CS230

# Product Price Suggestions for Online Marketplaces

**Emmie Elise Kehoe**
Department of Electrical Engineering
Stanford University
emkehoe@stanford.edu

**Ian Andrew Naccarella**
Department of Chemical Engineering
Stanford University
inaccare@stanford.edu

**Javier Raygada**
Department of Electrical Engineering
Stanford University
jraygada@stanford.edu

## Abstract

First-time sellers in online marketplaces often have difficulty determining a reasonable price point for their products. This fact, coupled with the emerging prevalence of online marketplaces, has led to significant interest in algorithmic methods for suggesting prices to these sellers which would reduce inefficiencies and provide a baseline for both buyers and sellers to work from. Using a dataset containing 1.5 million products along with their descriptions and prices [2], we trained neural networks with a variety of different architectures and input types in an attempt to solve this problem. Our best model encoded the item descriptions as word2vec vectors trained on our own corpus and fed these to an LSTM layer. The output of this layer was fed through three fully-connected (FC) layers which output either a linear or softmax function depending on the desired accuracy metric (see Figure 1). The softmax function sorted items into 12 price buckets and our model was able to achieve a training accuracy of 25.5% and a test set accuracy of 22.7% using this output. The linear function attempted to predict the specific price of the product using a Root Mean Squared Logarithmic Error cost function and a percent error accuracy function. We were able to achieve a cost of 0.58 (which falls within the top half of competitors in the Kaggle competition we entered) and a percent error of 48%. Thus, while there are definitely improvements to be made, we believe this work to represent notable progress towards solving this highly complex problem.

## 1 Introduction

Given the current pervasiveness of online shopping, many online marketplaces are investigating ways to offer suggested prices to people selling items through their website as a guideline. This guideline would help improve the efficiency of these marketplaces as it would allow sellers to estimate the price point for their specific product more quickly than if they were required to search online for the price points of similar products. In order to address this in our project, our aim was to build an algorithm which automatically suggests prices to these online sellers based on the descriptions, brands, conditions, and categories of their products, 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

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

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 potentially reduce inefficiencies there.

Table 2 shows an example of a product as provided in the dataset we received from the Mercari Price Suggestion Kaggle challenge [2]. In our initial models, we vectorized the item description using either a bag of words or word2vec representation and then fed these into a neural network. The more advanced models (Figure 1) fed word2vec representations of the item description into an LSTM layer. The final output of this layer was then fed through three fully-connected (FC) layers and the output of this was fed, along with vectorized forms of the item condition, category name, and brand name (Table 3), through another three FC layers which output either a softmax function or a linear function. The softmax function sorted the products into 12 different price buckets while the linear function attempted to output the actual price of the product. In order to evaluate these models we determined the classification accuracy of the models using a softmax output and the percent error for the models using a linear output.

## 2 Related work

It was challenging to find literature that addressed our problem so we stitched together ideas from disparate areas including sentiment classification and housing models to develop our baseline model. There were three elements from literature that we found useful: word embedding, model architecture, and linear output activation functions.

The first element that we explored was word embedding. We found literature that argued for training word2vec vectors on GloVe rather than skip-gram with negative sampling (SGNS) [5]. However, we also found the contrary with evidence supporting SGNS over GloVe [3]. After consulting the notes for the class Natural Language Processing with Deep Learning (CS 224N), we decided to train our word vectors using the GloVe since this approach seems more common.

The second element was developing a model architecture. We found a simple architecture used in sentiment classification for documents which sent sentences into a gated recurrent unit (GRU) and made predictions from the last output as a section of their larger model [7]. Additionally we found that when using product descriptions to predict sales, an attention model was used [6]. Since we were tackling a simpler text analysis task with short product descriptions, we decided to implement the simpler network using an LSTM unit. From here, the next step would be to implement an attention model.

The last element was developing a linear output activation function. We found that the housing model used a linear output function with root mean squared error [4]. In our activation function, we used a linear output as well, but we measured cost using root mean squared logarithmic error in order to compare our results with the Kaggle competitors.

## 3 Dataset and Features

As this problem was an open Kaggle challenge, the Mercari Price Suggestion challenge, we obtained a dataset of $\approx 1.5$ million items along with their prices, descriptions, conditions, etc. from Kaggle [2]. We proceeded to split this dataset into train, dev, and test sets with a split of 98% | 1% | 1%. This split was chosen because the size of our dataset made us confident that our dev and test sets would be large enough even at 1% to provide an accurate metric for the performance of our models. Each of the provided products came with name, item condition, category, brand, price, shipping, and description (Table 2).

We then required several steps to preprocess the data into a form we could feed into our model (Table 3). First, we encoded the item descriptions in two different manners. For our binary bag of words implementation, we built a vector the size of our vocabulary and inserted a 1 or a 0 at each index corresponding to whether or not the word associated with that index appeared in the item description. While this implementation achieved high accuracies, it ran very slowly due to the large input size (500,000), forcing us to utilize word2vec encodings. Our first word2vec encodings came from a pre-trained GloVe vector (glove.6b.100d) and then later on we trained these word2vec vectors on our own corpus of item descriptions which nearly doubled our classification accuracies. In both cases, they were 100-dimensional vectors which attempted to capture the meaning of the item description.

In our initial implementations, we simply averaged these word2vec vectors, but upon switching to an LSTM model, we began feeding each vector individually to a cell so as to capture the series data. Because our LSTM model could only handle inputs of a single length, we initially tried zero-padding all item descriptions to the max length, 412 words. However, we found that when

we truncated to the 95th percentile, 72 words, we got similar accuracies and trained faster.

Lastly, we encoded the other inputs (brand, item condition, and categories) so as to feed them into a fully-connected layer. Since there were only 5,000 brands, these were encoded as a one-hot vector. Similarly, item conditions ranged from 1 to 5 so a one-hot vector was used there as well. Since there were multiple categories to which an item could belong we decided to encode category as a multi-hot vector with a 1 or 0 corresponding to whether the item belonged to each category.

## 4 Methods

In total, we tested 13 different models throughout this process (see Table 1 for detailed descriptions). Initially we chose to focus solely on the item descriptions because we believed they would provide the most information. Using the aforementioned binary bag of words (model 1) and averaged word2vec (model 2) encodings as inputs, we ran a simple 3-layer neural network to achieve our initial accuracies (see Figures 3 and 4 for accuracies). These models were trained using a softmax cross entropy loss function and the accuracies were evaluated using a softmax output function to sort the products into 12 price buckets. Though bag of words gave significantly higher accuracies, it took over 8 hours to run one epoch, forcing us to switch to word2vec for future models.

Next, we sought to capture the order of the words in the item description rather than simply averaging them all together and losing the meaning encoded in the ordering. To do this, we built the LSTM layer described in the previous section. We fed both pre-trained GloVe vectors (model 3) and word2vec vectors which we trained on our own corpus of product descriptions (model 4) into this model. The vectors trained on our own corpus has accuracies nearly double those of the pre-trained vectors which makes sense given that product descriptions tend to differ significantly from standard writing.

We then incorporated the other provided data into our model by taking the last LSTM output and feeding it into a fully-connected (FC) layer along with the vectorized forms of the item condition, brand, and category (model 5). Doing this actually decreased our accuracies which indicated that we were correct in assuming the item description to be the most important component.

At this point, we also began incorporating alternative output functions so as to better capture the efficacy of our model. We expanded our softmax output function to 20 buckets (model 6) and also utilized a linear output function (model 7) which used an RMSLE cost function (Figure 2) and a percent error accuracy function. Because these two output functions gave us new insights into the accuracy of our model, we decided to continue to use them for future evaluations.

Next, we further expanded our model by replacing the single FC layer with 3 FC layers in an attempt to better learn from the item condition, brand, and category. As mentioned before, we tested this model using softmax functions with 12 buckets (model 10) and 20 buckets (model 8) and also a linear function (model 9). However, these models still achieved worse accuracies than when we only fed in the item description and so we decided to revert back to that.

Our final set of models thus incorporated an LSTM model with word2vec inputs which fed into 3 FC layers and output either a classification into 12 buckets (model 11), 20 buckets (model 12), or a linear function (model 13). Model 11 achieved the highest accuracies of the models we tested, and thus we decided to tune the hyperparameters of this model and also train it for longer (150 epochs vs 50 epochs for each of the other models). We found that this did not significantly improve the accuracy of the model, and led to a final classification accuracy of 22.7% on the dev set compared to 21.5% before tuning.

## 5 Experiments/Results/Discussion

The results for each of our models can be found in Figures 3 and 4. Our initial hyperparameters (learning rate, batch size, and LSTM cutoff length) were tuned on model 10 and then utilized for each of the other models so that we could compare them effectively. Because standard performance metrics, such as precision and F1 score, did not apply to the task we were attempting to solve, we derived our own metrics. Thus, to evaluate our models we used a simple classification accuracy for the softmax output models and percent error for the linear output models. These were chosen due to their simplicity which made it easy to gauge how well they were performing relative to both each other and human-level performance.

There were several notable results from our experiments. First, we significantly improved our accuracies by training our word2vec vectors on our own corpus rather than using pre-trained vectors. This implies extensive differences in the meaning of words in an product description setting versus a more formal setting.

Furthermore, as has been already noted, the addition of item condition, category, and brand decreased our classification accuracy. This was highly unexpected since each of these inputs contains valuable information which we thought would improve accuracy. If we had more time we would have investigated this further to determine the cause. Lastly, when we began incorporating our LSTM model, we noticed significant differences between train and dev accuracies which indicated overfitting. To resolve this we added dropout to our network which effectively prevented this. We also added Xavier initialization and utilized an Adam optimizer in order to speed up training as our initial models were not converging very quickly.

## 6 Conclusion/Future Work

In conclusion, we believe that we made substantial progress in solving the problem of price prediction, but that there is still a significant way to go. Through testing we performed on ourselves, we learned that human classification accuracy with 12 price buckets is 35% which is not that much higher than our highest accuracy. Furthermore, this problem is complicated by changing tastes which vary the cost of items over time. However, we believe we have made valuable strides. We learned that training the word2vec vectors on the proper corpus is enormously beneficial and that item description is disproportionately valuable in determining the price of an item.

We believe that the immediate next step would be to incorporate attention into our model to emphasize the parts of the product description which provide the most information. The focus of Pryzant's paper was using attention to predict product sales from product descriptions [6], which indicates the efficacy of this method. Additionally, since our bag of words description embedding achieved such a high accuracy, a next step in sentence embedding would be to incorporate Facebook's fastText word classifications. These can be trained on large corpuses in under a minute, and could give embeddings that produce better accuracies since the technique used is bag of n-grams, which is bag of words but captures local order as well [1]. Given more time, it would also have been helpful to examine which examples our model misclassified and attempt to determine why this was the case.

## 7 Contributions

Javier: Cleaned and divided data, prepared for incorporation with model, developed solution to memory complexity problem which allowed our training speed to increase. Developed a better bucketing system from 12 buckets to 20 buckets. Made JSON files to iterate through hyperparameters. Debugged LSTM model. Incorporated inputs other than item description into models.

Ian: Incorporated data with model, word2vec implementation with training on GloVe vector and training on own corpus. Implemented linear activation function for predicting prices. Made the poster.

Emmie: Implemented bag of words and set up the multilayer perceptron model and LSTM model as well as deepened the LSTM model with fully-connected layers and deepened the network with additional inputs with fully-connected layers. Set up the script to save the graph to replicate our model's results.

We think that the work was split evenly across team members. In particular, the boundaries delineated above are soft since we helped one another whenever one ran into trouble with one of his/her tasks.

## 8 Appendix

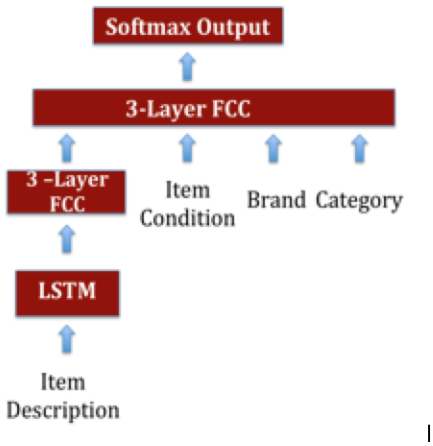


Figure 1: Architecture of one of our best models.

The RMSLE is calculated as

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

Where:

$\epsilon$ 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, and
$a_i$ is the actual sale price for $i$.
$\log(x)$ is the natural logarithm of $x$

Figure 2: Root Mean Squared Logarithmic Error Explained.

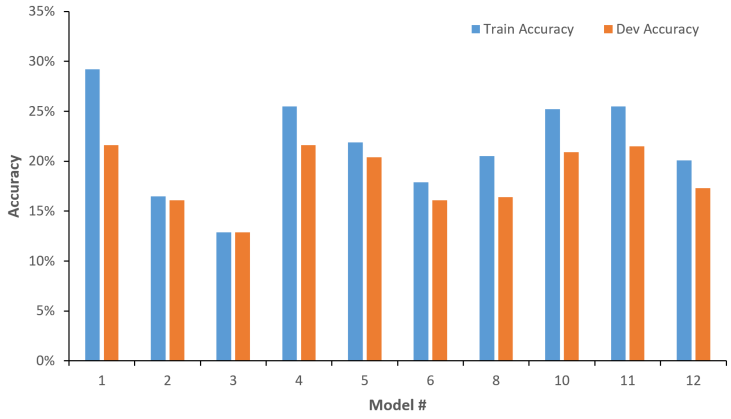


Figure 3: Results for models with softmax outputs.

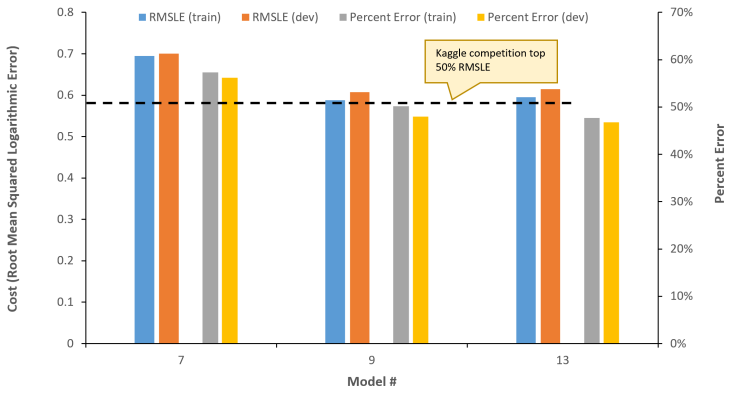


Figure 4: Results for models with linear outputs.

| Model Number | Model Description |
|---|---|
| 1 | Takes in BoW encodings and passes these through 2-layer FCC to output a softmax vector of length 12. |
| 2 | Takes in average w2v encodings and passes these through a 2-layer FCC to output a softmax vector of length 12 |
| 3 | Takes in product descriptions and passes these through an LSTM cell. Output of this LSTM cell then gets passed to a softmax output layer. Output is a softmax vector of length 12. |
| 4 | Same as model 4 except that it uses w2v encodings based on our corpus. The rest of the models use our trained w2v encodings. |
| 5 | Takes in product descriptions and passes these through an LSTM cell. Output of this gets concatenated with vectorized form of other inputs (brand, categories, and condition) before being put through a softmax output layer. Output is softmax vector of length 12. |
| 6 | Same as model 5 except that output softmax vector is of length 20 instead of 12. |
| 7 | Same as model 5 except that output layer is linear and thus output for each sample is a scalar corresponding to predicted price. Cost function used for all linear outputs used is root mean squared logarithmic error. |
| 8 | Takes in product descriptions, passes these through LSTM, followed by 3-layer FCC network. Output of this gets concatenated with other inputs (brand, condition and categories) before being passed through a final 3-layer FCC. Output layer is a softmax layer. Output for each sample is a softmax vector of length 20. |
| 9 | Same as model 8 except that output layer is linear and thus output is a scalar value corresponding to predicted price. |
| 10 | Same as model 8 except that output softmax vector is of length 12 instead of 20. |
| 11 | Descriptions get fed into an LSTM cell. Output of this then gets fed into 6-layer FC network. Output is a softmax vector of length 12. |
| 12 | Same as model 11 except that output is a softmax vector of length 20 instead of 12. |
| 13 | Same as model 11 except that output layer is linear and thus output value for each sample is a scalar corresponding to predicted price. |

Table 1: Description of each model according to model number.

| Train ID | Name | Item Condition | Category Name | Brand Name | Price | Shipping | Item Description |
|---|---|---|---|---|---|---|---|
| 396558 | PINK-size... | 2 | Women/Athletic... | PINK | 14 | 0 | 2-pair of size small PINK... |

Table 2: Sample product inputs from Kaggle Dataset.

| Encodings | word2vec | categories-indices | brand-indices | price-bucket |
|---|---|---|---|---|
| [3 2 1 1...] | [-0.19541138 0.25392095 ...] | [ 3 35 40] | 11 | 7 |

Table 3: Additional features added to dataset to reduce computational complexity when running models. Encodings column denotes non-zero indices for Bag Of Words vectors. word2vec column denotes average w2v encodings for a given description. Categories indicies denotes non-zero indices for categories vectors. Price bucket denotes price bucket in which the product falls into.

**Link to GitHub repository: https://github.com/inaccare/Price-Suggestions**

## References

[1] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv*:1607.01759.

[2] Kaggle. (2018). Mercari Price Suggestion Challenge. Retrieved January 15, 2018, from https://www.kaggle.com/c/mercari-price-suggestion-challenge/data.

[3] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. TACL.

[4] Limsombunchai, V., Gan, Ch. & Leem M. (2004). House Price Prediction: Hedonic Price Model vs. Artificial Neural Network. *American Journal of Applied Sciences* 1.

[5] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar, October. Association for Computational Linguistics.

[6] Pryzant, R., Chung, Y. J., & Jurafsky, D. (2017). Predicting Sales from the Language of Product Descriptions.

[7] Tang, D., Qin, B., & Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In EMNLP.