Predict the Relevance of Search Results on Homedepot.com

Luyang Chen
ICME, Stanford University
lych@stanford.edu

Ruoxuan Xiong
MS&E, Stanford University
rxiong@stanford.edu

1 Introduction

As more and more people shop online, customers pay more attention to online shopping experience. Many bad shopping experiences come from the difficulty in finding the right products. When customers are unfamiliar or unclear with the products they want, it makes the searching harder and the shopping experience annoying. However, if online retailers can more accurately predict the relevance between search terms and products and pop out the products that can better match customers' need, this is extremely attractive and interesting. Therefore, many online retailers are working on such a relevance model.

For large online retailers, they have a large database of search terms and products with descriptions and attributes, which makes it possible to develop a good relevance model. The relevance between search/product pair can be evaluated by human raters. However, human rating is a slow and subjective process. There are almost infinite search/product pairs so it is impossible for human raters to give a relevance score to all possible search/product pairs. A better way is to build a model based on search/product pair and its relevance score and use it to predict the relevance score of out-of-sample pairs.

To rate a pair, human raters first need to understand the product functionality by reading the product description and attributes and then give a score. Therefore, the first step in the relevance model is to some vectors to represent these product information. The natural way to implement it is to use word-vector representation, such as SkipGram, GloVe, and then aggregate the vectors coming from all the words in description and attribute terms or search terms into a single vector. The next step in the relevance model is to give a relevance score based on the search vector and product vector. The process of comparing and giving a score requires mental efforts and can be regarded as a nonlinear process. Therefore, a nonlinear model, such as a deep learning model, may be better at mimicking the human understanding and comparing process, and predicting the relevance score of search/product pairs.

2 Problem Statement

The Home Depot, a chain home improvement retailer held a competition on Kaggle asking for Kaggler to develop a model that can accurately predict the relevance of search results. [1] The data provided are as follow:

2.1 Product Descriptions

There are 124428 products. Each product has a unique ID and a text description.

[1]https://www.kaggle.com/c/home-depot-product-search-relevance

2.2 Attributes

Most products are given extended information like bullet points of product functionality descrptions, size, color, material, etc.

2.3 Traing/Test Set

The training and test sets both contain search/product pairs and their corresponding relevance scores. The relevance score is between 1 (not relevant) to 3 (perfect match). A score of 2 represents partially or somewhat relevant. Each search/product pair was evaluated by at least three human raters. The provided relevance scores are the average value of the ratings. There are 74067 pairs in the training set and there are 166694 pairs in the test set.

The metric to evaluate the prediction errors is Root Mean Square Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

A trivial prediction is $\hat{y} \equiv 2$, which guarantees RMSE to be at most 1.

3 Technical Approach and Models

3.1 Baseline Model

We use a simple regression model as a baseline. For each search/product pair, we count the average number of times of a word in search keywords

- **exactly** showing up in product attributes, denoted as n_{ea}.
- **exactly** showing up in the product description, denoted as n_{ed}.
- being **part** of words in product attributes, denoted as n_{pa}.
- being **part** of words in the product description, denoted as n_{pd}.

We take these four counts as features of search/product pairs, add a nonlinear function to counts, and run a linear regression model (see `search.py`).

$$s = [a_1, a_2, a_3, a_4][f(n_{ea}), f(n_{ed}), f(n_{pa}), f(n_{pd})]^T + b_0$$

where $f(x) = \min(x, M)$. The reason to add this f-function is because the marginal increase of relevance score by increasing counts is trivial when counts are greater than 0.

3.2 Representation: GloVe

In order to use deep neural network models, the first thing we do is training word vectors. Instead of using pretrained word vectors, we build word cooccurence matrix based on our dataset and use GloVe to train the word vectors. The reason is that there are some special strings, say ”4×4”, which apparently do not appear in the pretrained word vectors, but they may contain key information, say when people search for something that is square, so we do not want to use '<unknown>' to represent these strings. We would like to represent each word as a row vector $w \in \mathbb{R}^d$ through minimizing the loss function

$$J = \sum_{i,j=1}^{|V|} f(P_{ij})(w_i w_j^T - \log P_{ij})^2$$

where $|V|$ is the number of words, P_{ij} is the probability of j^{th} word showing up in a window of length l of i^{th} word. We get the co-occurrence matrix X_{ij} by counting the number of times when j^{th} word shows up in a window of i^{th} word from the product description file `product_descriptions.csv` and the product attribute file `attributes.csv`. Then we approximate P_{ij} using

$$P_{ij} = \frac{X_{ij}}{\sum_{k=1}^{|V|} X_{ik}}$$

We implement the GloVe model by ourselves (see `train_GloVe.py`). In our implementation, we choose the dimension of word vector $d = 100$ and function f to be

$$f(x) = \begin{cases} 0 & x < 0 \\ 2x & 0 \leq x \leq 0.5 \\ 1 & x > 0.5 \end{cases}$$

Since the word vectors trained here only serve as initializers, we will not train this model for too much time. It is also why we only train one set of word vectors.

3.3 NN_Model

Now we build our first neural network model. For each search/product pair, we use three vectors $v_S \in \mathbb{R}^d$, $v_A \in \mathbb{R}^d$ and $v_D \in \mathbb{R}^d$ to represent search keywords, product attributes and product descriptions respectively by simply averaging the word vectors of those words that shows up. We treat these three vectors as inputs to our neural network.

The hidden layer is defined as follows:

$$h_1 = \text{sigmoid}(v_A W_A + v_S U)$$

$$h_2 = \text{sigmoid}(v_D W_D + v_S U)$$

where $W_A \in \mathbb{R}^{d \times h}$, $W_D \in \mathbb{R}^{d \times h}$ and $U \in \mathbb{R}^{d \times h}$, h is the dimension of the hidden layer. We hope h_1 to capture the relationship between product attributes and search keywords, while h_2 can capture the relationship between product descriptions and search keywords.

We predict the relevance between search keywords and product item by multiply h_1 and h_2 with a vector $u \in \mathbb{R}^{2h}$:

$$\hat{y} = [h_1, h_2] u^T$$

The loss function is defined as root mean square error (RMSE) plus regularizaiton:

$$L = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} + \frac{\lambda}{2} (\|W_A\|_F^2 + \|W_D\|_F^2 + \|U\|_F^2 + \|u\|_F^2)$$

where n is the number of training examples (i.e. the number of keyword-product pairs), y_i is the relevance evaluated by human raters, and λ is the regularization constant.

In our implementation (see `NN_model.py`), we choose the hidden dimension $h = 100$ and regularization constant $\lambda = 0.01$.

3.4 RNN_Model

Instead using the average of word vectors appearing in the search terms or product attributes/description, we can take in one word vector at a time, and choose the last output vector to represent the search terms or product attributes/description. For example, search terms have n_s words, $x_{s1}, x_{s2}, \cdots, x_{sn_s}$,

$$h_s^{(t)} = \text{sigmoid}(h_s^{(t-1)} H_S + x_s^{(t)} I_S)$$

we use $h_{n_s}^{(t)}$ as the single R^d vector to represent the search terms. The model is similar for product attributes/description, except that matrices H_S and I_S change to H_A and I_A for product attributes, and change to H_D and I_D for product descriptions. We then use h_1, h_2 and $\hat{y}$ defined in subsection 3.3 to capture the nonlinear relevance relationship between product attributes/description and search terms.

3.5 LSTM_Model

Most parts are similar to RNN Model except $h_s^{(t)}$ is defined in another way. The motivation is that some words are more important than others in the product description/attributes or search terms and we want $h_{n_s}^{(t)}$ to capture more important information in the sequence of words. For example, search terms have n_s words, $x_{s1}, x_{s2}, \cdots, x_{sn_s}$,

$$\begin{aligned}
i_s^{(t)} &= \text{sigmoid}(h_s^{(t-1)} W_S^{(i)} + x_s^{(t)} U_S^{(i)}) \\
f_s^{(t)} &= \text{sigmoid}(h_s^{(t-1)} W_S^{(f)} + x_s^{(t)} U_S^{(f)}) \\
o_s^{(t)} &= \text{sigmoid}(h_s^{(t-1)} W_S^{(o)} + x_s^{(t)} U_S^{(o)}) \\
\tilde{c}_s^{(t)} &= \tanh(h_s^{(t-1)} W_S^{(c)} + x_s^{(t)} U_S^{(c)}) \\
c_s^{(t)} &= f_s^{(t)} \circ c_s^{(t-1)} + i_s^{(t)} \circ \tilde{c}_s^{(t)} \\
h_s^{(t)} &= o_s^{(t)} \circ \tanh(c_s^{(t)})
\end{aligned}$$

we use $h_{n_s}^{(t)}$ as the single R^d vector to represent the search terms. The model is similar for product attributes/description. For both RNN_Model and LSTM_Model, in order to feed batch data in tensorflow to train the model, we need the number of words for search terms (or product description/attributes) to be fixed for different pairs. We, therefore, pad search terms (product desciption/attributes) to the maximum length with word '<unknown>'.

3.6 From Regression to Classification Problem

There are three to four human raters give a relevance score of 1, 2 or 3 to each pair of search terms and product, so the average relevance score for a pair can only take a finite number of values. We can then turn this problem into a classification problem: each relevance score is a class, we build a NN model to predict which class the pair belongs to. We change the $\hat{y} = [h_1, h_2]u^T$ to $\hat{y} = \text{softmax}([h_1, h_2]u^T)$. We then take the index i of the maximum element $\hat{y}_i$ in the vector $\hat{y}$ as the predicted class for each pair. We find the corresponding score to the class as the predicted relevance score.

4 Experiments & Results

4.1 Baseline Model

We use the 74067 training examples to train the baseline model. Then we test its performance on the 166694 test examples[2]. The RMSE for training examples is 0.5226, the RMSE for test examples are 0.5245 and 0.5247 for "Public" and "Private" examples respectively.

4.2 GloVe Word Vector

We train the word vectors using GloVe model and use them in the NN_Model_1 model. Now we only train the model over weight parameters W_A, W_D, U and u and we want to see if the performance is affected by how well the word vectors are trained in the GloVe model.

Table 1: Test RMSE v.s. Number of Epoches in GloVe

Number of Epoches	1	2	3	4	5
RMSE on "Public" Examples	0.5473	0.5469	0.5354	**0.5337**	0.5353
RMSE on "Private" Examples	0.5475	0.5473	0.5347	**0.5336**	0.5353

From table 1, the RMSEs increase after epoch 4, which suggest we only need to train the GloVe model for a few epoches and we should treat the pre-trained word vectors as initializers to the neural network models.

[2]The test examples are marked as "Ingored", "Public" and "Private" and we use the latter two for test.

4.3 Performance

We can see although training loss keeps decreasing, but dev loss starts increasing after a few epoches. The model is easily overfitting after training a few epoches. We compare the RMSE of three NN models after we train the model for one epoch when word dimension is 50, see table 2, and compare the RMSE of three NN models after we train the model for two epoches when word dimension is 100, see table 3.

As we can see, NN model has smallest test RMSE when word vectors are 50d. RNN model has smallest test RMSE when word vectors are 100d. The performance of different models are similar on test set. Generally, test RMSE is smaller when word vectors' dimension is smaller, although test RMSEs of RNN model are quite close for 50d and 100d word vectors.

We run the same model as a classification problem. The results see table 4. For RNN and LSTM models, the test RMSEs for the classification problem is higher than the test RMSEs for the regression problem. RNN performs better on the test set for both regression and classification problems. Although intuitively, classification problem should have better performance, because the output is restricted to the possible scores that can appear in the data set. The results are counterintuitive. The possible reason is that the classes are not clearly distinctive (even though the output scores are distinctive) because human rating is a subjective process, so there exists some ambiguity among classes, which makes the model harder to train.

Table 2: Test RMSE for 50d Word Vectors (1 epoch, lr = 0.001, l2 = 0.005)

Model	NN Model	RNN Model	LSTM Model
RMSE on "Public" Examples	**0.5155**	0.5256	0.5200
RMSE on "Private" Examples	**0.5155**	0.5260	0.5201

Table 3: Test RMSE for 100d Word Vectors (2 epoches, lr = 0.001, l2 = 0.005)

Model	NN Model	RNN Model	LSTM Model
RMSE on "Public" Examples	0.5577	**0.5256**	0.5359
RMSE on "Private" Examples	0.5553	**0.5256**	0.5411

Table 4: Test RMSE for 50d Word Vectors (Classification, 5 epoches, lr = 0.01, l2 = 0.001))

Model	RNN Model	LSTM Model
RMSE on "Public" Examples	**0.5694**	0.5853
RMSE on "Private" Examples	**0.5676**	0.5817

5 Problems & Future Work

Based on what we have done so far, there are still lots to improve.

- For all these models, we can consider batch-optimization. The obstacles here are that sentences are of different length and different products have different number of attributes. We want to consider the length of different texts.
- Another improvement we can do is tuning hyper-parameters to achieve better performance.

References

[1] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532-1543).

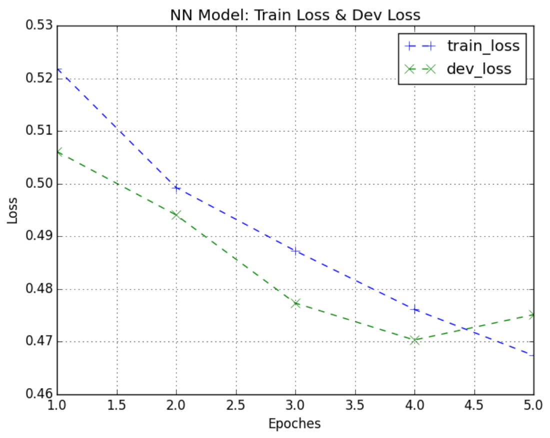

Figure 1: NN loss

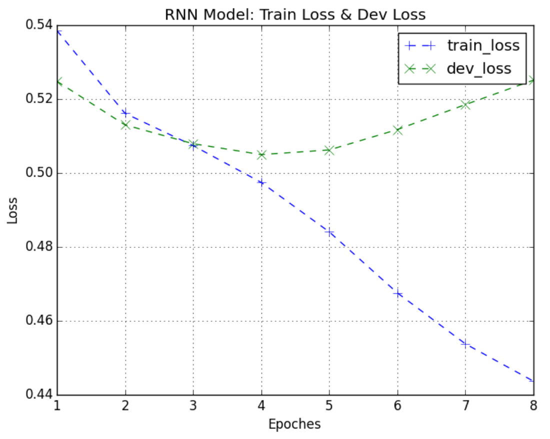

Figure 2: RNN loss