Deep learning model for Sales Price Suggestion

Jiachuan Deng
Department of Computer Science
Purdue University
deng159@purdue.edu

Abstract

We introduce a deep learning framework model trained on top of pre-trained word vectors with textual (review) and categorical data for commodity price prediction task. Different deep learning modules (CNN and GRU) are experimented, and simple CNN structure with little hyperparameter tuning is shown can achieve excellent performance. Skip structure and Empirical Bayes tricks are used to enhance the model performance. And we additionally propose a simple but effective trick to deal with missing value in this specific task.

1 Introduction

Knowing how much something really worth is difficult. Small details can mean big differences in pricing. Product pricing gets even harder at scale, considering just how many products are sold online.
Fortunately most products also provide textual (reviews) and categorical (condition, shipping, etc) information, which can be easily modeled by deep learning tricks, although most categorical data commonly suffers missing value problem.
In this paper, we propose a simple but very effective way to deal with categorical missing value, and two types of deep learning structures CNN and GRU will be discussed on dealing with the textual data. Embedding method as described in Guo and Berkhahn (2016) and Empirical Bayes method as described in Micci-Barreca (2001) will be applied in dealing with categorical data. In order to enhance model performance, skip structure is used.

2 Method

2.1 Data Set

The data set we use is from Kaggle Mercari Price Suggestion Challenge [1]. It consists of two textual features: **name**, **item_description**, and four categorical features: **brand_name**,**category_name**, **shipping**,**item_condition_id**, and a float value we want to predict: **price**.

2.2 Deal with Missing Value

Some categorical features is extremely important compared with others. The feature brand_name should be one of the most important attributes in the dataset, since different brand name may result in huge difference in price. For example, the price of a H&M bag in general should be much lower than a Louis Vuitton bag. However, the brand_name feature suffers severe missing value problem. Fortunately, by using textual information (reviews), we propose a simple but very effective algorithm

[1] https://www.kaggle.com/c/mercari-price-suggestion-challenge#description

10807 Class Project (2016).

1 that can help us to ameliorate this issue .

```
brandnames=new list;
while not at end of dataframe do
    read current brand_name;
    if current brand_name not NAN then
        brandnames.add(current brand_name)
    else
        continue;
    end
end
while not at end of dataframe do
    read current brand_name;
    read current review;
    if current brand_name is NAN then
        guessbrand=search all brand_name in brandnames from current review;
        if guessbrand not NAN then
            fill current brand_name with guessbrand;
        else
            fill current brand_name with <UNK>;
        end
    else
        continue;
    end
end
```

Algorithm 1: Guessing Missing values Algorithm

This guessing process helped us find about 2/3 missing brand values.

2.3 Model Architecture

The model architecture, shown in Fig 1, can be mainly separated into three parts: 1. Textual module, 2. Categorical module, 3. Meta data module, 4.Skip module. Each of the modules are modeled with different deep learning structure and tactics.

2.3.1 Textual Module

This part modeled textual data, i.e. Name, Description. In order to represent textual information efficiently, Name and Description are represented by word embedding with a shared embedding lookup table. In our implementation, word embedding uses pre-trained GLOVE Pennington et al. (2014) embedding to initialize the embedding lookup table. Let $G \in R^{N \times k}$ be the pre-trained GLOVE Embedding and $We \in R^{N' \times k}$, where N is the vocabulary size, N' is the vocabulary size of all reviews, and k represents embedding dimension. For word w is in GLOVE Embedding, initialization is done by $We[w] = G[w]$ and for word w' is not available in G, initialization is done by $We[w'] \sim N(\mu, \sigma)$, where $\mu = mean(G)$ and $\sigma = std(G)$.
There are two optional deep learning modules that can be used to model textual data here, CNN and RNN (GRU).

1. **Recurrent Neural Network-based:** RNN is a traditional way to deal with textual data. For sentence i (either Name or Description) s_i, is represented as sequence $[x_i(0), x_i(1), ...x_i(m)]$ after Embedding lookup, where $x_i(j) \in R^{1 \times k}$ is the corresponding word embedding representation of sentence i's j^{th} word. The output of GRU given x_i is:

$$o_i, h_i = GRU(x_i(0), x_i(1), ...)$$

Where $o_i \in R^{m \times k}, h_i \in R^{m \times k}$ is the output and hidden states after GRU. Instead of only using GRU's hidden layer features, we also applied the concat pooling trick as suggested in Howard and Ruder (2018). Which means we use a vector representation T_i to represent

textual information from sentence i.

$$maxpool_i = max(o_i, axis = 0) \in R^{1\times k}$$

$$mean_i = mean(o_i, axis = 0) \in R^{1\times k}$$

$$T_i = h_i[m] \oplus maxpool_i \oplus meanpool_i \in R^{1\times 3k}$$

where $\oplus$ is concatenate operator.

2. **Convolutional Neural Network-based:** CNN on the other hand, is usually used in dealing with image data. But inspired by Kim (2014), CNN can also be a good option here as long as sentences are properly padded. To use CNN in our architecture, simply replace the RNN component in Fig 1 with structure shown in Fig 2. Here sentence i with length m (padded where necessary, in our task we set m=70 for Description and m=20 for Name) can be represented by

$$X_{i,1:m} = x_i(0) \oplus x_i(1) \oplus ... \oplus x_i(m) \in R^{m\times k}$$

A convolution operation involves a filter $w \in R^{h\times k}$, which is applied to a window of h words to produce a new feature. For example, a feature c_j is generated from a window of words $X_{i,j:j+h-1}$ by:

$$c_j = f(w \cdot X_{i,j:j+h-i} + b)$$

Here $b \in R$ is a bias term and f is a non-linear function. And therefore, given a filter w, sentence i can be extracted as a feature map

$$M_h = [c_1, c_2, ...c_{n-h+1}]$$

Using different window size h can give us different feature map. When h=1, h=2, h=3 it can be regarded as 1-gram, 2-gram ...language model. And the final vector representation we use for sentence i T_i is given by

$$T_i = maxpool(M_1) \oplus maxpool(M_2) \oplus maxpool(M_3)$$

Where M_1, M_2, M_3 is feature map with window size 1, 2, 3 respectively.

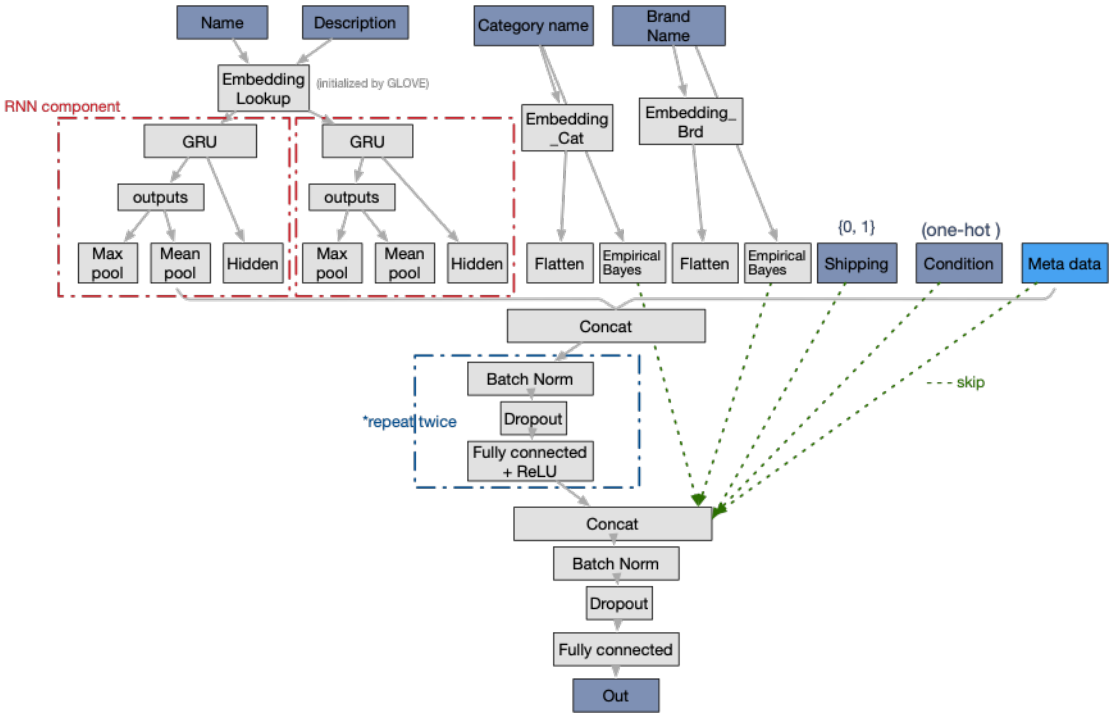

Figure 1: Deep learning model framework

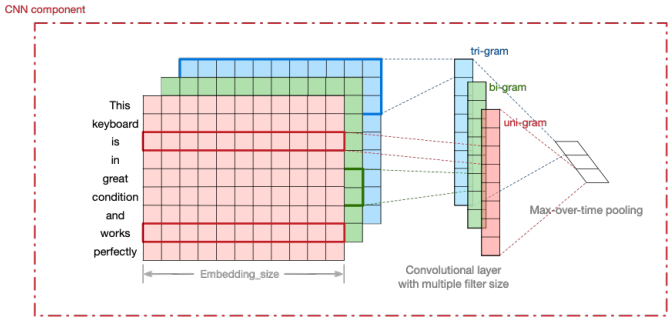

Figure 2: CNN component for textual feature

2.3.2 Categorical Module

For low-cardinality categorical attributes the most widely used numerical representation method is the one hot encoder. Here shipping and condition use one-hot-encoding to represent. For high-cardinality categorical attributes category_name and brand_name, using one-hot-encoding will make the input matrix be too sparse. We apply same way as Guo and Berkhahn (2016), categorical embedding is used for these two attributes.
Empirical Bayes method as described in Micci-Barreca (2001) is applied on category_name and brand_name together with embedding. The Empirical Bayes method has been proved to be a simple but effective preprocessing scheme for high-cardinality categorical data that allows this class of attributes to be used in predictive models. For example, if we want to compute empirical bayes value of categorical feature a's value v.

$$\lambda(a,v) = \frac{1}{1+e^{-\frac{(n-k)}{f}}}$$

$$\mu(a,v) = mean(y_v)$$

$$prior(a) = mean(y)$$

$$Eb(a,v) = (1-\lambda(a,v)) \cdot prior(a) + \mu(a,v) \cdot \lambda(a,b)$$

where n=#feature a with value v, y is target label/value(in this task price to be predict) of columns whose feature a, and y_v is those y whose feature a's value is v. k and f are hyperparameters minimal leaf sample size and smoothing parameter.

2.3.3 Meta Data Module

Meta data is additional features that we extracted from original data in feature engineering way. Given category name c, brand name b, shipping sp, they can be calculated as following SQL style commands:

$$aggDF = dataframe.groupby([c,b,sp])$$

$$count = aggDF.count()$$

$$mean = aggDF.mean()$$

$$median = aggDF.median()$$

$$std = aggDF.std()$$

$$meta = count \oplus mean \oplus median \oplus \ mean + 2 \cdot std \oplus mean - 2 \cdot std$$

2.3.4 Skip Module

Some part of features are more important than others, especially meta data, as shown in Joshi et al. (2010). It can be simply implemented by concatenating the "important features" right before last Fully Connected Layer, to enhance such features' influence in model.

2.4 Optimization Objective

Let $\hat{y}_i \in R$ is the predicted $commodity_i$'s price given Textual Data (review, name of $commodity_i$) t_i, and categorical data (shipping, condition, etc.) c_i , θ is model's parameters, f(.) represents the model. Our optimization goal is to find θ^* so that it can minimize Root Mean Squared Logarithmic Error **L** over N commodities between $\hat{y}_i$ and true price y_i.

$$\hat{y}_i = f(t_i, c_i; \theta)$$

$$L = \sqrt{\frac{1}{N}\sum_i (log(\hat{y}_i + 1) - log(y_i + 1))^2}$$

$$\theta^* = argmin_\theta L$$

3 Experiments

Baseline Model

XGBoost: The model defined by K trees (parameter $n\_estimators$), the probability of each sample is given by $\hat{y}_i = \sum_k f_k(x_i)$, $f_k()$ here means the k^{th} decision tree.
XGBoost's objective function is defined by : $Obj = \sum_i l(y_i, \hat{y_i}^{t-1}) + \sum_k \Omega(f_k)$, here $l()$ is the training loss, here $l()$ is RMLE. $\Omega()$ means the complexity of Trees.
Since XGBoost can not deal with textual data directly, we simply ignore the textual data Name and Description, the model input x_i here is only consists of the categorical features processed by Empirical Bayes as described in Section **2.3.2** and meta data as described in Section **2.3.3**.

3.1 Comparison with Baseline Model

The model we use here is with RNN/CNN (with only one kernel [3,embedding size]) architecture and Skip structure, we compare it with the baseline XGBoost model to show our model's effectiveness. 5-fold cross validation is done on both models, results shown below is on Test set.

	Fold-1	Fold-2	Fold-3	Fold-4	Fold-5	mean	std
XGB	0.5651	0.5660	0.5667	0.5680	0.5648	0.5662	0.0012
RNN	0.4265	0.4264	0.4259	0.4262	0.4258	0.4261	0.0002
CNN	0.4254	0.4255	0.4256	0.4251	0.4252	0.4253	0.00018

The results show with 95% confidencem XGBoost model can achieve MSLE 0.5662(+/-)2$\times$0.0012 . Our model with RNN architecture can achieve MSLE with 0.426 (+/-) 2$\times$0.0002, and the CNN architecture can achieve 0.4253 (+/-) 2$\times$ 0.00018 MSLE.
Our model significantly outperforms the baseline model. And the CNN architecture performs slightly better than RNN architecture, while CNN has much lower computation complexity.

3.2 Number of Convolutional Kernels

In this part, we evaluate CNN architecture model with different number of convolutional kernels. To be specific, we experiment on model with only 1D convolution whose kernel size is [3,embedding_size] (denote as CNN1Kernel), model with both kernel size [3,embedding_size] 1D convolution and kernel size [2,embedding_size] 1D convolution (denote as CNN2Kernel), and model with [3,embedding_size],[2,embedding_size], [1,embedding_size] 1D convolutions (denote as CNN3Kernel).

	Fold-1	Fold-2	Fold-3	Fold-4	Fold-5	mean	std
CNN1Kernel	0.4254	0.4255	0.4256	0.4251	0.4252	0.4253	0.00018
CNN2Kernel	0.4223	0.4230	0.4225	0.4220	0.4223	0.4224	0.0003
CNN3Kernel	0.4205	0.4198	0.4201	0.4211	0.4203	0.4203	0.0004

The results show with 95% confidence, CNN1Kernel MSLE is 0.4253 (+/-)2$\times$0.00018, CNN2Kernel 0.4224(+/-)2$\times$0.0003, CNN3Kernel 0.4203(+/-)2$\times$0.0004. It can still confidently say CNN architecture with more different 1D convolutions performs better. This can be explained when using three 1D

convolutions, we are actually ensemble 1-gram, 2-gram and 3-gram language models and train them together.

3.3 Skip Structure

Skip structure can be applied on important features to enhance their influence on final result. The table shown below is results on test set 5-fold cross validation with 95% confident.

	RNN	CNN1Kernel	CNN2Kernel	CNN3Kernel
No Skip	$0.4265 \pm 2 \times 0.0002$	$0.4258 \pm 2 \times 0.0002$	$0.4230 \pm 2 \times 0.0003$	$0.4214 \pm 2 \times 0.00035$
With Skip	$0.4261 \pm 2 \times 0.0002$	$0.4253 \pm 2 \times 0.00018$	$0.4224 \pm 2 \times 0.0003$	$0.4203 \pm 2 \times 0.0004$

The results show by adding connection between meta data and last fully connected layer, model performance can be improved slightly.

4 Conclusion

In the present work we have described a series of experiments with our deep learning model with CNN/RNN architecture trained based on pre-trained word embedding vectors. We showed our model performs quite well compared to baseline XGBoost model, which means our deep learning model can model textual and categorical data well. We also showed that CNN architecture could be a better choice in our task, not only can reduce computation complexity, but also improves performance slightly. Additionally, skip structure is shown to be a effective way to enhance important feature's influence to the model's final output.

References

Cheng Guo and Felix Berkhahn. 2016. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737* (2016).

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 328–339.

Mahesh Joshi, Dipanjan Das, Kevin Gimpel, and Noah A Smith. 2010. Movie reviews and revenues: An experiment in text regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 293–296.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

Daniele Micci-Barreca. 2001. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter* 3, 1 (2001), 27–32.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.