Applied Machine Learning: Final Project
Mercari Price Prediction
Team 27

{Arnav Saxena (as6456), Nuanyu Shou (ns3492), Hongou Liu (hl3518), Xinfu Su (xs2444), Zihao Liu (zl2986)}@columbia.edu

Introduction

Mercari is a Japanese e-commerce company. Their main product is the community-powered marketplace they offer via the Mercari marketplace app. It's just like eBay where people can buy and sell used products ranging from clothing to electronics and more.

The main objective of this proposed project is to use data of items sold in the past on mercari marketplace and build a predictive model that could suggest the optimal product prices to sellers.

Dataset used

Mercari hosted a kaggle contest around the same topic in 2017. We used the data they provided for the same (link). The data provides the following information for about 1.5 million SKUs:

1. **train_id, test_id** — the id of the product
2. **name** — the title of the product
3. **item_condition_id** — the condition of the product provided by the sellers
4. **category_name** — category of the product
5. **brand_name** — the product's brand name
6. **shipping** — 1 if shipping fee is paid by seller and 0 if shipping fee is paid by buyer
7. **item_description** — the full description of the product
8. **price** — the price that the product was sold for

Data Preprocessing

1) Data Split & Sampling
We had originally split our data into 7:2:1 (train:val:test). And though we used all the ~150,000 observations in the test set for model evaluation, we decided to use only 200,000 observations of the original train + validation set due to limited computing resources as well as our ambitions of incorporating advanced and heavier text representations. This helped us cut a lot of training time as well.

2) Categorical features
Based on our observations and discussions in the previous deliverable on *'Data Analysis and Visualization'* wherein we explored the relationship between categorical features (item_condition / brand_name / category / shipping) and price, we performed target encoding on all of them in the hopes of capturing the bias of each of their categories on target price.

3) Textual Features

- We had three textual columns in our dataset - *'name'*, *'brand_name'* and *'item_description'*
    - There were about ~42% of brand names missing in the data, half of which we were able to extract using the item name (discussed in the previous deliverable)

    - Later, we concatenated all the three features to have a *'super text'* feature.
- Furthermore, we had this assumption that this *'super-text'* feature is one of the major indicators for price. Hence, we experimented with three different representations for this feature as follows:
    - **Term frequency-Inverse document frequency (TF-IDF):** We employed the sklearn TfidfVectorizer package to transform our text feature to a matrix with 400 most important dimensions.
    - **Word2Vec:** We trained word2vec model on our training set and created a word2vec representation with 150 dimensions for the 'super text' column
    - **Birdirection Encoder Representation from Transformers (BERT)**: We utilize a pre-trained BERT encoder to extract semantic information from the text, generating a 768-dimensional embedding vector.

Experimentation

For our experiments, we have considered using **mean squared logarithmic error** as our choice of evaluation metric instead of mean squared error since the former is not influenced by the variation and scale of the data. Also, we have evaluated our models using the normal price as well as the log transformed price so as to understand if the skewness of this variable can have any impact on the models. Furthermore, to obtain a reasonable baseline result, we considered the mean price of our training set as the predicted value and evaluated it against our test set. This gave us a baseline score of 0.68 MSLE on normal price and 0.57 MSLE on log transformed price.

Next, we divided our experimentations into two phases. In the first phase, we worked with traditional machine learning algorithms such as linear regression, random forest, and XGboost while in the second phase we experimented with various deep learning regression architectures. The table below shows the specifications of the tuned deep learning regression models. Lastly, as discussed in the previous section, we decided to experiment with different representations of this feature. As discussed in the previous section these representations were tf-idf vectors, pre-trained word2vec and pre-trained BERT embeddings.

Model	**Number of Hidden Layers and size**
DL + tfidf (dim 400)	7: 512>256>128>64>32>16>8
DL + word2vec (dim 150)	6: 256>128>64>32>16>8
DL + BERT (dim 786)	8: 1024>512>256>128>64>32>16>8
nb_epochs for each model = 10, batch_size for each model = 32	
Optimizer used: Adam with learning rate 1e-4	

Results & Conclusions

As can be seen in the table below, XGBoost model trained on the dataset with TFIDF embeddings outperformed all other models with a significant margin. Even the best deep learning model still lags XGBoost by about 3 percentage points. Further, we noticed that the highly advanced BERT embeddings produced almost comparable results as that of the TFIDF embeddings across all models. These two observations demonstrate that simply using a complex model does not guarantee the best results. In addition, on comparing the model performance trained with original and log scaled target price, it is

obvious that the errors are reduced significantly for most combinations of traditional and deep methods when log transformed price is used as target. In terms of efficiency, though the random forest and XGBoost both improve the prediction accuracy compared with the linear regression model, the time spent for training the random forest is much greater than the one of XGBoost. Different embeddings do not affect the time cost so much. Thus, considering the accuracy of the prediction and efficiency of the training, XGBoost with TFIDF and the log scaled price is the best model for this price prediction case.

Future Work

Though our deep model with bert is slightly behind XGboost, there is still room for improvement. We still see the effectiveness of extracting semantics of textual description and brand name in promoting performance. Since we only used the pretrained BERT for encoding, we can definitely apply more text pre-processing of the raw text and fine-tuning to get more nuanced text encoding. Besides, we will also try an ensemble of our best traditional model and deep model to try push the performance even further.

S. No.	**Model + "Super-text" feature representation**	**MSLE (price)**	**MSLE (log(price))**
1.	Baseline (mean of training set)	0.68	0.57
2.	Linear Regression with TFIDF	0.51	0.36
3.	Linear Regression with pre-trained Word2Vec embeddings	0.55	0.42
4.	Linear Regression with pre-trained BERT embeddings	0.56	0.34
5.	Random Forest with TFIDF	0.42	0.36
6.	Random Forest with pre-trained Word2Vec embeddings	0.42	0.36
7.	Random Forest with pre-trained BERT embeddings	0.42	0.36
8.	XGBoost with TFIDF	0.35	**0.29**
9.	XGBoost with pre-trained Word2Vec embeddings	0.45	0.37
10.	XGBoost with pre-trained BERT embeddings	0.37	0.31
11.	Deep Learning Regression with TFIDF	0.40	0.45
12.	Deep Learning Regression with Word2Vec	0.50	0.55
13.	Deep Learning Regression with pre-trained BERT embeddings	0.37	**0.32**