Final Project Report

Sergio Castillo

April, 2018

1 Project Definition

1.1 Project Overview

E-commerce has changed the manner in how people buy and sell products. In the past, a person had to go to the store, chose the desired item and pay. Nowadays, consumers do not need to physically inspect the product to acquire it. Instead, they have developed patterns based on the information that retailers provide on their website[1]. This data has helped sellers to understand their client's behaviour and use it for different purposes. Such as, forecasting sales[2], customer's segmentation[3], detect trending items [4], and products' price prediction[5][6][7] which is the area that this study will cover.

Price prediction has been a subject of study for a long time in the economy, specifically in the stock market[8]. Different techniques have been proposed, from probabilistic methods[9] in the 70's to ensemble techniques combining neural networks(NN) with swarm optimisation[10], or deep learning [11]. Currently, research to predict prices has been adopted in more areas as, travel[12][13], real estate[14][15], agriculture[16], or retail[1][2][5][6]. In this research, we will focus only in the retail industry.

The importance of predicting prices in the e-commerce retail industry can be divided into 3 areas: 1) for the customer, 2) for the retailer, and 3) goods manufacturers. First, the customer can obtain a baseline price or an expected price for a certain item[17]. Second, the seller can use the prediction to define a price when introducing a new product, or forecast sales[2]. Last, the manufacturer can use the information for a deeper study that could include the total number of items to be sold, and define a strategy in their production line to estimate the number of raw materials needed for the expected demand[18].

The objective of this study is to have hands-on experience in a dataset provided by Mercari, one of the leading retail companies in Japan via a Kaggle competition. Also, as part of the contest, and the motivation to earn a monetary prize, for me, it is important to compare my models with the best people in the data science community that can only be found in Kaggle. The sample dataset can be found here: https://www.kaggle.com/c/mercari-price-suggestion-challenge/data

1.2 Problem Statement

Mercari, the biggest shopping app in Japan, knows that price prediction within the e-commerce retail industry is a complex task. For instance, two sweaters with a similar description have a difference in their prices as big as $345.00 USD. If we scale this example to the thousands of products that are sold daily, the problem becomes more complex. In order to help their clients, Mercari wants to suggest prices to the sellers when uploading their products. Thus, the challenge in this competition is to develop a model that could recommend a price with a high level of certainty. [19]

To solve this problem, this project is divided into six parts. First, there is an analysis of each feature, the target value and the relation between them. Also, prices equal to zero are removed and the remaining transformed with the log function. The second part corresponds to the preprocessing step, where the entire features are converted to a sparse matrix using different natural language processes. Third, a benchmark is conducted using the KNN for regression with a parallel prediction to decrease time. In the fourth section, a feature selection is done, and the four models proposed

(i.e. Decision tree regression, Linear Regression, Support Vector Regression, XGBoost) with their default parameters are developed. Fifth, from the previous section we select the best model (i.e. XGBoost) to tune its parameters and decrease the error. Last, a conclusion and future work are stated.

1.3 Metric

The evaluation metric selected by Mercari is the **Root Mean Squared Logarithmic Error**. Which is calculated as follows:

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

Where:

- ε is the RMSLE value (score)
- n is the total number of observations in the (public/private) data set.
- p_i is your prediction of price
- a_i is the actual sale price for i
- $log(x)$ is the natural logarithm of x

The reason to choose such metric is not provided in the competition's section. However, it might be possible that the decision was based on the benefits that this metric provides. For instance, it penalises more the underestimates than the over estimates[20]. Because of the square, it also takes more importance to a large error than the small ones[21]. And Additionally, by using the logarithmic from the values, it reduces the difference between the maximum and minimum price, which tend to generate a Gaussian distribution.

2 Analysis

2.1 Features

Briefly, the training set is composed of 1,482,535 instances with seven features and the target price. The seven features are:

- **train_id**, a unique consecutive key number to identify each instance.
- **name**, a brief summary of the product.
- **item_condition_id**, a descriptive value, from one to five to qualify the condition of the item.
- **category_name**, the corresponding product's section, it is divided into three levels, from a general to a specific one.
- **brand_name**, the item's brand.
- **shipping**, whether the product prices already include the delivery or not.
- **item_description**, a detail explanation of the product's characteristics.

Below, Table 1, presents the data type and if the feature contains empty values. In this section, each feature is discussed in detail. There are discussions about the possible impact in the model and whether it could be useful or not. Also, interesting findings from the data are reported.

Feature	Data Type	Contains Nulls?
train_id	int64	False
name	object	False
item_condition_id	int64	False
category_name	object	True
brand_name	object	True
shipping	int64	False
item_description	object	True

Table 1: Presents the data type and whether the feature has null values or not.

Figure 1 shows a sample of the data in each feature. It can be seen that brand_name and item_description contains null values. train_id, item_condition_id, and shipping are descriptive values. Meanwhile, name, category_name, brand_name, and item_description are strings. Price is the only decimal value.

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

Figure 1: Sample data from the training set with the entire features and the target value.

2.1.1 Price - Target Value

Figure 2, presents the price distribution. Overall, the range goes from $0.0 to $2,009.0. It is clear from the histogram that the distribution is not normal. Instead, it has a shape of a one-sided F distribution or a positively skewed shape with a wide dispersion.

Using the function called ”describe()” in the data-frame more details from the price were obtained. The mean price is $26.73, with a standard deviation of $38.58. As seen in the graph below, most of the prices are between $0.0 and $200.0. Specifically, up to 98% of the values does not cost more than $122.0. The 75% of the items have a price less than $29.0. Figure 3, provides a zoom of the distribution limiting the range to $200.0.

In addition, Figure 3, shows that 874 instances have a price equal to $0.0. These items were removed from the training set for two reasons. First, in practice and as a policy, Mercari does not allow you to publish a product with a value of $0.0. For instance, in the subset that includes the items with prices equal to zero, we can find jeans Levis, old navy shirts, Nike shoes, to name a few. Obviously, none of these items could cost something close to zero. Second, these items could add noise to the model and increase the error for future predictions.

As part of the best practices, we always want to have or try to have a normal distribution. Hence, to narrow the space between the cheapest and most expensive price, and reshape the graph to have a normal form, the value was transformed

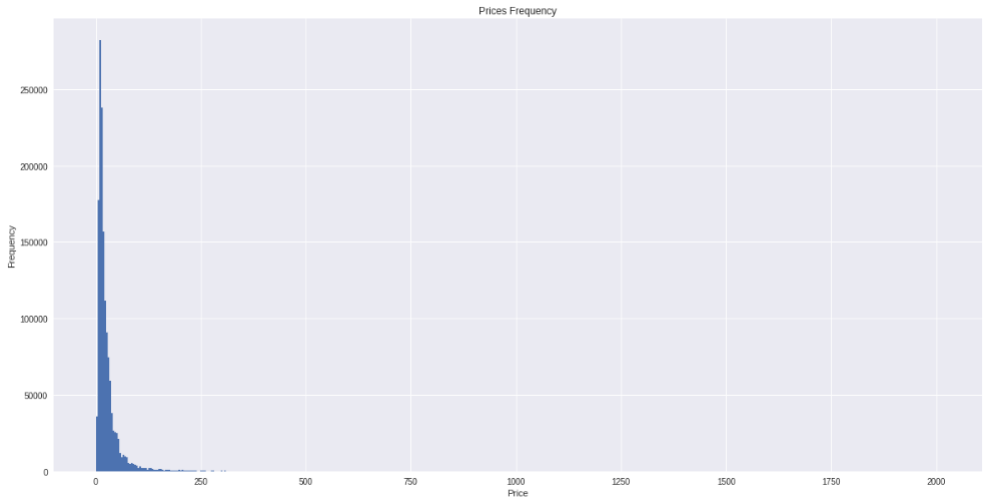


Figure 2: Price histogram with the entire instances.

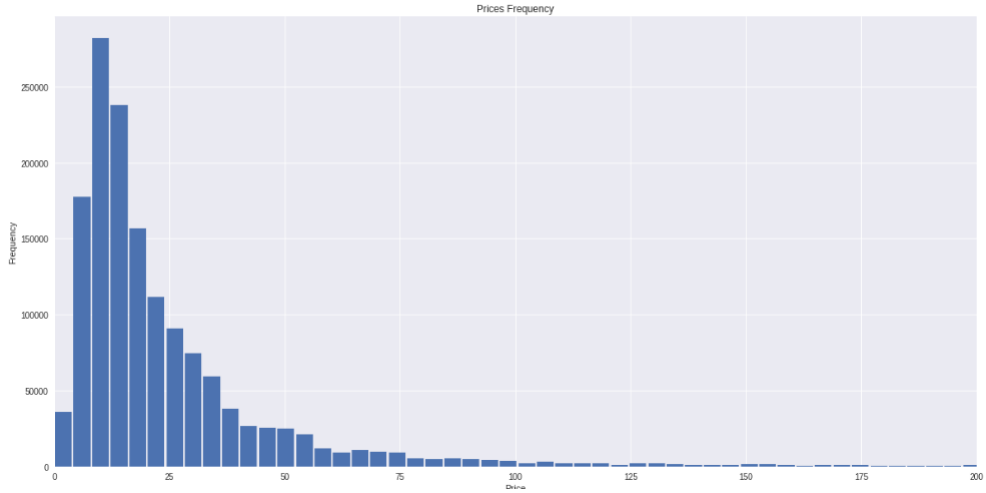


Figure 3: Price with items that cost less than or equal to $200.0.

using the numpy's log function plus one or:

$$logprice = log(price + 1)$$

.

Figure 4 illustrates the price distribution after transforming it with the log function. As seen, the range was reduced from 1.38 to 7.60. The shape even when does not have a Gaussian bell yet, was moved closer to the middle. Now, the mean is 2.98 with a standard deviation of 0.7459 with 75% of the values having a log price up to 3.40.

2.1.2 item_condition

Item condition is a categorical feature. It is a score from one to five that describes how the item's condition is. The range goes from "new" to "poor." The complete explanation can be found in this link. We are going to verify the products' condition based on this feature.

As is observed, the bar chart above indicates that most of the items sold are either new or in good conditions. According to the graph, almost the new condition(i.e. id 1) is equal to the sum of condition two and three. The first one encloses 640,246 instances, meanwhile, two and three together counts 807,088. These three options represent more than the 97%

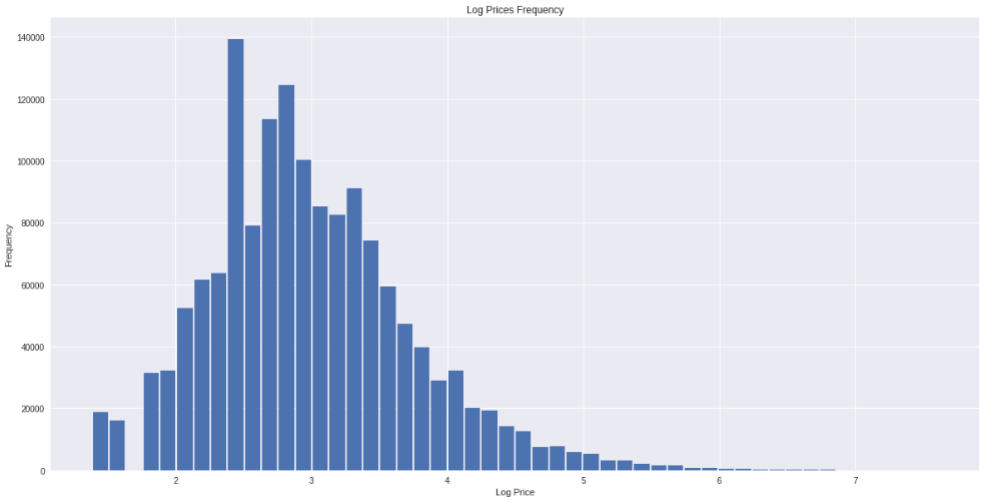


Figure 4: Price histogram with log(prices+1)

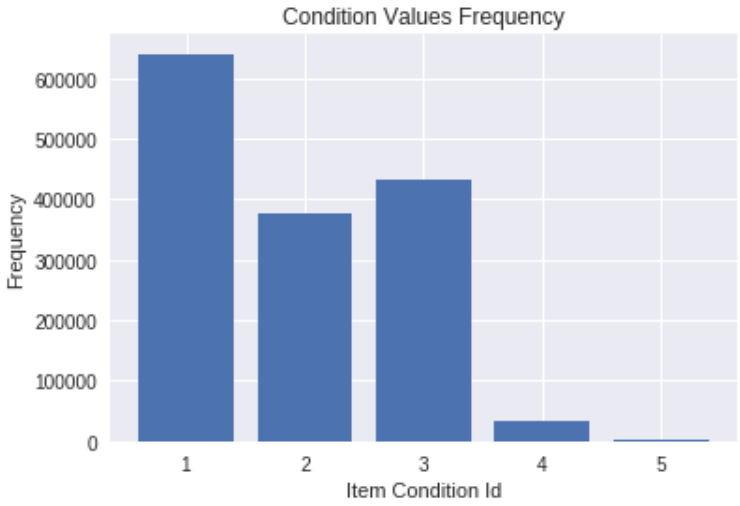


Figure 5: Item condition frequency

of the entire training set. The rest of the ids have 31,945 and 2,382 for four and five respectively.

From the box plot above, it can be seen that the mean log price for the five conditions is similar, around three. The inadequate range(IQR) from the first four conditions are between 2.5 and 3.2. In contrast, the third quartile from the condition with value five is higher, above 3.5 approximately, which could mean that products with this condition are more expensive. It might be possible that even when condition five means poor, these products are antiques or collectible. Moreover, it is clear from the chart that the price constantly declines from the condition one to four constantly and climbs around 0.5 in the last one.

2.1.3 shipping

At first glance, in figure 7a, we discover that the distribution is similar between them. Items without free shipping are 818,876 against 662,785 that include it. Moreover, contrary to the common thought that free shipping products already ”include” the delivering fee in the final price, the histogram above(i.e Figure 7b) proves the opposite. The products with free shipping are cheaper than those where the client has to pay.

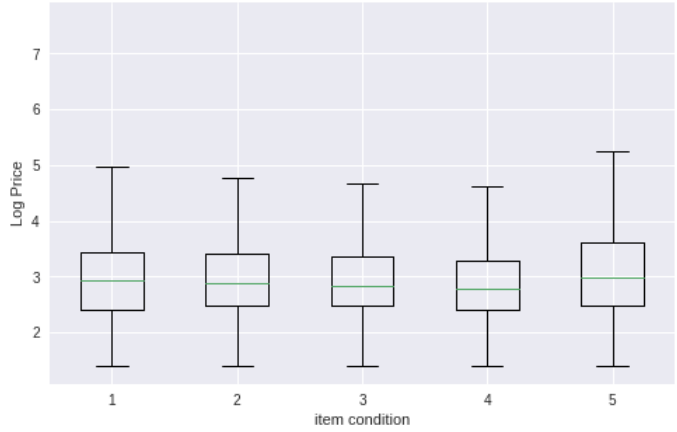
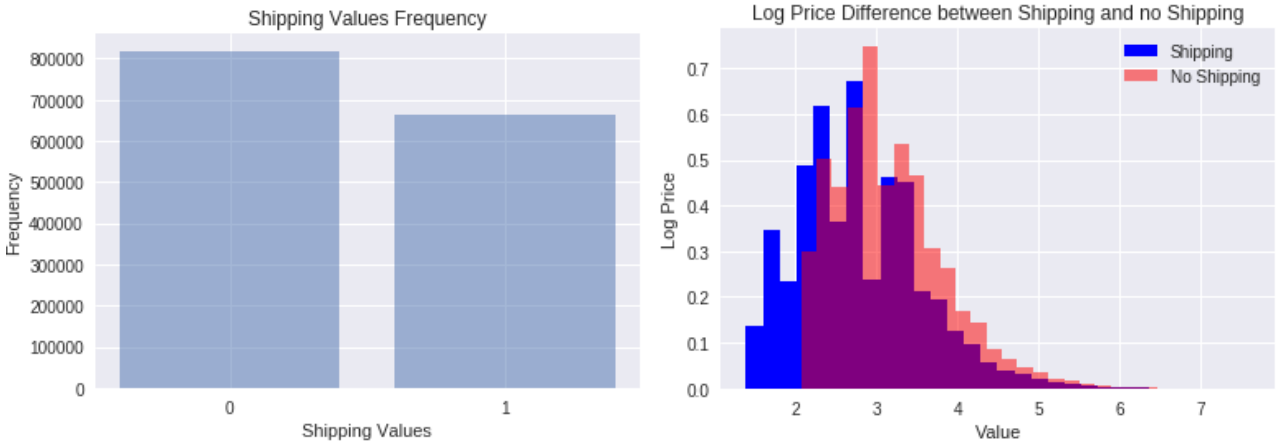


Figure 6: Box plot that presents the log price dispersion per condition



(a) Shipping frequency by option

(b) Log price by shipping option

Figure 7: Shipping feature

2.1.4 category_name

The category_name is a descriptive feature formed by three strings separated by a slash. This analysis starts with the entire string and then separates. Also, compares each category part with the log price to review the price variance among them.

In general, the total number of unique categories is 1,287 with only 6,314 items without one, which represents less than the 0.5% of the training set.

Figure 8 indicates the name of the top 30 categories in the left with the corresponding number of items in the right, representing the 42% of the total set. From the 1287 categories, we note that from the top 10, 8 categories sell products for women. Only two categories are not for that segment, ”Electronics/Video Games & Consoles/Games”, and ”Electronics/Cell Phones & Accessories/Cases, Covers & Skins”.Moreover, from the entire 30 first categories, just six categories are not intended to sell products to females. This could tell us that Mercari is focused on selling products to women, could help them to develop marketing campaigns for them, or can focus their design on them.

In contrast, the chart above presents the top 30 categories with the highest median price. Comparing the two lists, we can detect that the categories with high prices are not related to woman items. The highest price is located in the ”Vintage & Collectables/Antique/Furniture” category with a price of $195.00, however, it was just one item. Then, there

Women/Athletic Apparel/Pants, Tights, Leggings	60152
Women/Tops & Blouses/T-Shirts	46349
Beauty/Makeup/Face	34320
Beauty/Makeup/Lips	29901
Electronics/Video Games & Consoles/Games	26547
Beauty/Makeup/Eyes	25200
Electronics/Cell Phones & Accessories/Cases, Covers & Skins	24668
Women/Underwear/Bras	21254
Women/Tops & Blouses/Tank, Cami	20270
Women/Tops & Blouses/Blouse	20269
Women/Dresses/Above Knee, Mini	20068
Women/Jewelry/Necklaces	19750
Women/Athletic Apparel/Shorts	19518
Beauty/Makeup/Makeup Palettes	19091
Women/Shoes/Boots	18853
Beauty/Fragrance/Women	18614
Beauty/Skin Care/Face	15825
Women/Women's Handbags/Shoulder Bag	15317
Men/Tops/T-shirts	15095
Women/Dresses/Knee-Length	14763
Women/Athletic Apparel/Shirts & Tops	14730
Women/Shoes/Sandals	14655
Women/Jewelry/Bracelets	14487
Men/Shoes/Athletic	14247
Kids/Toys/Dolls & Accessories	13951
Women/Women's Accessories/Wallets	13607
Women/Jeans/Slim, Skinny	13377
Home/Home Décor/Home Décor Accents	12993
Women/Swimwear/Two-Piece	12754
Women/Shoes/Athletic	12650

Figure 8: Top 30 categories with the number of items

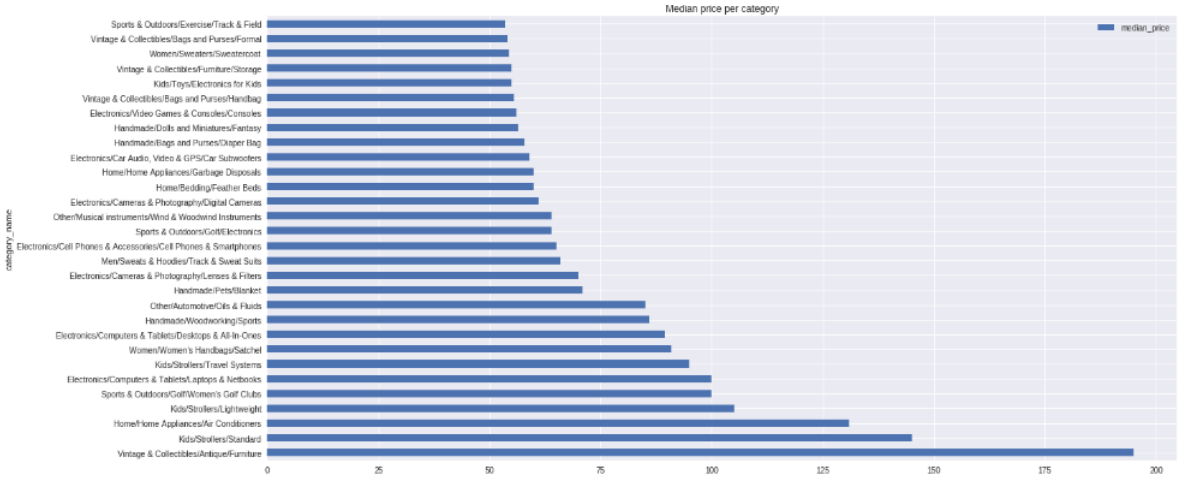


Figure 9: Top 30 categories with the highest median price per item

is a considerable reduction with ”Kids/Strollers/Standard” having a median price of $145.0. However, the prices are far from the median(i.e. $29.8), these top 30 categories only represent the 1.45% of the training set.

In conclusion, the analysis with the entire string, further action in the preprocessing might be conducted. For instance, we can train a model with and without the outliers and compare the results. It might be possible that the categories with few items only produce noise to our model or not.

Splitting the string by the slash, the total categories in the first level are 10, the second level has 113 unique values, and the last one has 871. Figure 13, lists the name of each category at first level, including the average price and the number of products per group.

Similar to the previous analysis, in this case, the top two categories are targeted for women. Surprisingly, the top category by the number of items(i.e. Women) has around three times more products than the second one called ”Beauty” with 663990 and 207725 respectively. Furthermore, the total number of items in the rest eight categories which are 603,632 is less than ”Women.” The lowest median price is $12.0 for ”Handmade,” and the most expensive is ”Men” with $21.0.

It might be possible that because almost half of the information corresponds to only one category or gender. Two models could be created, one for women related products and the other for the rest of them. Or maybe create a model per

category1	median_price	count
Women	19.0	663990
Beauty	15.0	207725
Kids	14.0	171555
Electronics	15.0	122632
Men	21.0	93609
Home	18.0	67831
Vintage & Collectibles	16.0	46519
Other	14.0	45329
Handmade	12.0	30835
Sports & Outdoors	16.0	25322

Figure 10: Total categories in the first level with the number of items and the median price.

these 10 categories.

2.1.5 brand_name

Most of the times, a brand defines the price. That is, you expect a certain price for a product from a specific company, compared to a similar product with another corporation. For instance, it is not the same price of an iPhone X than the Samsung Galaxy or the price of a Gucci handbag with a similar one but with an unknown manufacturer. Analysing this feature might help to understand the price, the most common, or the total number of brands and the number of products without one. In the end, we could define whether it is important or not, and the relationship with the price.

brand_name	median_price	count
AA Aquarium	3.0	1
Old Glory	3.0	1
Kae Argatherapie	3.0	1
A.B.S. by Allen Schwartz	3.0	1
Ask	3.0	1
Genica	3.0	1
Play MG	3.0	1
Archie Comics	3.0	1
Feetures!	3.0	1
Peanut Shell	3.0	1

(a)

brand_name	median_price	count
Demdaco	429.0	2
Auto Meter	344.0	1
Proenza Schouler	315.5	4
Oris	300.0	1
Longines	254.0	1
Blendtec	250.0	5
Dainese	230.0	1
Frédérique Constant	224.0	1
David Yurman	220.0	242
Vitamix	205.0	9

(b)

brand_name	median_price	count
PINK	20.0	54072
Nike	22.0	54006
Victoria's Secret	19.0	48011
LuLaRoe	29.0	30995
Apple	22.0	17314
FOREVER 21	12.0	15178
Nintendo	20.0	14998
Lululemon	39.0	14550
Michael Kors	49.0	13916
American Eagle	14.0	13245

(c)

Figure 11: a)Top 10 Brands with lowest median prices and the total number of items. b) Top 10 Brands with highest median prices and the total number of items. c) Top 10 Brands with more products and median prices.

Overall, there are 4.807 different brands. Still, only 57% of the products have a brand, the rest have no value.

Table 11c, shows that the most common brands have a range median price between $12(i.e. FOREVER 21) and $49(i.e. Michael Kors). The number of items is around 13000 and 55000 among the top 10 categories. Similar to the analysis with category_name, except by Nintendo, Apple and Nike, the other 7 categories are focused on the female market.

As seen above with other features, the extreme values, in this case, the highest and lowest median prices per brand name contain few instances. Table 11a and 11b illustrate the earlier statement. The highest and lowest median prices

contain only 1 item or less than 10, except with the brand ”David Yurman” that reported 242 items. The lowest median price is $3 from several brands without a specific category or target. In contrast, the most expensive median price is reported by ”Demdaco” with $429. In this top, we can easily group them in two. First, luxury brands namely Proenza Schouler, Oris, Longines, Frédérique Constant, and David Yurman. Second, electronic/home products from Demdaco, Auto Meter, Blendtec, and Vitamix. Only Dainese sells items that are not related to any of these categories.

2.1.6 item_description

Item_description provides a brief explanation of each merchandise which might contain valuable data to predict the price. First, we determine the number of products with and without description. Second, the minimum, maximum, and mean from the description length. Last, the relationship between the description length and the price.

During the preprocessing step there is a deeper analysis of this feature. In this first stage, I did not try to change the dataset except the price with the log function. Later, I use the nltk library to extract as much information from the name and description.

From the entire products, there are 82431 that do not have any description. The largest length is 2144 characters, including spaces, although up to the 98% of the instances have a maximum length of 803. The mean is 154.05 with a standard deviation of 178.3.

As mentioned, in this part, I only took the number of characters in order to determine whether there is a correlation with the price or not. Apparently, the heat map below indicates that there is not, the correlation between the description length and price is 0.043.

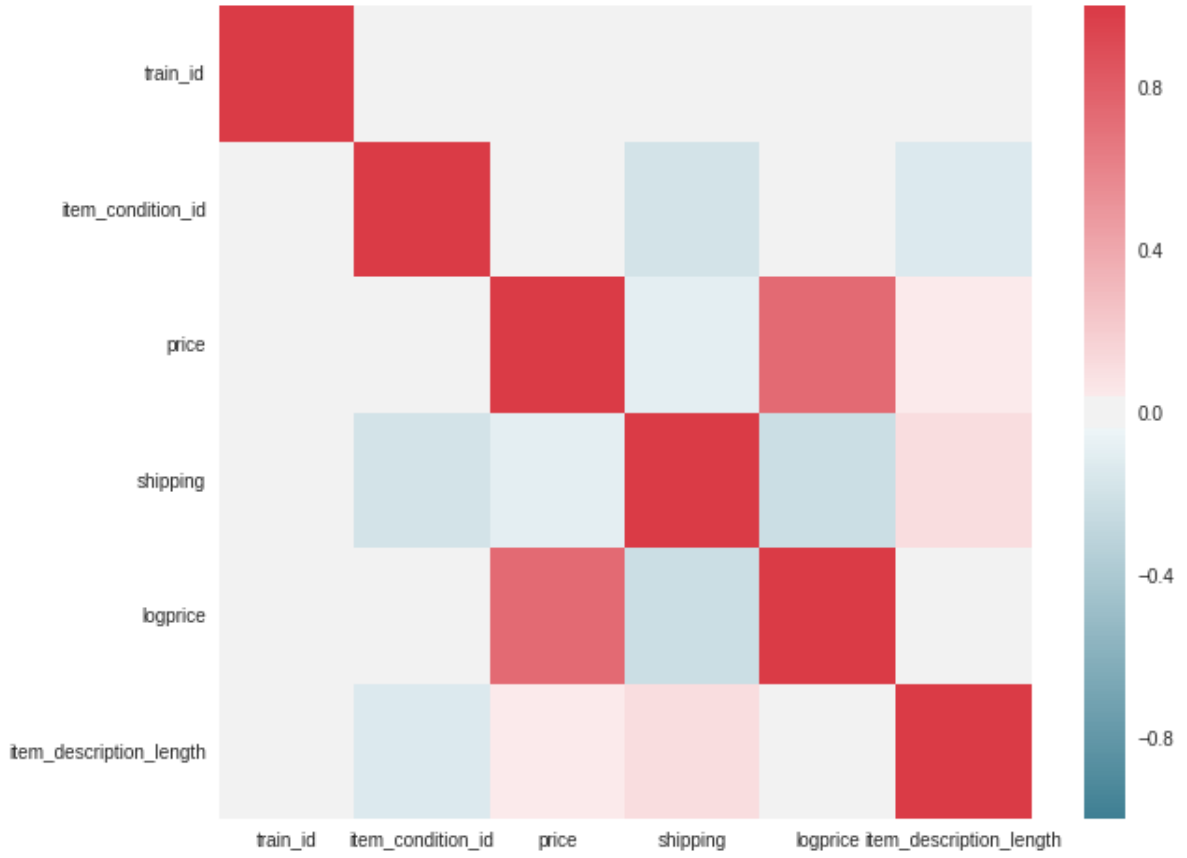


Figure 12: Heat map from the continuous data contained in the training dataset.

In summary, the analysis can be divided by the type of features. On one hand, from the heat map above, none of

the discrete features have a clear or strong correlation with the price or log price. Nevertheless, in my opinion, both item_condition_id and shipping might support the string features after the preprocessing. On the other hand, text variables provide a better understanding of the dataset by themselves and in combination with the price. For instance, the target market, number of items by category or brand, prices by brand, and prices by category, to name a few.

2.2 Algorithms proposed

During the capstone proposal six algorithms were proposed based on the literature review:

- **Decision Tree Regression**. This technique has been reported to work well to conduct regressions[14]. It contains 3 advantages that were key to use it in this project. First, it works well for discrete and continuous values (presented in this project). Second, it also can be used for feature selection, or review the feature relevance of the model. Last, after training, the prediction time is reduced due to the number of instance.

  As the name suggest, this method constructs a model implementing a tree structure. An example can be seen in the image below. The algorithm implemented is called ID3 by J.R. Quinlan[22]. In summary consist in 4 steps:

  1. Calculate entropy from each feature to decide the best split.
  2. Split the instances into subsets using the feature with minimum entropy or maximum gain.
  3. Develop the tree with the selected feature.
  4. Repeat steps 1-3 (recurson),

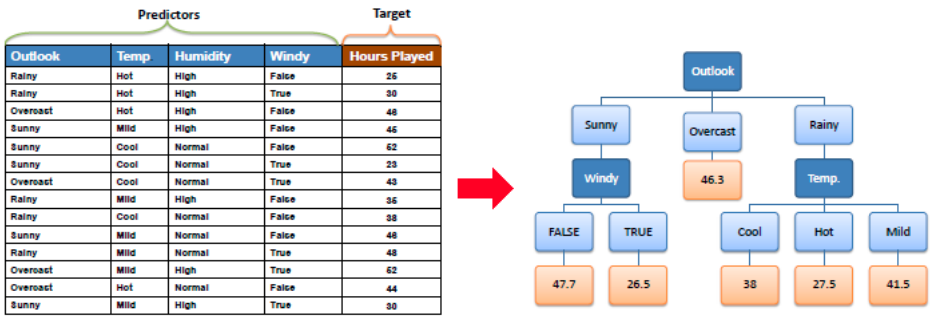


Figure 13: Decision tree sample. It predicts the number of hours played based on 4 features: Outlook, Temp, Humidity, Windy. Source [23].

- **Linear Regression**. As the previous technique, it was selected because has reported performing well when predicting prices for house[15] and retail[6] prices. In general, consists in minimising the sum of squared residuals between a dependent variable and an independent one. The general formula to calculate the residual and the sum of it is presented below.

  $r_i = y_i - f(x_i, \beta)$

  Where x_i is the independent variable, y_i the dependent variable calculated from the observation. The model function is $f(x_i, \beta)$ that has the adjustable parameters held in the vector β.

  The best values are found minimising the sum S of the residuals:

  $S = \sum_{i=1}^{n} r_i^2$

The minimum sum of the squares is found by setting the gradient to zero using:

$$-2\sum_i r_i \frac{\partial f(x_i,\beta)}{\partial \beta_j} = 0$$

Where j are the total number of parameters in the vector β which result in m number of gradient equations $j = 1,...,m$

- **Support Vector Regression(SVR)**. Since the dataset has features as the name and the description, part of the research looked on finding models that could work well with text analysis to predict a continuous variable. This is the case for Support Vector Machines [24].

  There are three main reasons for choosing this algorithm. First, it avoids over-fitting, a common problem with the Decision trees and KNN. Second, by using the kernel trick, different approaches can be implemented to provide different results. Third, because establishes a decision boundary with different types of kernels, it can groups or predicts values, not linearly separable, a big problem with the linear regression.

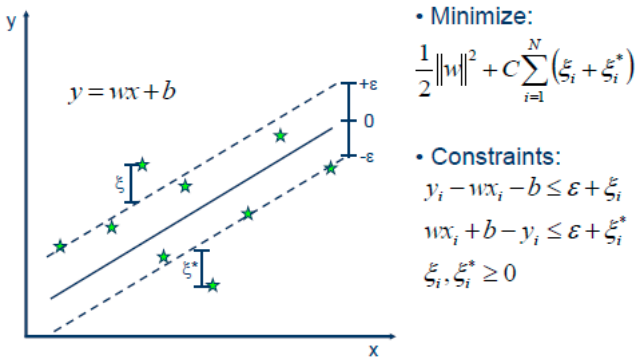


Figure 14: Illustration that contains the main parts of a SVM. Source [25].

Overall, the SVR works similar to an SVM. That is, works by finding the best hyperplane with the lowest error and maximising the margin. Figure 14, presents the equations for the hyperplane $y = wx + b$, minimise the error (under the Minimize dot), and finding the margin (under the Constraints part) where:

  - y - is our goal function that has the most ε deviation from the actual obtained targets.
  - w - is the normal vector to the hyperplane.
  - x - is the p-dimensional vector containing the entire information from each instance.
  - b - determines the offset of the hyperplane from the origin along the normal vector w.

- **XGBoost**. From the list, this is the most recent algorithm. Proposed by T. Chen and C. Guestrin[26] in 2016, is a tree boosting system that supports high scale data. Even when the algorithm was proposed in 2016 and there are not many formal papers. It was selected because it has gained wide acceptance by the data science community, and has probed of being an excellent method to conduct regression, winning many competitions in kaggle[27].

  In a nutshell, it is composed by three parts: 1) Regularized Learning Objective, 2) Gradient Tree Boosting, and 3) Shrinkage and Column Subsampling.

  First, a set of classification and regression trees (CARTS) are created. Each instance is classified in a particular leaf and a score is assigned to it. For instance, Figure 15, shows how a decision tree is created from different family members to decide whether they like video games or not.

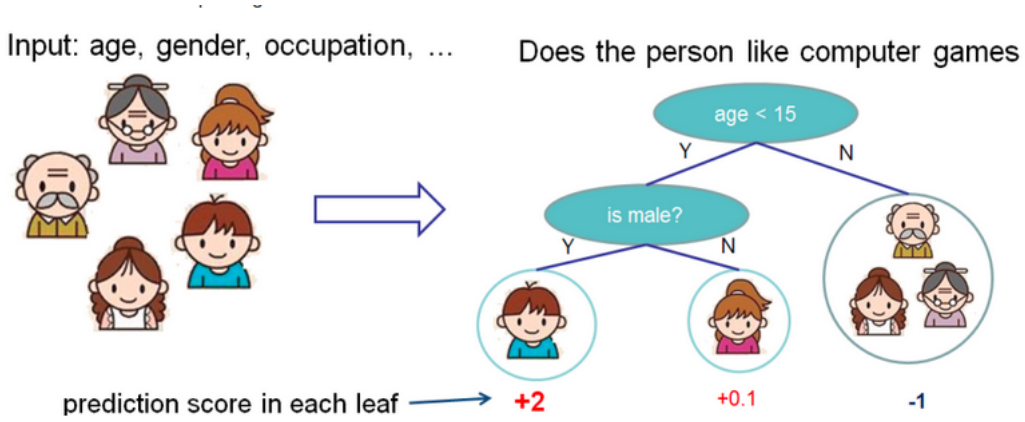


Figure 15: Instance classification and evaluation per leave. Source [27].

However, because this is an ensemble method, different trees are created and the sum of their scores are added.

Second, a structure calculation is done. That is, it measures how well a structure tree is. The image below shows how this is done. Overall, each leaf on each tree has its own g_i and h_i which are summed to calculate the tree score. This score can be seen as the impurity measure per tree. Although, it also takes the model complexity into account.

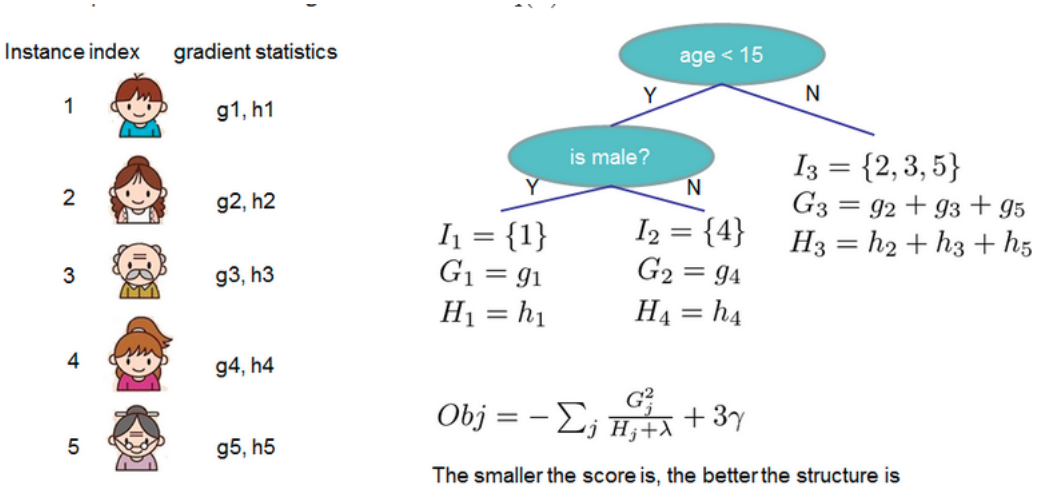


Figure 16: Instance classification and evaluation per leave. Source [27].

Last, in order to prevent overfitting, two methods are implemented. First, the shrinkage technique proposed by Friedman[28]. And Second, column subsampling, commonly used in Random Forest[29]

3 Modeling

3.1 Preprocessing

This section focuses on the transformation of values such as name, category_name, brand_name, and item_description. Although item_condition_id and shipping are discrete values, at this moment I do not that they need any treatment. Previously, we had modified the price with the Log function and remove prices with a value equal to $0.0 due to the fact that in reality, none item could value $0.0.

As a first step, we can remove the train_id, category_name, item_description_length, price. Train_id does not provide any value for the model, is just a consecutive key in the dataset. Category_name was split in the previous section. This

column is extra since we have three new called ”category1”, ”category2”, and ”category3.”

Item_description_length was created to calculate the correlation between the price and the length, after the results and confirm that does not exist, it is not needed anymore. Since we have the log price, the column with the real price is not required anymore.Then, we will follow some best practices to clean and process strings[30]

Details on the implementation can be seen in the ”.ipynb” file annexed to the report under the Pre-Processing section. In this document, I only discuss the reasons for such approach, the results and the challenges.

The entire preprocessing was divided in 7 steps. These were based on samples and best practices found during the research.

1. **Change to lower case**. As the first step, it is recommended that all the characters should be lower case[31]. As mentioned, most of the times is a good idea to start with this strategy. It allows treating words equally without caring whether they are at the beginning of a sentence or the author made a mistake mixing the letters. Since the writing type in the Mercari website is informal, these kinds of errors can be easily done. An example of the implementation is shown below:

```
train['name'] = train['name'].str.lower()
```

2. **Remove stop words**. Usually, common terms do not add any value to the model. Instead, increase the instance dimension, which could lead to noise and performance issues. For example, prepositions, conjunctions, and pronouns. The nltk[32] has a method called ”stopwords” that contains a list of common words to remove. Below is a sample from the implementation.

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
train['name_without_stop'] = \
train['name'].apply(lambda x: ' '.join([word for word in x.split() \
if word not in (stop_words)]))
```

   from nltk.corpus import stopwords

3. **Remove punctuation**. Similar to the previous point. Most of the times, punctuation marks have little or no value. Thus, as in step two, they were removed. The method to remove them was by regular expressions, part of the is shown.

```
train["name_without_stop"] = train['name_without_stop'].str.replace('[^\w\s]','')
```

4. **Lemmatization the name and item description**. Depending on the context and syntax, the same word can be written differently. For instance, the verb ”run” can be written as runs, ran, or running by the user to describe tennis shoes. The lemmatization is a technique in natural language processing that transform the words in their basic form, since, in the end, we care about the word and not its grammar. In the case presented, the verbs running, runs, and ran would be changed to just ”run.”

   Nltk provides a method called ”WordNetLemmatizer” to modify each word to its basic form. However, if the type(e.g. noun, verb, adjective) is not passed, the function assumes that is a noun. To determine it, there is another method called ”pos_tag” which returns the class.

The implementation consists of two functions. The first one called ”LemaWord” receives the word to lemmatize it. The second function ”get_wordnet_pos” runs inside ”LemaWord”, it is the one that maps the type of word so ”WordNetLemmatizer” can transform it. Here is a snippet that shows how the first function is called.

```
train['name_without_stop'] = \
train['name_without_stop'].apply(lambda x: ' '.join([ LemaWord(word) \
for word in x.split()]))
```

5. **Word analysis**. In the beginning, this step was not considered. Yet, after the words were vectored, the number of features increased drastically. Consequently, to decrease the number dimensions, a deeper analysis to remove not valuable words was conducted.

   This process consisted in 3 steps:

   (a) *Split words*. From each instance, the entire sentence or sentences were cut to create a list of words. Numpy has the method ”split” which facilities the task. This process was done with the name and the description. Also, a new feature was created with the name and description called ”full_description” to understand the entire dictionary from both variables. An example is presented.

   ```
   train['name_without_stop'] = train['name_without_stop'].str.split()
   ```

   (b) *Words analysis*. The first part the number of words in the three features(i.e. name, description, full_description). The second part used the library called ”FreqDist” from the nltk to check the word’s frequency.

```
count    1.481661e+06
mean     4.125052e+00
std      1.523331e+00
min      0.000000e+00
50%      4.000000e+00
75%      5.000000e+00
85%      6.000000e+00
95%      7.000000e+00
max      1.300000e+01
Name: count_name, dtype: float64
```

(a)

```
count    1.481661e+06
mean     1.773811e+01
std      2.030410e+01
min      0.000000e+00
50%      1.100000e+01
75%      2.100000e+01
85%      3.100000e+01
90%      4.100000e+01
95%      6.100000e+01
max      1.930000e+02
Name: count_description, dtype: float64
```

(b)

Figure 17: a). Word distribution in the name feature b) . Word distribution in the description feature

As shown above in the figure 17, the average number of words in the name and the description are 41.2 and 17.7 with a standard deviation of 1.52 and 20.3 respectively. Up to 95% of the instances have at most seven words in the name and 61 for the description. Nevertheless, the remaining 5% can have as much as 13 words with the name and the descriptions 193.

After filtering the instances with the information of the 95% percentile, in total, there are 87,614 unique instances that have either a name longer than seven words or a description that exceeds 61 words. Moreover, there are 2,016 instances where both conditions are met.

In general, after implementing ”FreqDist”, we found that there are 334,316 unique words. From these, they are used 32,476,943 times, if an equal distribution would be assumed, each word would be used 97.14 times.

```
[('new', 566352),
 ('size', 551746),
 ('free', 294255),
 ('brand', 290607),
 ('condition', 258852),
 ('use', 234685),
 ('shipping', 234534),
 ('bundle', 224183),
 ('rm', 214864),
 ('pink', 201385),
 ('black', 199311),
 ('worn', 186995),
 ('color', 175879),
 ('price', 174810),
 ('2', 172304),
 ('never', 158947),
 ('great', 155574),
 ('1', 139198),
 ('small', 137404),
 ('one', 136953)]
```

Figure 18: List of the 20 most used words with the number of times used in the description and the name.

However, using the function called ”hapaxes” which returns the numbers of words that are used only one time, it is discovered that more than half of the words are written once.

In particular, there are 205,158 terms with one occurrence around the 61%. Figure 18 illustrates the reason of having so many words used once. For instance, the top 2 which are ’new’ and ’size’ are in more than a third of the entire instances each, together they count more than 1,000,000. Then, the 8 following words ’free’, ’brand’, ’condition’, ’use’, ’shipping’, ’bundle’, ’rm’, ’pink’ are used more than 200,000 times each. As seen, most of the words are related to the condition, the shipping, and colours.

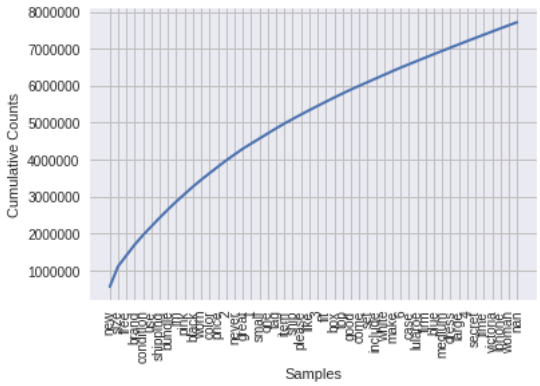


Figure 19: Cumulative chart of the 50 most used words.

Figure 19 gives a better understanding of the usage of the word. As is observed, the top 50 words appear almost 8,000,000 times, which is about the 25% of the total amount. This logarithmic tendency is steady over the entire universe of words.

(c) *Cut name and description*. Using particular python properties with lists, the decision was to cut the name to allow only 7 words, and description to a maximum of 61. Here is a snippet from the code with the implementation.

```
train['name_without_stop'] =\
train['name_without_stop'].apply(lambda x: x[:7] if len(x) > 7 else x)
```

6. **Vectorization**. The first approach was using the one-hot encoding from pandas. However, due to the size of the dimensions was impossible. Then, Brownie[33] proposes different techniques with sklearn. I decided to use

CountVectorizer for the features ”condition”, ”shipping”, ”brand”, ”category”, ”categories”. Where, to clarify, ”Category” is the first level, which contains only 10 options, ”categories” is the union from the second and third level. TfidfVectorizer was implemented for ”name” and ”description.”

CountVectorizer is almost similar to one-hot encoding. The only difference is that returns a sparse matrix. Two parameters were customised. First, ”token_pattern” so the function accepts one word or digit, otherwise throws an error. Second, ”preprocessor”, also used in the TfidfVectorizer, prepares the entire feature to be vectored, an error occurs if the feature is not pre-processed before the vectored method.

Besides ”preprocessor”, TfidfVectorizer has two more parameters ”ngram_range” and ”max_features”. This document does not intend to explain in details the parameters, for more information, please consult the technical documentation[34]. ”Ngram_range” restricts the terms that will be extracted, for the name the range was (1, 2) and for the description (2,3). ”Max_features” established the maximum number of features to generate. By testing, the optimal number selected was 49,000 for the name and 97,000 for the description.

Last, the initial idea was to vector each feature separately, but by performance, time, and memory was not possible. Then, searching more option, I found FeatureUnion which concatenates the transformed features all at once, making better use of the memory and improves the performance, reducing the processing time, more details can be found in the documentation.

7. **Sparse matrix**. This is an extra step and can be removed. The reason to include it was that even when FeatureUnion reduced the vectorization time, the process used to take between 10 and 20 minutes. Storing the sparse matrix helped to reduce the time and memory use every time the kernel had to be run.

3.2 Benchmark

As mentioned in the capstone proposal, the algorithm selected for the benchmark was the K-nearest neighbours for regression.

Initially, the implementation did not present any difficulty with sklearn[35]. However, due to the number of features(i.e. around 150,000), the prediction part added complexity during the prediction phase. The data was split in 80% for training and 20% for testing with the ”train_test_split” method. The training part of the ”KNeighborsRegressor” method was configured with the default parameters, except the number of neighbours, changed from five to 10. Moreover, the training time was drastically low, at most one second, which was expected due to the type of algorithm.

The prediction with the default function only allowed 400 instances before breaking the cell with a memory error. The process used to take around 5 minutes for these instances. Thus, to predict more instances, parallel programming was coded using the library called joblib[36] from python.

It consisted in two functions, ”_predict” and ”parallel_predict”. The first is the actual place where the log price is estimated. The second configures the number of jobs, batches per jobs, batch size, and calls the first function with the method ”parallel”.

Contrary to a single thread, the parallel prediction reported improvements in performance an items supported. After several attempts, the maximum number of instances that were able to predict without errors was 40,000 in around 30 minutes. Comparing with the default implementation, running the single thread six times which would consume the 30 minutes, it would only predict 2,400 samples. This implementation increased the prediction time up to 1,666%.

The result from the KNN regression is a RMSLE of 0.5596. Comparing with the best result found in Kaggle which

is 0.3775, the difference is 0.1821. Considering that the log prices are between 1.38 and 7.6, I think the error is within an acceptable tolerance.

The laptop model and specifications where all the models were developed are:

OS:	Ubuntu 16.04 64-bit
Processor:	Intel® Core™ i7-2670QM CPU @ 2.20GHz * 8
Graphic Card:	GeForce GT 525M/PCIe/SSE2
RAM	8 GB

Table 2: Laptop Specifications where the kernel ran.

3.3 Implementation

The first approach to run each algorithm proposed in section 2.2 was not successful. The size of the sparse matrix turned either the training or prediction in a never-ending process. Therefore, a feature selection was conducted to be able to run all the algorithms at an appropriate time.

The features were selected using the f_regression function from Sklearn. Overall, it calculates the correlation between each regressor with the target value. Then, the value is converted first to an F-score and later to a p-value. In my opinion, it was the best algorithm from the entire options that Sklearn has. ”chi2” and ”mutual_info_classif” are for classification tasks. ”mutual_info_regression” relies on k-nearest neighbours and estimates the dependency between the variables, not the target value.

As mentioned above, initially, an RNN was planned. However, it was found that Keras, as such can not work with sparse matrices. I tried to implement some workarounds that exist in different documents without success. Every time the kernel broke with a memory error message. Apparently, in order to use an RNN with NLP, a completely different preprocess must be done from start to finish and specifically for this model.

Model	RMSLE
Decision tree regression	0.7513
Linear Regression	1.3545
Support Vector Regression	0.7212
XGBoost	0.6351

Table 3: RMSLE results from the models selected.

The table above presents the results from each model. A discussion on the results is conducted in the next section. Every model was configured with their default parameters to select the algorithm based on the best results. As seen, the lowest RMSLE was reported by the XGBoost, which made it the selected model for refinement.

3.4 Refinement

After tuning the XGBoost, there is a clear improvement. It started with 0.6351 and reduced the error 0.1681 to have a final RMSLE of 0.4670, the best score obtained.

In the grid search presented in the code, there are presented only the final parameters. The process to obtain them was one by one, starting with a range of them. Then, they were reduced to get the best possible. The tuning was conducted with a sample of 15,000 instances due to the time to try the entire training set.

The XGBoost package supports 21 parameters[27], two of them already deprecated (i.e. nthreads, seed). In the final list presented in code, there are seven variables that were changed from their default value because presented a positive impact in the model: max_depth, learning_rate, n_estimators, booster, min_child_weight, reg_alpha, reg_lambda. The rest did not provide any improvement(i.e. gamma, max_delta_step, subsample, colsample_bytree, colsample_bylevel, scale_pos_weight) or were not tried since are more focused on the configuration as silent, objective, random_state, missing, n_jobs, and base.

The method to tune each parameter was by using GridSearchCV running each parameter one by one with values greater and less than the default. Then, the margin was reduced to find the best score. For instance, max_depth as a default variable of 3, the initial set was [2, 3, 5, 10, 20, 50, 100, 200]. From that list, the best score was 20. Hence, the next group was [15, 18, 20, 22, 24, 26, 30] having 18 as the best. The last set was [16, 17, 18, 19] confirming that 18 provided the lowest error rate. The entire 13 parameters were tested as the sample above.

The final tuning is the following:

- max_depth = 18
- learning_rate = 0.19
- n_estimators = 92
- booster = gblinear
- min_child_weight = 1
- reg_alpha = 0.25
- reg_lambda = 1.1

Compared with the results in the competitions, this model performs well by looking the best result which is 0.3777. If we put this result on the leader board it would be right in the middle.

4 Results

During the previous section, four models were created with the default options to decide the best one. As seen, the best model comes from the fourth method ”XGBoost” with an RMSLE of 0.6351, followed by the SVR reporting 0.7212. XGBoost proved to be the best model not just by the metric, but also because it was able to handle the entire sample set in the least amount of time. Around 10 minutes for the training phase and about a second to predict 296,333 instances.

To clarify, XGBoost was first tested with only 20,000 instances as the other models, providing similar results as the one presented with the entire instances.

In contrast, SVR took less time to train. However, it could not manage the entire training set due to a Memory error and the prediction time takes around 20 minutes.

It might be possible that at this stage we could define the KNN as the best one since it shows less RMSLE with 0.5596. But same as the SVR, the prediction time takes a lot of time. For instance, only 40,000 test instances took around 30 minutes, turning the model in not-optimal if we consider the time to train and predict. Points where the XGBoost outperforms its competitors.

In addition, to test the reliability and robustness of the XGBoost model, the cross-validation was conducted. It was from the test set which contains 296,333 instances. It consisted of a k-folds cross-validation with a random state of 42, and shuffle enabled. From the 20 folds tested, the RMSLE went from 0.4803 to 0.4984 a quite small difference of 0.0181. The reported mean was 0.4904 with a standard deviation of 0.0043, only 0.0234 more than the value reported during the refinement (i.e. 0.4670). As a result, it can be concluded that the model implementing the XGBoost is consistent, reliable and robust to be used for more data.

5 Conclusion

This project brought many challenges, from the analysis to the final tuning. In my opinion, contained more challenges that any project in the course. Which resulted in gaining more experiences in each area. Namely, the analysis, the preprocessing, Sklearn libraries, programming, and a new library called XGBoost. Here I resume the most relevant ones:

Exploration

1) It is very useful to use the logarithmic trick in order to reduce the dispersion in the data. In this case, for the price.

2) Surprisingly the items that do not include the shipping in the price are more expensive.

3) The granularity for the categories increases drastically from the first to the third option.

4) Most of the products are for women.

Preprocessing

5) Natural Language Processing (NLP) is expensive in terms of hardware.

6) There are several techniques to homogenize the text data. For such purposes, the library called Nltk was used. Lemmatization, removing stop words, removing punctuation, and turn all the words to lowercase were used in this project.

7) Implementation of the library called FreqDist to understand on details the corpus, words, and dictionary from the text features. This, to take better decisions when selecting, removing and joining features.

8) Undestanding and implementation of the functions called FeatureUnion and Pipeline from Sklearn to take advantage of the hardware usage and reduce time.

9) Sparse Matrices were created after conducting the transformation from the text features. The data type that was not used during the course.

Models

10) Parallel programming using the joblib library to work with high dimensional instances and improve performance.

11) I tried to implement RNN as proposed in the capstone. However, after trying with sparse matrices it used to take much time without success. There is not an official implementation with sparse matrices and a complete different preprocessing should've been conducted in order to fully implement the RNN.

12) XGBoost is an excellent option. The library is easy to impĺement, configure and offer better results than the other algorithms.

Additionally, the chart below presents the error rate between the predicted value and the real one. As seen, the blue dots together form a figure similar to a rhomboid tilted up, divided by half from the tendency line. This line goes from the bottom left to the upper right. This plot clearly defines where the predictions have more discrepancy, happening in the cheapest and the most expensive products in an opposite way. In one hand, the inexpensive products often are predicted with a higher value. On the other hand, the pricey items are estimated at a lower price.

As the literature mentions, it is true that 80% of the time in a machine learning project is for preprocessing. For me,

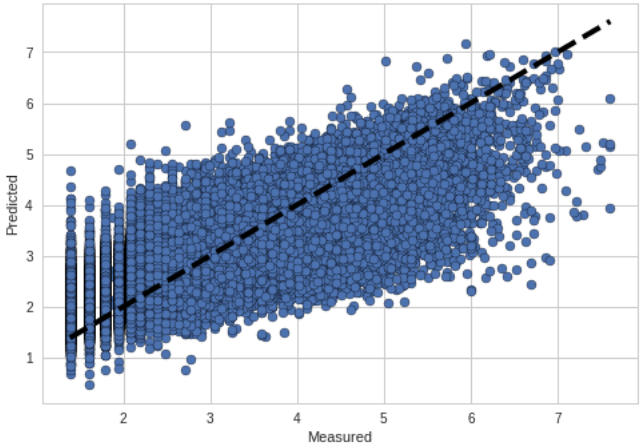


Figure 20: Scatter plot, illustrates the error rate in the prediction.

since I had no previous experience with NLP took more in this area. There is not an unique path to preprocess text, and depending on the methods implemented the model could increase or decrease its error.

It might be possible that in this project we could have had two models, one for the women and the rest for the other categories. The number of items for women is by far more than the other categories together as mentioned in the analysis.

In conclusion, on one hand, this project allowed me to reinforce the knowledge gained during the course. For instance, Sklearn functions, analysis techniques, algorithms as SVR, KNN, Linear Regression. On the other hand, it lets me explore new techniques, algorithms, tools, and libraries as mentioned in the list above. I think that helped me to understand how real machine learning projects should be conducted by applying what you know, but also, to learn and adapt what is being implemented by the community.

5.1 Future Work

In future work, we could conduct an entire different preprocessing step focused to develop the RNN and avoid sparse matrices. Or develop a solution for the community since there is not a current implementation. Also, for every model develop a parallel solution either for the training or predicting phase. It might be possible to refactor the parallel solution develop in the benchmark to make it work with other algorithms. Also, a deeper feature selection in the natural language part, in this study only the default f-regressor was used. Moreover, test different models to work without the top 10, 20, or 50 most common words and compare results.

References

[1] Reid Pryzant, Young-joo Chung, and Dan Jurafsky. “Predicting Sales from the Language of Product Descriptions”. In: (2017).

[2] Kris Johnson Ferreira, Bin Hong Alex Lee, and David Simchi-Levi. “Analytics for an online retailer: Demand forecasting and price optimization”. In: *Manufacturing & Service Operations Management* 18.1 (2015), pp. 69–88.

[3] Dipanjan Sarkar, Raghav Bali, and Tushar Sharma. “Customer Segmentation and Effective Cross Selling”. In: *Practical Machine Learning with Python*. Springer, 2018, pp. 373–405.

[4] KM Anil Kumar et al. “Effective Approaches for Classification and Rating of Users Reviews”. In: *Proceedings of International Conference on Cognition and Recognition*. Springer. 2018, pp. 1–9.

[5] Hassan Waqar Ahmad. “Prediction of retail prices using local competitors”. PhD thesis. Faculty of Graduate Studies and Research, University of Regina, 2014.

[6] Prajakta Badhe. “Retail pricing prediction using linear regression”. In: *Neural Networks & Machine Learning* 1.1 (2017), pp. 1–1.

[7] Michael P Wellman, Eric Sodomka, and Amy Greenwald. “Self-confirming price prediction strategies for simultaneous one-shot auctions”. In: *arXiv preprint arXiv:1210.4915* (2012).

[8] Jerry Felsen. “Learning pattern recognition techniques applied to stock market forecasting”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 5.6 (1975), pp. 583–594.

[9] Manak C Gupta. “Money Supply and Stock Prices: A Probabilistic Approach”. In: *Journal of Financial and Quantitative Analysis* 9.1 (1974), pp. 57–68.

[10] Salim Lahmiri. “A Technical Analysis Information Fusion Approach for Stock Price Analysis and Modeling”. In: *Fluctuation and Noise Letters* (2018), p. 1850007.

[11] Xiao Ding et al. “Deep Learning for Event-Driven Stock Prediction.” In: *Ijcai*. 2015, pp. 2327–2333.

[12] Slava Kisilevich, Daniel Keim, and Lior Rokach. “A GIS-based decision support system for hotel room rate estimation and temporal price prediction: The hotel brokers’ context”. In: *Decision Support Systems* 54.2 (2013), pp. 1119–1133.

[13] Stacey Mumbower, Laurie A Garrow, and Matthew J Higgins. “Estimating flight-level price elasticities using online airline data: A first step toward integrating pricing, demand, and revenue optimization”. In: *Transportation Research Part A: Policy and Practice* 66 (2014), pp. 196–212.

[14] EJTG van der Burgt. “Data Engineering for house price prediction”. In: (2017).

[15] Adyan Nur Alfiyatin et al. “Modeling House Price Prediction using Regression Analysis and Particle Swarm Optimization”. In: ().

[16] Changshou Luo et al. “Prediction of vegetable price based on Neural Network and Genetic Algorithm”. In: *International Conference on Computer and Computing Technologies in Agriculture*. Springer. 2010, pp. 672–681.

[17] Austan D Goolsbee and Peter J Klenow. “Internet Rising, Prices Falling: Measuring Inflation in a World of E-Commerce”. In: ().

[18] VL Raju Chinthalapati, Narahari Yadati, and Ravikumar Karumanchi. “Learning dynamic prices in multiseller electronic retail markets with price sensitive customers, stochastic demands, and inventory replenishments”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 36.1 (2006), pp. 92–106.

[19] *Mercari Price Suggestion Challenge Description*. https://www.kaggle.com/c/mercari-price-suggestion-challenge. Accessed: 2018-01-21.

[20] Dhruv Bhatia Kirubakumaresh Rajendran. *What is the difference between an RMSE and RMSLE (logarithmic error), and does a high RMSE imply low RMSLE? Description*. https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE. Accessed: 2018-4-10.

[21] *MAE and RMSE Which Metric is Better? Description*. https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d. Accessed: 2018-4-10.

[22] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[23] *Decision Tree - Regression Description*. http://chem-eng.utoronto.ca/~datamining/dmc/decision_tree_reg.htmd. Accessed: 2018-4-10.

[24] Robert P Schumaker and Hsinchun Chen. “Textual analysis of stock market prediction using breaking financial news: The AZFin text system”. In: *ACM Transactions on Information Systems (TOIS)* 27.2 (2009), p. 12.

[25] *Support Vector Regression Description*. http://www.saedsayad.com/support_vector_machine_reg.htm. Accessed: 2018-4-10.

[26] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[27] *XGBoost python API reference Description*. http://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn. Accessed: 2018-4-10.

[28] Jerome H Friedman. “Stochastic gradient boosting”. In: *Computational Statistics & Data Analysis* 38.4 (2002), pp. 367–378.

[29] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[30] Matthew Mayo. *Natural Language processing Key Terms Explained Description*. https://www.kdnuggets.com/2017/02/natural-language-processing-key-terms-explained.html. Accessed: 2018-01-21.

[31] Hinrich Schütze Christopher D. Manning Prabhakar Raghavan. “Capitalization/case-folding. Description”. In: (2008). Accessed: 2018-04-10.

[32] *Natural Language Toolkit Description*. https://www.nltk.org/. Accessed: 2018-4-10.

[33] Jason Brownlee. *How to Prepare Text Data for Machine Learning with scikit-learn Description*. https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/. Accessed: 2018-04-10. 2017.

[34] *TfidfVectorizer Description*. http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Accessed: 2018-4-10.

[35] *sklearn neighbors KNeighborsRegressor Description*. http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html. Accessed: 2018-4-10.

[36] *Joblib python pipelines Description*. https://pythonhosted.org/joblib/. Accessed: 2018-4-10.