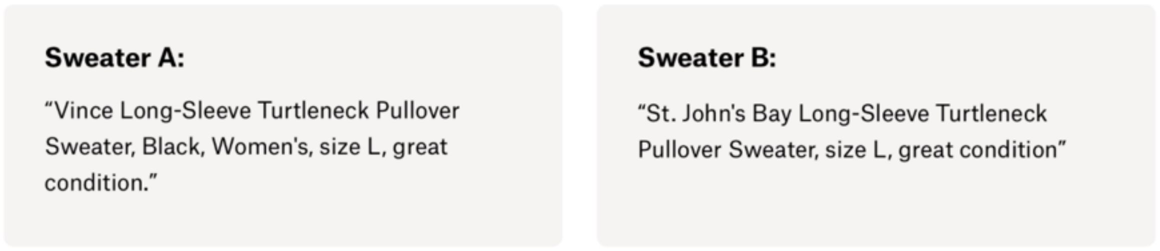
# Mercari Price Suggestion Challenge: Can you automatically suggest product prices to online sellers?

*Group 1: Abinav Bharadwaj, Evan Kang, Meng Wang, Kami Wu*

## Table of Contents

## Business Understanding

Online retailers and resale markets have consistently struggled with the task of gauging how much a given item is worth. Often times, there are multiple items with similar descriptions but, in reality, are wildly different in terms of value. The discrepancies in valuation often lie in the minute details. Take this example from the Mercari Price Suggestion Challenge itself:


**Sweater A:**

"Vince Long-Sleeve Turtleneck Pullover Sweater, Black, Women's, size L, great condition."

**Sweater B:**

"St. John's Bay Long-Sleeve Turtleneck Pullover Sweater, size L, great condition"

From an initial glance, both of these items seem practically identical. Both Sweater A and Sweater B are Long-Sleeve Turtleneck Pullover Sweaters, both are of large size and both are said to be in great condition. Seemingly, these two sweaters would have similar value and, therefore, would be similar in price. However, the Challenge tells us that one of these sweaters' costs $9.99 while the other costs $335. For two sweaters that seem comparable, the prices clearly are not. How could we know that one of the sweaters is significantly more expensive than the other? And how do we know which is which?

Mercari understands this problem in depth. Online purchase decisions are affected by many attributes. Some customers may be attracted by brand names while others fancy item descriptions. Human biases, when it comes to decision making, can cause customers to overspend on an item – paying above the item's value. This theory applies to e-commerce sellers, as well. They want their goods to sell at a price as high as possible, while remaining attractive to potential customers. Mercari would like to be able to offer pricing recommendations for the items people post on its community-powered shopping platform. However, finding that “right” price is difficult, because sellers are enabled to list anything, at any price, online.

In order to recommend a price at which sellers should list their item on Mercari’s platform, a data mining solution is necessary. The data mining solution will address the business problem by creating an algorithm that automatically suggests ideal prices, based on user-inputted text descriptions of their products. This adds business value by creating a way for sellers to know how best to price their “for sale” items based on their description of said items.

By implementing our model on Mercari’s community-powered shopping app, sellers will have a reference regarding how much they should charge for a specific item. Consequently, they are less likely to leave money on the table by either charging too low of a price or scare customers away by charging too high. With the algorithm properly applied and sales prices recommended, Mercari also looks to benefit from the boost in sales, due to platform taxes on sales and increased traffic to their app.

## Data Understanding

As mentioned in the Business Understanding, the goal of this supervised machine learning, data mining problem is to create an algorithm that automatically suggests the right product prices, based on user-inputted text descriptions of their products, including details such as product category name, brand name, and item condition. A sample instance from the dataset is {name:MLB Cincinnati Reds T Shirt Size XL, item_condition_id: 3, category_name: Men/Tops/T-shirts, brand_name: NA, price: 10, shipping:1, item description: No description yet}. The target variable is price, ranging from $0 (free item) to $2,009.

Data is obtained from Kaggle, The Mercari Price Suggestion Challenge. This data set has 1,482,535 observations and 7 columns. Basic information about the data is shown below:

| **Column name** | **Data type** | **% of null values** | **# of unique values** |
|---|---|---|---|
| name | text | 0% | 1,225,273 |
| item_condition_id | ordinal/categorical | 0% | 1: 640,549<br>2: 375,479<br>3: 432,161<br>4: 31,962<br>5: 2,384 |
| category_name | text/categorical | 0.43% | 1288 |
| brand_name | text/categorical | 42.68% | 4810 |
| price (target variable) | numerical | 0% | / |
| shipping | binary | 0% | 0: 819,435<br>1: 663,100 |
| item_description | text | Almost 0% | / |

*Table 1: Information about the dataset*

All variables do not include null values, except category name and brand name. Brand name has 6,327 null values, accounting for 0.43% of all values, while brand name has 632,682 null values, accounting for 42.68% of all values. Although item_description does not have any null values, about 5.6 % of the column values have "no description yet," which will later need to be replaced with null. The target variable, price, has a typical right skewed distribution with the mean of 26.73752 and a standard deviation of 38.58607 (the baseline for further modeling evaluation), thus it needs to be transformed into log for the later modeling process.

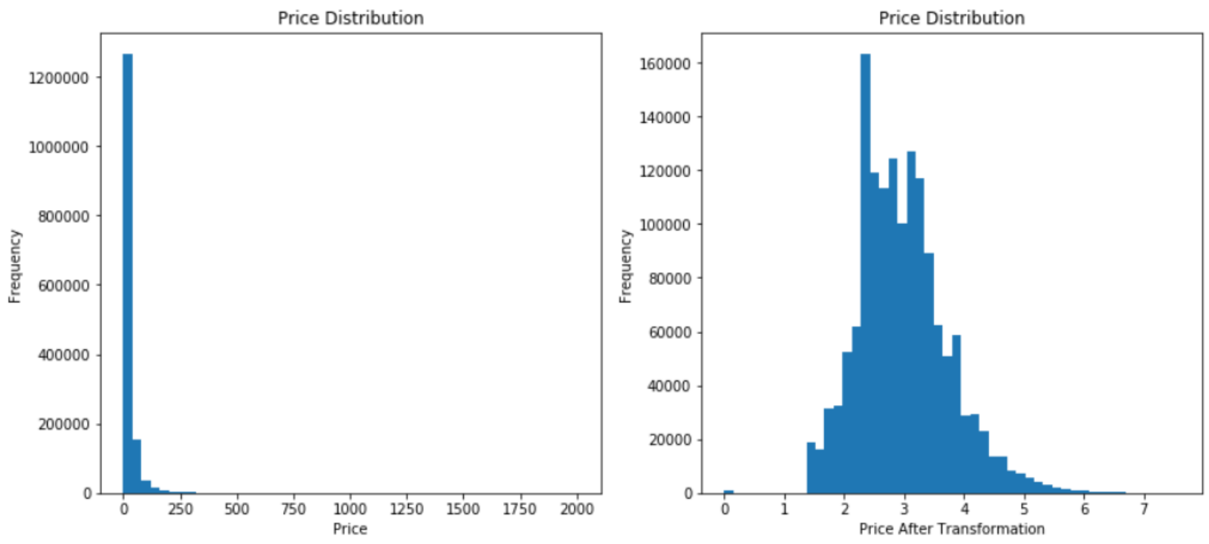


*Figure 1. The Distributions of Target Variable Price before and after log transformation*

Another interesting finding is that almost 55 % of the products have a shipping fee paid by the buyer. However, it is interesting that the price when buyers pay shipping fees is higher than the price when sellers pay shipping fees, which violates common sense. Unsurprisingly, the log-price displays the same pattern, which is shown in Figure 2.

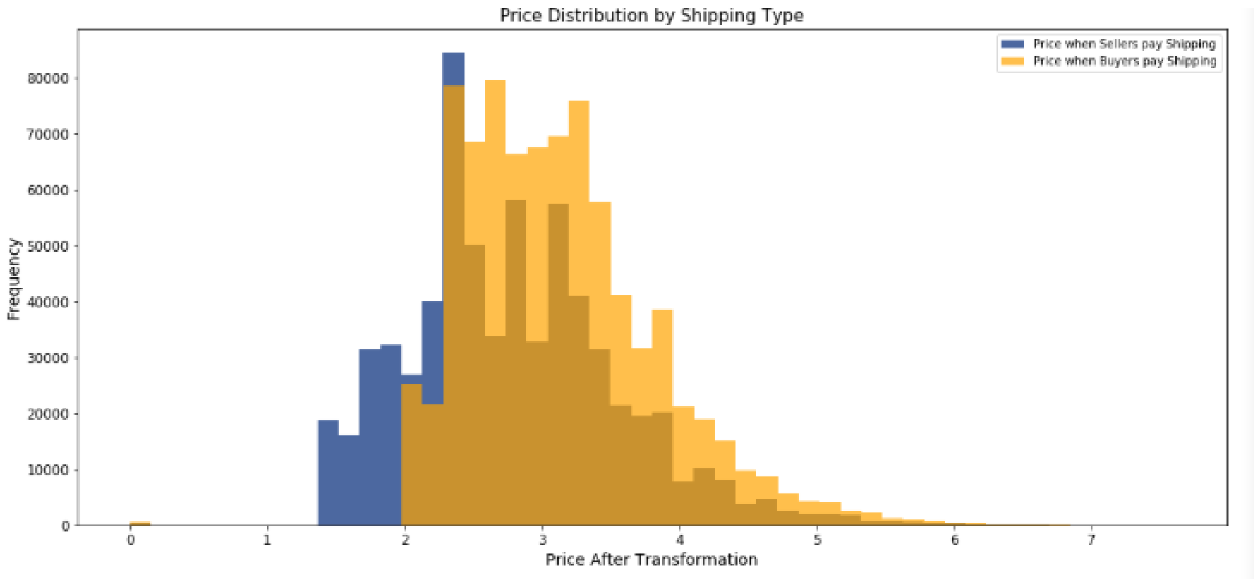


*Figure 2. The Distributions of log-Price for Sellers vs. Buyers paying for shipping*

## Data Preparation

After data understanding, we need to prepare data for the modeling process. The initial step of data preparation is dealing with missing values. In the data understanding part, we found that there are missing values in category_name, brand_name and item_description. Since these variables are all text-based, we re-categorize missing values as “missing.”

The second step is splitting training and test data sets. We use an 80:20 split validation here, other than for cross validation, because we believe cross validation will be too resource-consuming and time-consuming in this case. Following this, for the categories in brand_name or category_name in test data, which are not in training data, we re-categorize then as “missing,” so that we will not have new categories when we predict on test data.

The third and final step is transforming variables into a format that can be inputted into machine learning models. We realize we need to deal with 3 types of data - numerical, categorical/binary and text.

### Numerical Variables

We have only one numerical variable, which is our target variable: price. After conducting exploratory data analysis, we find variable price is right-skewed. Since this machine learning problem is a regression problem, we need to transform price in an attempt to make it more normally distributed. After using normal logarithm to transform the price, the distribution of the variable is more normal. In the following modeling process, we will use normal logarithm of price as the target variable, and we will transform the predicted value back when using evaluation measures.

**Categorical Variables**

In this data set, item_condition_id and shipping are two typical categorical/binary variables. Additionally, there are several other variables that can be regarded as either categorical or text, e.g. category_name and brand_name. Based on our understanding of the two variables, we decide to treat brand_name as a categorical variable and treat category_name as a text variable. The reason behind this is that category_name here has three levels and contains textual information about the product, which can determine the price of the product. Treating category_name as a categorical variable will likely cause us to lose important information.

For categorical variables, in order to input the data into machine learning models, in Python, we need to encode strings into numbers and then transform number-coded categorical variables into dummy variables.

Fortunately, item_condition_id and shipping are already number-coded, so we simply need to transform them into dummy variables.

For brand_name, we use Scikit-Learn LabelBinarizer to transform the variable from type string to a dummy variable in one step.

Something to note, since we are dealing with text data, and many zeros will be created after data cleaning process, we decide to use sparse matrices to store the data. Using sparse matrices to store data that contains a large number of zero-valued elements can both save a significant amount of memory and speed up the processing of that data.

**Text Variables**

For text variables, we have two different types - short text/phrase, such as name and category and long text/paragraph, such as item_description.

For short text, we tokenize the text and count the appearances of each word in said text. We use CountVectorizer from Scikit-Learn to do so and the outcome is directly transformed into sparse matrices.

For long text, we calculate the TF-IDF value of each token in item description, in order to find words that are characteristic for one item description within a collection of item descriptions. TfidfVectorizer from Scikit-Learn allows us to do this and the outcome is directly transformed into sparse matrices, as well.

After transforming all these variables, we stack them together and use them as the input for our models. However, after transformation, we end up with more than 73 thousand columns in our dataset. To pair the number of columns down, we delete all of the words that only appear once.

To further decrease the dimensions, we use singular value decomposition (SVD) to decrease the dimensions into certain numbers. However, after using SVD, the modeling result from the dimension-reduced data is not as good as the original data, and the training speed of the models is unacceptable, thus, we decide to maintain the large number of columns.

After completing these steps to prepare the data, we are ready to start the modeling process.

## Modeling

The machine learning problem we are dealing with is a regression problem. So far, the machine learning algorithms we have learned that can be used to deal with regression problems include linear regression, tree-based models and neural networks. Thus, we decide to try four different algorithms that could best suggest a price for sellers on Mercari- Ridge Regression, Lasso Regression, Light GBM and Neural Network.

Linear regression is the most common type regression model, so it is our first choice. We decide to use both ridge and lasso regression, in order to compare models, and to see which provides a better result. Linear regression offers a few primary advantages: it is comparatively fast to fit a linear regression model and it is easy to interpret.

For the tree-based model, we decide to use Light GBM. We choose Light GBM, because it can handle the large volume of the data, doesn't take as much memory to run and focuses on accuracy of the results. In our case, because we have a large volume of data, and we want to predict the price as accurately as possible, Light GBM is an ideal choice.

Our final model to recommend a sale price is the neural network, and we decide to use it because it can capture nonlinearities and works well with large datasets.

After building the four models, we decide to use stacking methods of ensemble modeling to combine the results from different models in hopes of having a better model performance.

For the modeling process, we would hope to use different bagging samples to train a single model and then combine the results together. If we were to develop these models further, we would push on with the bagging process.

When tuning parameters for each model we try to use Grid Search, in Python, but the Grid Search process is extremely time-consuming, even for a simple linear regression model (we attempted to run a Grid Search process on AWS but the process did not end after 12 hours). Thus, we decide to manually tune parameters for each model.

In order to evaluate our model results, we use RMSE because RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. Relating this back to our project, we do not want the prediction of price to be far off base from the real price, so RMSE is a better metric for this task.

The results of parameter tuning and model evaluations are as follows:

**Ridge Regression**

For ridge regression, we tune different values of alpha. Alpha in Scikit-Learn refers to the regularization strength, which determines the generalization of the model. The best alpha we get is 0.9, which leads to an RMSE on the test set of 27.9359. The alpha we tune and the corresponding RMSE is shown below.

| **Alpha** | **RMSE** |
|---|---|
| **0.9** | **27.93588307** |
| 1 | 27.9359809 |
| 0.8 | 27.93647809 |
| 0.7 | 27.9377059 |
| 0.5 | 27.94372472 |
| 0.2 | 27.96762046 |
| 0.05 | 27.99760053 |

*Table 2: Parameter tuning and outcome of ridge regression*

**Lasso Regression**

For lasso regression, we tune different values of alpha, as well. Originally, we believed lasso regression would perform better than ridge regression, because it includes automatic feature selection. However, the RMSEs from lasso regression are generally 10 points higher than those of ridge regression, which is close to the standard deviation of price. This means lasso regression models perform worse than ridge regression models. Because the result of lasso regression is poor, we choose not to waste our time tuning parameters. The best alpha we get is 0.01, which leads to an RMSE on test set of 38.0820.

**Light GBM**

For Light GBM, we tune learning rate, maximal depth of trees and lambda. We use RMSE as the loss metric, and set application as regression, which will allow Light GBM to perform the regression model. Here are the results of the Light GBM model:

| **Learning rate** | **Max depth** | **Lambda** | **RMSE** |
|---|---|---|---|
| **0.3** | **5** | **1** | **28.437102231623157** |
| 0.3 | 5 | 2 | 28.63552293751575 |
| 0.05 | 15 | 1 | 28.656794881915232 |
| 0.75 | 5 | 1 | 29.43896077410975 |
| 0.05 | 5 | 1 | 31.42046778313502 |

*Table 3: Parameter tuning and outcome of light GBM*

**Neural Network**

For the neural network model, we try both "deep" and "wide" networks. "Deep" networks are neural nets with multiple hidden layers, while "wide" networks are neural nets with more neurons. The activation function we use for hidden layers is ReLu, and the activation function we use for output layer is linear. ReLu is the most popular activation function used in deep learning models, and the linear activation function is the proper choice for regression models.

Additionally, we found that wider neural nets work better than deeper neural nets for this problem. The best neural net we get is a neural net with 1 layer and 50 neurons, which leads to an RMSE of 26.3330. The parameters we tune and the corresponding RMSE is shown below.

| **Layer** | **Unit** | **RMSE** |
|---|---|---|
| **1** | **50** | **26.33298874** |
| 1 | 10 | 26.83587488 |
| 3 | 4 | 27.18324601 |

| | | |
|---|---|---|
| 2 | 4 | 27.913051708396132 |
| 3 | 10 | 28.93781354 |

*Table 4: Parameter tuning and outcome of neural networks*

After outputting the best model for each tested algorithm, we use stacking methods to combine the outcomes together. In order to do this, we use average, weighted average and linear regression. We opt not to use the prediction from lasso regression because the prediction is much worse than that of other models and we believe that including it into stacking models will have a strong negative impact on the ensemble model. With all of that in mind, the final predictions we input into stacking models are predictions from the best ridge model, best light GBM model and best neural net.

For the average stacking model, we take the average of the three predictions and get an RMSE of 26.5660.

For the linear regression stacking model, we fit a simple linear regression on the three predictions and get an RMSE of 25.9960.

From the linear regression model, we can get the coefficients of each prediction. We then round the coefficients to 2 digits and use the number as a suggestion of the weights in the weighted average model. The weighted average model gives us an RMSE of 26.0179.

Reflecting on the modeling process, ridge regression, light GBM and neural network models all perform well in this task, whose best model give us a RMSE dramatically lower than the standard deviation of price, the raw target variable. Dimension reduction methods, including SVD and lasso regression, do not work well for our project, so it is better that we keep all the information we have to input into models. Ultimately, using stacking methods to combine the outcomes from the various algorithm we tested dramatically increases the performance of the prediction, and the stacking linear regression model gives us the best outcome.

## Evaluation

First, our approach considers the item description comprehensively. By transforming the item_description column using TF-IDF method, we are able to arrive at over 50K+ columns containing the words in the description, with the inverse frequency of their appearance. As a result, the information inputted by the seller, regarding the item description, will be understood well.

Second, by dividing the category_name column into three separate columns, we then have 10 main categories, 114 first sub categories and 871 second sub categories. The mass coverage of the category names provides us the capability of predicting a wide range of items, as shown in Figure 3.

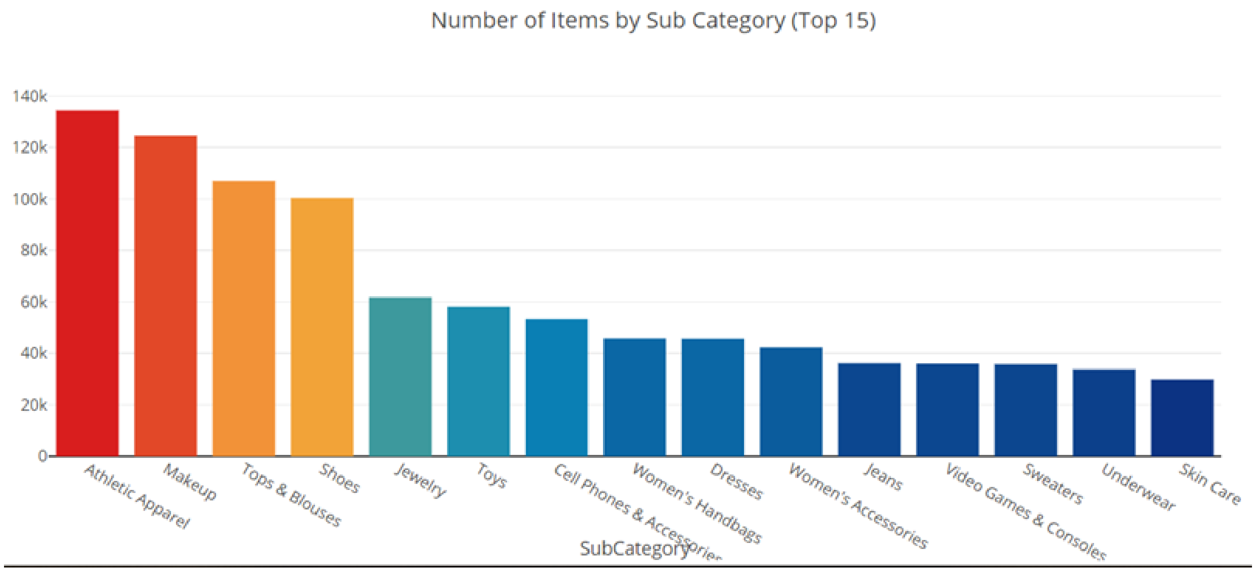


*Figure 3: Number of Items by Sub Category*

Third, our model is better at predicting the items in the main category “Women” than those in other categories. Because nearly 50% of the items (more than 650K) from the original dataset are considered “Women” goods, the model is naturally trained to have a better prediction accuracy in “Women” category. On the flip side, “Homemade,” “Sports & Outdoor,” and “Other” categories have a total number of items less than 100K in our training dataset, causing the performance of the model in predicting these three categories to be relatively poor.

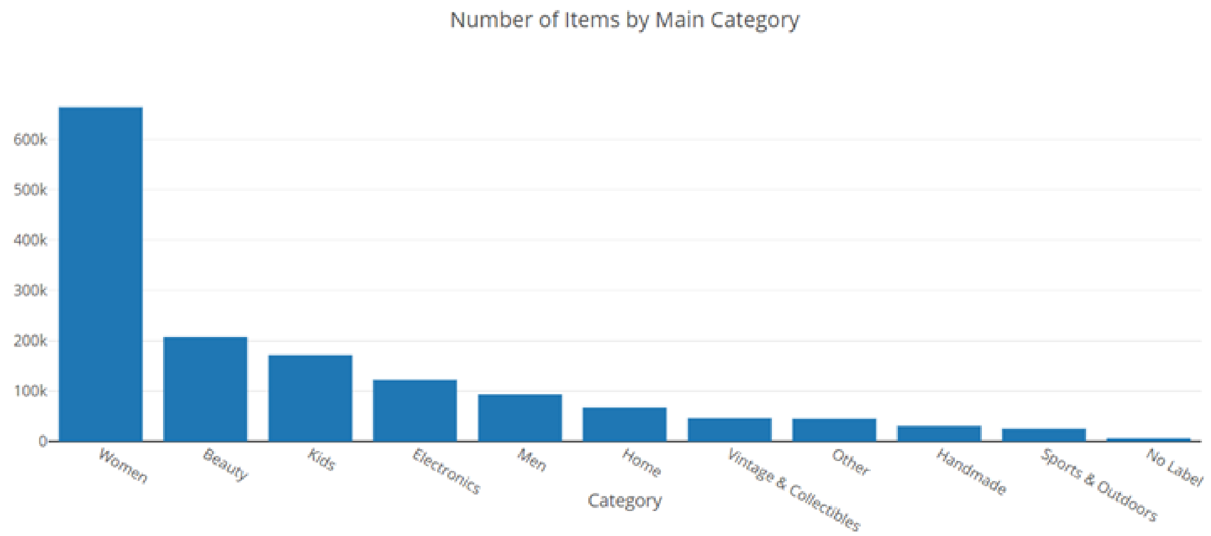


*Figure 4: Number of Items by Main Category*

Last, the brand names column has more than 4,000 unique values, so it would appear that it is a good suggestion that our model takes brand into consideration. However, considering that over 630K (45%) items have missing brand values, only 17 brands appear more than 10,000 times in the data, and 70% of the brands appear less than 15 times, we can clearly see that item brand does not provide a good indication of the price.

Although the model result promises us a good prediction of the price given the item information, we face the great concern that missing too much information related to brand will lower the prediction accuracy in the real business world. From a common sense perspective, nearly identical products in different brands cover a large variance in price. For example, a Chanel handbag can be priced around $100,000 while a handbag from Walmart may only be worth $10. Though the item description can tell us a part of the value of the item, we still cannot ignore the impact that lacking brand information will have on our price prediction.

Inspired by this, we suggest that Mercari use our model on a trial basis, so that the platform can evaluate the performance of the model and allow for wider "range" in suggested price it recommends to the sellers.

When the seller enters the category of item, name of the item, condition of the item, whether or not shipping is offered and finally the detailed item description, the seller will ultimately be provided with a suggested price range at which to list the item, with a plus or minus 20% range for the suggested price. For example, if the model spits out a recommendation of $50 USD, then the seller will see "suggested list price: $40 - $60" in their application.

However, sellers are still able to set the item price on their own. That is to say, the seller can either follow the price suggested by the Mercari app or choose their own list price outside of the given range. Mercari can then track performance across all price settings and compare with those listed in the suggested price range. If the platform finds that a large portion of sellers set the list price out of the recommended price range, and the difference between seller set price and the recommended price range is drastic, then our model surely requires modification. However, if we find that a majority of the sellers follow the recommendation and set a list price within the price range, and the few out-of-range prices are still close to our prediction, then Mercari will know that our model is meaningful to their business and they can opt to utilize the model regularly on their app after the trial period.

An additional method of evaluating the model would be through an A/B test. Mercari can extract 2 groups of items with the same number of items in each group. Items in Group A are provided a suggested price while items Group B, considered the control group, will have their prices set without any suggested price. From here, we can then compare the length of time that it takes for items in each group to sell. If the length of time it takes for items in Group A to sell is significantly shorter than that of Group B, we can conclude that our model, which suggests the price at which to sell, is meaningful to the platform's business. Shortening the time it takes for an item to sell means sellers will having better experience using Mercari's app, which will lead to sellers being more willing to use the platform, and thus, driving more traffic to the platform.

## Deployment

Historically, listing an item for sale on the resale market, especially for a used item, has been mostly guesswork when it comes to pricing. The seller tends to factor in original price paid, condition of the item, and, when applicable, price listed by competition on the same platform. From there, sellers have the tendency to create some arbitrary value in their heads of the item's worth. As a seller seeks to maximize return from the item for sale, this valuation is often much higher than the price any buyer is willing to pay. This causes the seller to sit and wait anxiously at first, and then they likely have to decide whether or not to lower the price of their item. If so, by how much should the price be lowered? The result of the stacking model and ultimately, the deployment on Mercari's community-powered platform, work to aid in solving this problem. As mentioned in the Business Understanding, when deployed the project will be able to recommend a price at which the seller should list an item for sale, based on the seller inputted, text description of the item. Provided that the model is accurate in its evaluations, the deployment of this feature on Mercari's platform will lead to a more friendly user experience for the sellers, a decrease in the time it takes for sales to occur, increasingly fair prices for buyers, and an overall boost in quality for all involved, driving up traffic to the app.

However, deploying the model on the application is not without potential issues and risks. Right off the bat there is the concern that the lack of information regarding brand name data inputted into the model, for training purposes, could lead to gross mispredictions of an item's worth. Another risk to consider is that sellers could figure out patterns and certain keywords that drive up the recommended sales price outputted by the model. If the sellers are able to exploit this and drive up the entire market price for a given item, then buyers bear the impact as they would now have to pay more for the item than originally expected. In order to mitigate this risk, it is important to remain vigilant in feeding data and updating the model from Mercari's platform. Continued learning for the model would decrease the likelihood the model is duped through the exploitation of certain words or phrases.

Finally, it is vital to consider any potential ethical risks. In order for the model to suggest a sales price for any item, the model receives the category of item, name of the item, condition of the item, whether or not shipping is offered and finally the detailed item description. Across thousands and thousands of items listed on the app, this is a significant amount of data being collected. Mercari could use this data to then target sellers, by recommending they purchase items with similar features to the ones they listed for sale. Perhaps these promoted items would remind the sellers why they purchased the item in the first place, increasing the likelihood they then buy the items. As long as Mercari stays above board regarding their data policy, and only personally data mine with the user's consent, these ethical concerns are minimal to nonexistent.

The deployed data mining solution to Mercari's business problem will ultimately increase the satisfaction of users on the platform, drive more traffic to the app, and in turn, increase the capital gained by Mercari through platform taxation of sold items.

## Works Cited

Jj. “MAE and RMSE - Which Metric Is Better?” *Medium*, Human in a Machine World, 23 Mar. 2016, medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.

Mandot, Pushkar. “What Is LightGBM, How to Implement It? How to Fine Tune the Parameters?” *Medium*, Medium, 17 Aug. 2017, medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc.

“Mercari Price Suggestion Challenge.” Kaggle, www.kaggle.com/c/mercari-price-suggestion-challenge.

“Select a Web Site.” Computational Advantages of Sparse Matrices - MATLAB &amp; Simulink, www.mathworks.com/help/matlab/math/computational-advantages-of-sparse-matrices.html.

## Appendix

| **Team Member Name** | **Contribution** |
|---|---|
| Abinav Bharadwaj | Report: Research into Mercari's platform and focus mainly on Business Understanding and Deployment sections but also formatted and edited entire report, including fixing spelling and grammatical errors<br><br>Slides: Focused mainly on Business Understanding and Deployment but also formatted and edited entire deck, including fixing spelling and grammatical errors |
| Evan Kang | Code: Data exploration and tried topic modeling; Parameter Optimization; Evaluation<br><br>Report: Evaluation<br><br>Slides: Evaluation |
| Meng Wang | Code: Data exploration, Data Preparation (only for RNN), Recursive Neural Network modeling and its evaluation<br><br>Report: Data Understanding<br><br>Slides: Data Understanding |
| Kami Wu | Code: Data exploration, Data preparation, Modeling and Evaluation (ridge regression, lasso regression, light GBM, neural network, stacking; parameter tuning)<br><br>Report: Data preparation and modeling process<br><br>Slides: Data preparation and modeling process |