Predicting Product Prices From Descriptions

Emily Kehoe (05864271, emkehoe), Ian Naccarella (05867462, inaccare), Javier Raygada (05894540, jraygada)

Github link for project: https://github.com/inaccare/Price-Suggestions

I. Introduction

Given the prevalence of online shopping, many online marketplaces are investigating ways to offer suggested prices to people selling items through their website as a guideline. For this project, our aim is to build an algorithm which automatically suggests prices to these online sellers based on the product descriptions, brand names, item conditions, etc. that they provide, thus allowing these sellers to save time and resources when determining the value of their products. This project is particularly interesting because while there has been a lot of work on pricing algorithms in areas such as financial markets and real estate, these two problems turn out to be quite different from predicting the prices of products. Furthermore, given the dearth of research papers available in this space, we feel that it's an untapped opportunity with potentially major possibilities for the rapidly growing world of online retail. Pricing objects comes up in almost all marketplaces, so it would be interesting to discover how various neural networks could compare to the status quo on sites such as Ebay, Craigslist, and Amazon and could potentially reduce any inefficiencies there.

II. Details of the Dataset

This problem is currently an open Kaggle challenge, the Mercari Price Suggestion challenge, and thus we obtained a dataset of ~1.4 million items along with their prices, descriptions, conditions, etc. from Kaggle. We then proceeded to split this dataset into train, dev, and test sets with a split of 98% | 1% | 1%. This split (implemented in makeCSVs.py) was chosen due to the large size of our dataset which allowed us to put most of the data into our train set and still be confident that our dev and test sets would be large enough to provide accurate information about the performance of our algorithm.

III. Approach

For our initial attempts, we chose to focus solely on the item descriptions because we believe them to be the most informative piece of the provided information. Given this decision, we chose to implement bag of words as a way of encoding product descriptions for our baseline model. This presented a challenge due to the size of our corpus of 1.4 million item descriptions which lead to our vocabulary being ~500,000 words. Thus, each input vector was the size of our vocabulary with a 1 or a 0 indicating whether or not a given word in the vocabulary appeared in the item description, making them very large and sparse and also requiring a lot of memory. To work around this memory consumption issue, we are instead storing the indices within the vocabulary corresponding to the words in each product description in a vector, passing that

vector into the minibatch *for loop*, and then building the larger input vector within each minibatch.

We chose to implement the binary bag of words model as opposed to frequency because the binary model tends to outperform the frequency model. This means that we are concerned with the occurrence of words in the product description as opposed to the frequency of words in the product description.

In order to evaluate this model, we chose to bucket the item prices across 12 buckets such that each bucket contains the same number of items. This allowed us to utilize a softmax activation output layer with 12 nodes and then calculate a softmax cross entropy loss. In the future, we plan to design our own activation function which will output an exact predicted price for an item rather than the price bucket the item is believed to be in. This will also necessitate a new metric to calculate the accuracy of our algorithm as we will no longer be able to simply check if the correct bucket was output. As a first step, we plan to use a least squares measurement.

This baseline algorithm achieved decent results given its restrictions, namely the amount of time it takes to train. Running a single epoch with the entire dataset took around 6 hours, and thus we were only able to run a single epoch on the entire dataset or two epochs on half the dataset. The results are provided in Section IV, but in both cases we achieved training set accuracy ~24% and dev set accuracy ~22%.

The next algorithm we implemented utilized word2vec vectors in an attempt to encode the meaning of the product description and also decrease the size of our input vectors. We used the glove.6B.100d.txt as our word2vec model and then for each product description, we averaged the vectors of each word in the description and then fed that averaged vector into the neural network. Prior research indicates that averaging word2vec vectors does an adequate job of encoding the combined meaning of the words, although we will improve on this in future iterations, potentially introducing tf-idf weightings as well.

This algorithm was evaluated in the same manner as bag of words and after running on half the dataset for 300 epochs, achieved both train and dev set accuracies of ~16%. This appears low given the performance of bag of words, but we believe this could be due to the input glove vector which was not trained on item descriptions in online marketplaces. We expect the accuracy to increase once we are able to refine these word vectors.

Our focus for this milestone was to have a baseline algorithm implemented and to have a multilayer perceptron network with 1 hidden layer using a relu activation and an output layer using softmax activation, which we achieved.

Going forward, we intend to build up our models as well as develop new ones. As mentioned before, we will refine the glove vectors we are using for word2vec and will also attempt to optimize our parameters as well as neural network architecture. Furthermore, we intend to develop an LSTM model with word2vec and then compare the accuracies of each of the

implementations. We expect that the bag of words model will produce the worst results and LSTM will produce the best results.

In our final model, we plan to expand our architecture to include about six hidden layers to make the model deeper than the baseline model and randomly sample to optimize the number of hidden units. We intend to replace our softmax output layer with a linear activation function. We also plan to design a linear activation function using least squares to estimate exact prices as opposed to bucketed prices. We will use mean square error cost and potentially construct a cost function that will better suit this output. In order to develop this new cost function, we will investigate research papers that designed the existing cost functions and explore a similar design process for ours. The mean square error cost function is as follows,

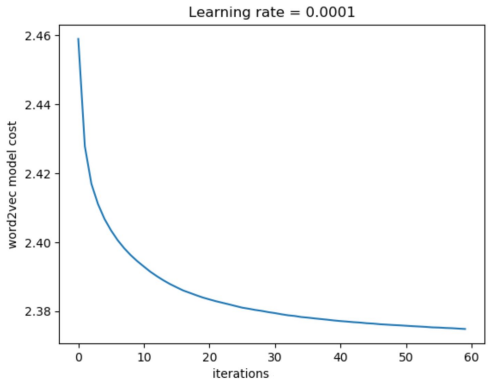
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

IV. Results

Train and Dev Accuracies for the Various Models

Model	**Train Set Accuracy**	**Dev Set Accuracy**
Bag of words (entire dataset, 1 epoch)	0.231	0.222
Bag of words (½ of dataset, 2 epochs)	0.242	0.216
Word2Vec (½ of dataset, 300 epochs)	0.166	0.164

Cost per Iteration for Word2Vec (½ of dataset, 300 epochs)



Softmax Cross Entropy Loss

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

V. Contributions Per Person

Javier: Cleaned and divided data, prepared for incorporation with model
Ian: Incorporated data with model, preliminary implementation of word2vec
Emmie: Implemented bag of words and set up initial model