Mercari Price Suggestion Challenge

Yuval Helman (ID. 315581819), Jakov Zingerman (ID. 316879899)

Submitted as final project report for Machine Learning Applications, Tel Aviv University, 2019

1 Introduction

The problem we chose to work on is the "Mercari Price Suggestion" that was held as a challenge on kaggle.com. The problem consists of using data of used products listings in order to suggest product prices for new product listings. This problem is both theoretically interesting since it allows to easily apply the advantages of complicated ML technologies for large amounts of data and both practically important as a real life problem as this problem is valuable by large companies that are trying to gain the most of the profit out of product prices by predicting their real value. This implies that offering a better service to the customers can draw more profit. The challenge given at Kaggle is an evidence for the practical value of the problem since its being funded by a large and important company. The problem itself consists of data which is not trivial for a computer to understand in its raw form (i.e text data with semantic meanings which needs to be translated to numbers for the computer to be able to work with it). The problem seemed as a good opponent for us to try and tackle using the material we learned in the ML applications class.
A Big challenge was translating sentences to numbers which will be able to represent the semantic meanings of the sentences (sentence embeddings). We saw in class word and sentence encoding methods (TF-IDF, w2v, 1-hots etc.), and we will try to imply them on our problem. The idea of transforming semantic texts into numeric data is magical in our eyes. We will try to use it properly and get good results out of it.

1.1 Related works

1. facebook's infersent sentences encoding: https://github.com/facebookresearch/InferSent
2. https://www.researchgate.net/publication/268279098_A_Neural_Network_based_Model_for_Real_Estate_Price_Estimation_Considering_Environmental_Quality_of_Property_Location
3. Kaggle's Mercari Price Suggestion Challenge 1st place solution - https://www.kaggle.com/c/mercari-price-suggestion-challenge/discussion/50256

2 Solution

2.1 General approach

We approached the problem with the set of tools and methods we learned throughout the course. We tried to examine them in front of the problem in order to find the ones that can achieve the best results when facing our problem. As a first phase, we went over every method learnt in order to try and see if it might fit to the regression problem we are trying to solve. The models we found out to be the most suitable for the problem were Linear regression, XGBoost, RandomForest and Neural networks, when all of them were taken into consideration for regression purposes. Then, we searched the web for existing similar solutions with those tools and algorithms and concluded out of the material we found a plan of how to examine those methods in order to be able to tell which one of them works best for us. Then, we proceeded into learning some new technologies such as working with Facebook's InferSent and going in depth into controlling pytorch, pandas, and generally dealing with regression problems. After the model selection phase, we found out that on small datasets and with a relatively limited training time, the best results comparing to the other mentioned models were given to us by a 3 layer simple neural network, which contained 20 neurons in each of its hidden layers. Therefore, we decided to keep our focus on working with the neural network model and to proceed with finding its best properties for our problem such as the number of layers, the number of neurons in each layer, the loss function and the optimizers that will give us the best results using the neural network.
Then, we tried to examine various neural network models, each with differnt amount of layers and with different numbers of neurons. Yet, we didn't get good results and therefore turned to another approach of trying to improve an existing solution. We went trough the architecture of the winning solution of the challenge (**?**), which is based on MLP and suggested some possible improvements to it to get some better results than those we got using simple fully connected neural networks with relu as an activation function.
A major sub problem we needed to solve, which was needed for every relevant approach we could choose, was to find a way to transform the product representation sentences to a numeric value while preserving the semantic connection of it to our data. The main feature (column) that affected the prediction is /"item-description" which gives a brief text description of the item. It consists of a large amount of text data which aims to represent the product and therefore we considered this feature to probably be the most informative one, as it can state if the item is old/new, small/big, broken/in-good-shape etc., which all can tell a lot about a possible pricing suggestion that an item can get. To demostrate it, the correlation between this feature and the pricing of the item can be shown in the plots section at the end of this file.
As was suggested by Kfir (which is probably the only one reading this - thanks :)), we used the InferSent sentence encoder created by Facebook.
InferSent has a model which has been trained on all the words in Wikipedia us-

ing web scraping. This model seemed better then anything else we could build by ourselves for our goal of transforming the text into numerical representation that preserves as much of the semantic value as possible. Altough the way InferSent obtains it's data is not really aimed at our goal of prediction, we think it can really capture the semantic context of our sentences and then be used accordingly (numeric-wise) to be learned by a proposed model for the prediction purposes.

2.2 Design

At first, we needed to prepare the data. We had to clean it from dirty and NaN values, to avoid type collisions and to convert every string value to a numeric value while keeping the semantic meaning of it inside its numeric value it got. We did this using the Facebook's Infersent code. Needed to mention, most of the solutions we saw introduced FastText as the sentence encoder yet we wanted try a different approach and go with our own different solution and something that we didn't see someone else try yet.
Browsing the internet looking for inspiration about the optimal solution, We found that most people combine text columns together before using a numeric encoder. We didn't really understand intuitively why this should make better results, because combining different sentences together one after the other will have a bad semantic (in a high probability). Although, Having Infersent output 4096 columns per encoding, together with dealing with a lot of data and weak processing power, we decided to do it too (Without it we didn't succeed in the encoding). Therefore, All of the columns that were encoded with Infersent were first concatenated.
Then we turned out to examining the methods we learnt in class and used mainly SKlearn, Pytorch and XGboost to test each of the models (Neural network,XGboost, RandomForest and Linear Regression). To get the most appropriate parameters for examining each of the models we assisted several resources such as (1) and (2).
Firstly, we tried working with tree models.
About our choices of tree parameters:
1. We suppose that the most of the outcome that can come from "Tree classifiers" in our problem is out of "logistic multi-class regression" solution in comparison to an exact prediction machine. We believe this conservative approach will be better on average.
2. Making the tree classifiers more shallow will let us have the sufficient resources (computation power and computation time) to take into consideration enough trees to get good results and mainly to avoid overfitting on the training data. We assume this approach leads to a better accuracy score on loss functions typical for regression (MSE, MAE etc.) in comparison to using a much smaller amount of deep trees. Thus, several parameters decisions we made were:
- 'min_samples_split' was given a higher value to prevent high tree depth, and stack similar price products in the same leaf, which may get their average on that leaf.

- 'subsample' was given a value less then one. This makes every boosting iteration to choose a certain % of the data to work on, and may prevent overfitting.

The rest of the parameters were decided by multiple recommendations seen on other people's solutions regarding regression problems, price prediction, or by common sense in order to have the algorithm run in reasonable time (max_depth, learning rate etc.)
In addition, we tested each of these models both with Mean Squared Error and with Mean Absolute Error loss functions to find the loss function that does the best on our data for every model.
After choosing the Neural Network as the model that gives us the best results, we iterated through a bunch of possible network architectures, tried both SGD and Adam optimizers and different learning rates to find out the combination that works the best for us.

At some point, we understood that even non-complex NN's outperform any of the trees we created. In the same time, we found out on the kaggle competition page that most of the competition discussions say Boosting is a poor solution for this problem.
Therefore, we tried training a neural network with different architectures and parameters yet with no significant success on the score side (relative to what we saw in the competition score board). Considering our next steps, we decided to try to test another approach to the winning solution of the challenge, feeding in Infersent's input to a MLP network.
The challenge itself made a restriction on the training time (60 minutes) and we wanted to examine a trade-off between this restriction and the aim to get the best possible score function while using PCA to reduce data dimensions and extracting the most important features.

3 Experimental results

All results were run on our laptop's CPU (8GB RAM), and we think that with more CPU power and time we could've had better results.
Experimental results are brought in this section on 400k rows of the data.
** At the end of the PDF we attached tables and plot regarding observations of the data etc.
Encoding data by 1-hot and Infersent brought the feature-space of the data into around 4200. In order for the models to train faster and the data-files not being too large, we used PCA into several alternatives. We saw that the best number for optimal run-time and preservation of the features correlation was at about 100 500. After that, we run all of our models on the data being reduced to a 100 column feature space. (mainly due to time constraints after struggling with InferSent to work properly.

Checking the regression problem with a basic linear regression predictor is always a good indicator to see how complicated the model should get further.

Running a linear regression model on the numeric data returned a predicted result which wasn't that good and probably wouldn't be given to a customer's item as a predicted price.

Running XGBoost with multiple choices of hyper-parameters resulted in kind of the same results, which were pretty poor on average and didn't outperform the standard linear regression model.
We also found out there's a possibility for creating a boosting model with multiple trees in each iteration which conducts resampling for each one (Random forest like we implemented in one of the HW) of the trees in every iteration. Even this model didn't outperformed the normal XGboost model.
As we stated above, we understood (saw some tips for that in the challenge website) that Tree predictors weren't too successful in the competition, and most top-rated competitors built a neural network to learn the data features.
Therefore, we tried looking for a successful structure of a NN and saw that it overcomes the other models and then tried other combinations of neural networks. Finally we got to our last approach of using the MLP with Infersent for the text sections. The results were as follows:
The test MSE score for 15 epoch (went over all the data) was 1663.409 and the test RMSLE (the measure with which the challenge was evaluated) score was 0.9798.

The numeric results we recieved weren't too precise in our minds. As we've seen (and plotted in the last section of this report) the feature to be predicted (the price) is ranging between 0$ and 300$, so the MSE/MAE results doesn't show of a really precise prediction rate on average. We can guess that the cause of the bad results is both the usage of Infersent and the PCA we did on the data which both negatively effected the results.

4 Discussion

To sum up, the whole process was our first experience dealing ourselves with a machine learning problem. Even though we went through the course We had to read and learn a lot to find out where should we start because facing a real world problem without structured guidelines is much more challenging (It is also pretty much fun). We learnt about how to preprocess data, about what is important when comparing models, about working with already proven to work models and that neural networks will probably out-preform every other model in precision matters eventually. We found out how sentence embeddings work and that the magic of representing a sentence by numeric values does actually work pretty well. Overall it was a really good and interesting experience which introduced for us the world of exploring and exploiting a machine learning problem in way the will allow us to do it by ourselves much more easily for a future problem we'll tackle.
We believe our solution could be improved a lot with more research of the ML world, and more time.

5 Code

https://github.com/YuvalHelman/mercariPriceSuggestion

Most of the python files are pretty self-explanatory from their names or from the comments on top of each file and function.
The files are split into logically uncorrelated code sections so each file can run by itself (basically pre_data.py is first, and then any of them could work by itself as each of them is related to a presentation of the data or to creating/running some independent model.)

Some information/files that aren't attached (mainly because they were too big to put on GitHub are:
1. cc.en.300.vec - the model trained on Wikipedia by infersent
(https://fasttext.cc/docs/en/english-vectors.html)
2. train/test data (taken from the challenge and was used for creating new CSV's of post-processed data which are also too big). test data had no labels due to competition restrictions. Therefore, we only used the train.tsv file, and split it into train/test. (It has like 1.5m rows)
(https://www.kaggle.com/c/mercari-price-suggestion-challenge)

** We had some problems uploading some of the files to GitHub since they were too large. Thus we only uploaded the most relevant ones that contain our final approach and logic. Any needed files for running the work from scratch will be noted with links to the relevant places we took them from, while others are generated from running the code (alternations of the preprocessed-data, and saved models)
** The Code also contains some sections that are commented out. These sections were used multiple times in order to preprocess the data or do some alternations of our process until achieving our last and best result.

The plots and visualizations are attached at the end of the file.

References

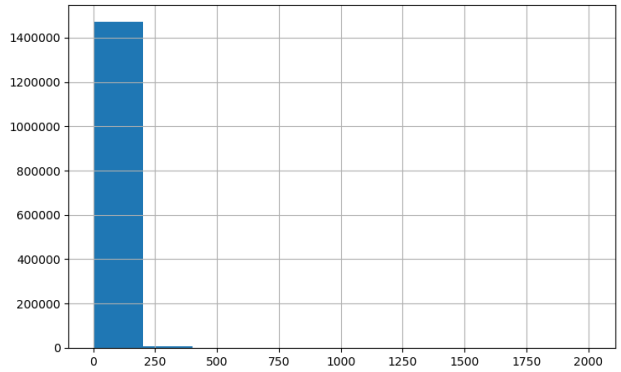
[1] Parameter selection
`https://machinelearningmastery.com/configure-gradient-boosting-algorithm`

[2] XGboost parameters tuning
`https://xgboost.readthedocs.io/en/latest/tutorials/param`$_t$*uning.html*

[3] FastText
`https://fasttext.cc`

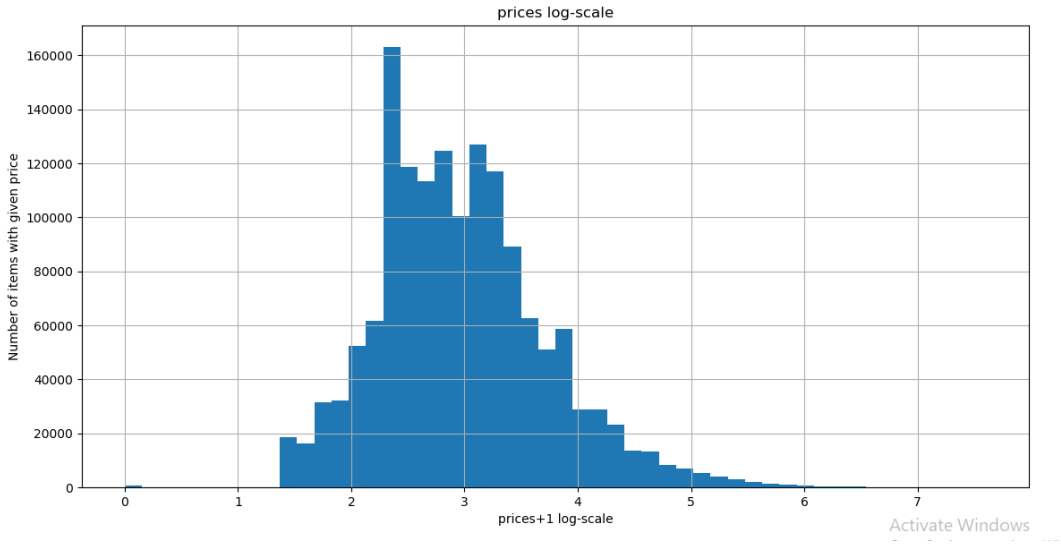
[4] InferSent Code
`https://github.com/facebookresearch/InferSent`

Presenting Information about the Data:

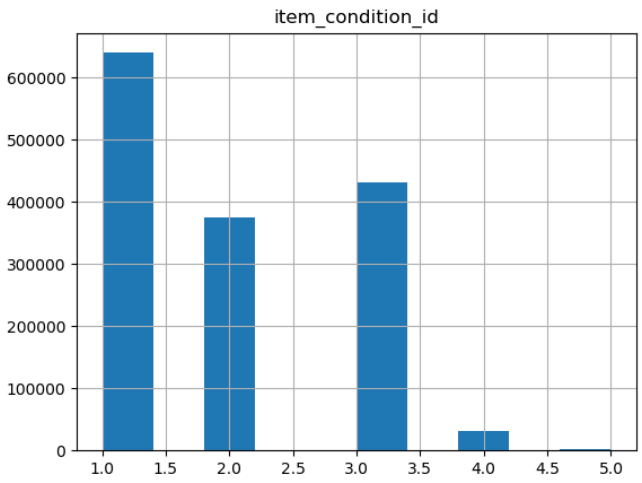



```
count    1.482535e+06
mean     2.673752e+01
std      3.858607e+01
min      0.000000e+00
25%      1.000000e+01
50%      1.700000e+01
75%      2.900000e+01
max      2.009000e+03
Name: price, dtype: float64
```

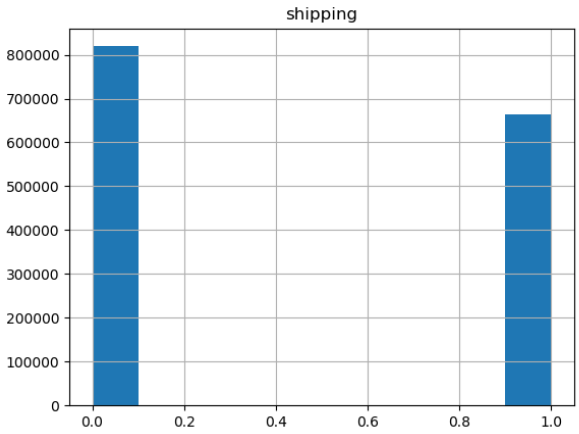
- It can be seen that Most prices of the items are ranged from 1$ - 300$, and another small fractions moves between 200-450 (approx.)
  This information could be useful to assist with evaluating a good\decent\really-bad model using their MSE\ MAE.



Presenting the log-scale of the prices values , can show us the distribution of the prices in a better way. (added +1 to each price to prevent zero values)



- We can see that almost half of the data rows contain item_condition=1. The other half have it's value on 2 / 3. Kind of a weak distribution but may still be helpful in our opinion in the prediction, as the condition of an item is generally highly correlated to its price.




```
Shipping distribution:
0    0.552726
1    0.447274
Name: shipping, dtype: float64
```

The 'shipping' feature is almost evenly distributed.

- Rest of the features are text, and therefore will be converted into vectors using some kind of sentence embeddings (Eventually used InferSent)

Some more general information:

The information is being calculated in ‘present_data.py’:

```
################################################
LOOKING ON THE DATA STRUCTURE:
################################################
data size:  1482535
```

```
############################################
LOOKING ON NUMBER OF UNIQUE VALUES IN EVERY FEATURE:
############################################
item_condition_id:  5
name:  1225273
category_name:  1288
brand_name:  4810
price:  828
shipping:  2
item_description:  1281427
General Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1482535 entries, 0 to 1482534
Data columns (total 8 columns):
train_id             1482535 non-null int64
name                 1482535 non-null object
item_condition_id    1482535 non-null int64
category_name        1476208 non-null object
brand_name           849853 non-null object
price                1482535 non-null float64
shipping             1482535 non-null int64
item_description     1482531 non-null object
dtypes: float64(1), int64(3), object(4)
```

Categoric features different values counting :

```
################################################
value_counts OF THE FEATURES:
################################################
1    640549
3    432161
2    375479
4     31962
5      2384
Name: item_condition_id, dtype: int64
0    819435
1    663100
Name: shipping, dtype: int64
```

Correlation Matrix of the initial numeric features:

	item_condition_id	shipping	price
item_condition_id	1.000000	-0.191154	-0.000807
shipping	-0.191154	1.000000	-0.097211
price	-0.000807	-0.097211	1.000000

We can see that shipping has a certain correlation with the price. Close to 10%.

The condition of the item (ranging from 1 to 5) not too much correlation by itself.

Checking each one of the other features correlation to the price may be possible with a linear regression model using the infersent encoding of the feature. Conclusions of this will be put here after all of the data encoding works (encoding too much data doesn't work on our computers).