Attention-Based Amazon Rating Prediction

Xiangchen Zhao
University of California, San Diego
xiz643@ucsd.edu

Peiran Liu
University of California, San Diego
pel076@ucsd.edu

Shuyang Cui
University of California, San Diego
s3cui@ucsd.edu

ABSTRACT

In our project, we made several approaches to predict Amazon customers' ratings based on their review, their personal info, and their purchase history record. We have built 3 different models that utilize different features including sentiment analysis, deep learning using the self-attention mechanism, and latent factors. Compared to the baseline model, which just predicts mean, we have found that using linear regression with sentiment analysis on comments gives the highest performance. Moreover, if setting aside the time and computational costs, the attention mechanism also gives promising results.

1. INTRODUCTION

Predicting customers' behavior has always been a popular study in the data science area. An accurate predictive analysis can lead to an effective marketing strategy, which, in turn, can result in better brand growth and more significant revenue. Rating prediction, which in the center of this topic, is the focus of our study. In this report, we aim to predict customers' ratings using three different approaches, the first one is to use latent factors, we think a user purchase history would tell us what the user thinks about the current product. The second one is to use sentiment analysis to calculate the sentiment score of each comment and use that, along with other features, to predict the rating of that product. The sentiment score is calculated using a pretrained network called huggingface, which gives us whether it thinks the comments are positive or negative and the sentiment score. The final approach is to use a multi-head self-attention deep learning model only based on customers' comments. Each comment will be remapped to a 2d word vector, and then feed into the network. We make this problem as a regression problem and we tried to minimize the MSE between the predicted rating and ground truth value. Once we get the result of those three models, we will do a pairwise comparison and compare them with the baseline as well.

2. DATASET

The data set renttherunway_final_data contains 192544 pieces of data that describe consumers' feedback on their purchase of clothes. Our prediction target is the rating of each purchase, so we ignore data without the rating label. Then we randomly sampled 50000 pieces of data from the original data to reduce the training time. We will use the following features in our models: 'fit' - describe if the cloth fits the consumer; 'review_text'- consumers review on this purchase; 'user_id' - consumer's id of the purchase, 'item_id' - product id of this purchase.

The rating is labeled from 1 through 10 and they are discrete numbers. As we can tell from Figure 1. 1, more than half of all purchases are labeled with rating 10. This is an unbalanced data set. Then we want to observe how the feature 'fit' is distributed on this data set. From Figure 1.2, we observe that most people think the product fits them. This observation also explains the observation of rating distribution that most people think product fits lead to a higher rating on the products. In the data set, we have 105508 unique user_id and 5850 unique item_id. In all the following models, we will use 80% as training data and 20% as test data.

Column	DataType	Comments
fit	str	Fit/Large/Small
User_id	int64	Customer id
Item_id	int64	Item id
Weight	str	Customer's weight
Rented for	str	purpose of renting
Review text	str	Comments
Review summary	str	Review summary
Rating	int64	Rating of the product
Category	str	Category of the product
Size	int64	Size of the product
Age	int64	Age of the customer

2.1 Data Pre-processing

The first thing to do with the data is to handle the null values. In our case, we simply drop those records with nulls in them because of the size of the data set. Moreover, considering the size of the entire data set, we down sample the size to 50k records to reduce the computational expense.

In order to pass categorical values into our models, we needed to encode the categories' strings into numbers. We use one technique known as the one-hot encode that expands one categorical feature into several binary columns. This encoding method is sufficient for both the latent factors model and baseline model as they do not take comments as one feature. However, the deep learning model and linear regression require different encoding methods as both of them rely heavily on the comments on their prediction.

For numerical data, we assumed each feature is Gaussian distributed (except for user id and item id) and therefore normalized them into N(0,1) distribution for simplicity of calculation. After cleaning and encoding, the data set had 50k rows and 19 columns.

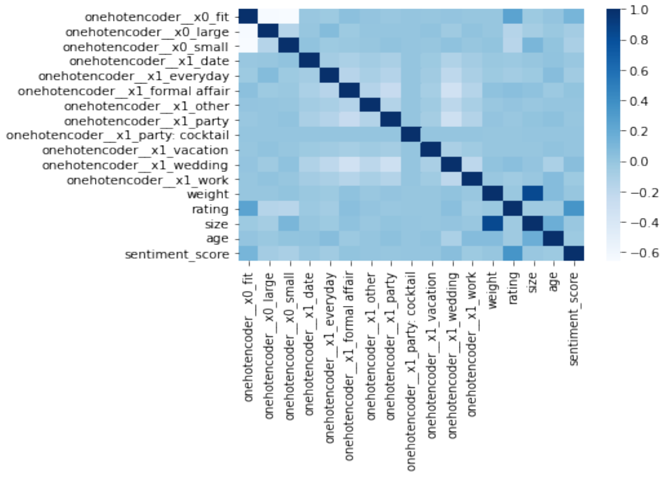


Figure 2: Correlation heatmap

3. EXPLORATORY ANALYSIS

First of all, the rating is labeled from 1 through 10 and they are discrete numbers. From figure 1, we can see the ratings are highly skewed, with over 80 percent of 10 ratings. Thus this is an unbalanced data set and this prior distribution should be considered during prediction. Then we want to observe how the features correlated with ratings in this data set.

In figure 2, We plotted the correlation heat map. and found the feature 'fit' highly correlated with the ratings. This observation also explains the observation of rating distribution that most people think product fits lead to a higher rating on the products. Moreover, In the data set, we have 35345 unique user_id and 5064 unique item_id. There are much more consumers than products in this data set. This might be a benefit for the latent factor model since each item might have more user data and rating data.

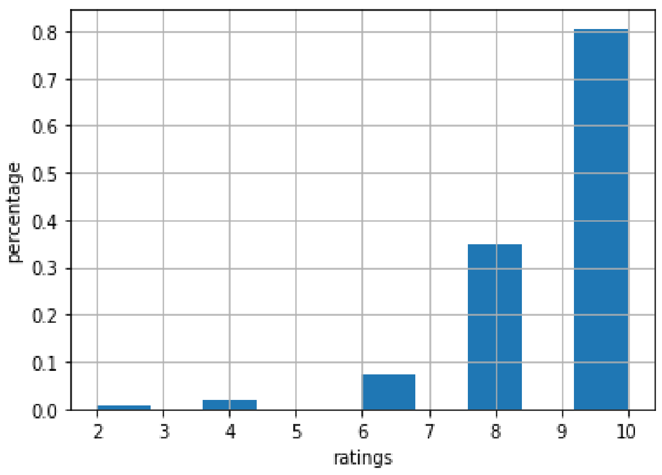


Figure 1: Ratings distribution

Furthermore, there is another feature 'sentiment score' that is calculated from review_summary and appears to have a strong correlation with ratings. This score is a binary feature with 1 meaning positive sentiment and 0 meaning negative sentiment. This score is calculated using a pre-train network called 'hugging-face'. The detailed implementation of that network is beyond the discussion of this report, we only utilize it as one important feature in prediction. We further plot the relationship between the ratio of positive sentiment and the category of rating and found that it almost exists a linear correlation, see figure 3.

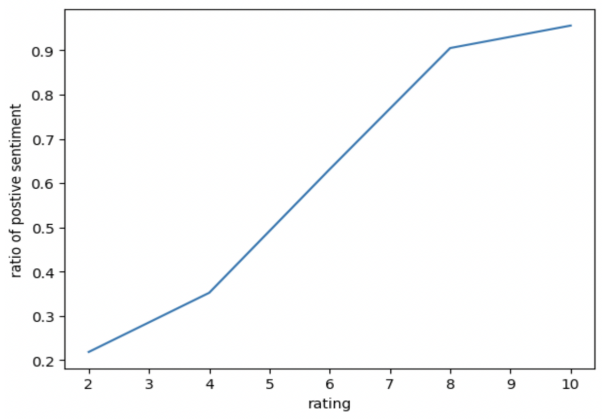


Figure 3: Positive sentiment percentage in each group

4. METHODS

From the above analysis, we make Five different models to predict customers' ratings. Although the ratings are discrete, we still treat this task as a regression test and use the mean square error to evaluate the performance of each model. The first two models just use a single value to predict and therefore will be treated as baseline models. Then we wish to utilize the user's purchase history to make prediction on his next rating, we will train a latent factor model to realize it. Then we want to heavily emphasize the customers' reviews to make predictions since it is most directly correlated with the rating. There are two different approaches, linear regression and deep learning with transformer. Logistic regression not only uses the sentiment score but also uses all other features altogether to make a prediction, while deep learning only uses the review. Here are detailed discussions of each model.

4.1 Baseline 1 - predict 10

$$y_{predict} = 10 \tag{1}$$

As we saw from the rating distribution, more than half of purchases have a rating 10. Thus predicting 10 is a reasonable baseline model. This model only makes a prediction based on the prior distribution of rating and its MSE = 2.9584. The large might be caused by using MSE error which has little tolerance for large errors. By looking at the mean absolute error (MAE), this model has MAE = 0.92, which is significantly lower, yet far from unsatisfying.

4.2 Baseline 2 - predict mean

$$y_{predict} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{2}$$

The simplest model just predicts the average rating in the training set and has MSE = 2.112 and MAE = 1.189. This model ignores all other features and distribution of the data set. It is slightly better than the above baseline in MSE, but still has a long way to go.

4.3 Latent Factors

Product user reviews are factored into user and item feature vectors using latent factor models. The models search for low-rank feature matrices to explain the data using this strategy. This is done by employing the stochastic gradient descent searching method to minimize a penalized least squares loss function. Clustering algorithms could be used as a method to draw meaning from these feature matrices. The items are proposed to be clustered using a heuristic clustering technique based on the models' identification of their hidden features.

In order to build a latent factor model, we assume that each user is associated with a user bias α_i, each item is associated with an item bias β_j, an inner product of the feature vector $u_i v_j$ and rating mean μ, then we can predict item rating j for user i with the following equation.

$$r_{ij} = \mu + \alpha_i + \beta_j + u_i v_j \tag{3}$$

We want to optimize the equation with minimizing the loss function L on the training set.

$$L = \sum_{ij} (r_{ij} - \mu - \alpha_i - \beta j - u_i v j) + \lambda(\|u_i\|_2^2 + \|v_j\|_2^2 + \alpha_i^2 + \beta_j^2) \tag{4}$$

Therefore, we propose gradient descent to optimize the above equation to get our desired parameters. The main concept is as follows: given initial values for model parameters, the algorithm updates the parameters by moving in the opposite direction of the objective function's gradient, which immediately leads to a local minimum. The method will eventually approach the global minima if the function is convex.

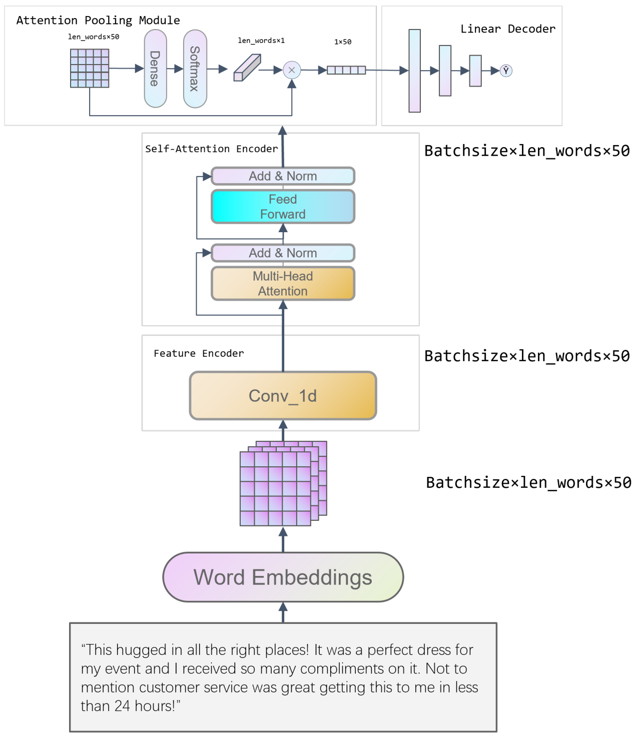
The latent factor model with parameter lambda equal to 0.01 gives the lowest MSE on the test set. The latent factor model with lambda = 0.01 has MSE = 2.108. This model has approximately the same MSE as baseline model 1. This model is not doing anything better by predicting the average rating.

4.4 Linear Regression with Sentiment Score

Now that we want to explore what information customer reviews can bring to make our prediction more accurate. Firstly, we need to convert the reviews into something that is trainable. Therefore we use a sentiment analysis tool called the 'hugging face'. This is a pre-trained sentiment analysis model that takes a string and computes its sentiment score. A positive score means the review is positive. Furthermore, we also bring in all the one-hot encoded features like 'fit', 'category', and 'rented for' as well as numerical features with large correlation to ratings, which include 'age', 'size', and 'weight'. Finally, after training, the linear regression model gives MSE = 1.701. So far it is the best model we have.

4.5 Deep Learning with Attention-Based method

To better exploit the information in the dataset, we also analyze the review text using the attention-based method and design a network. The network takes the word embeddings as input and outputs the rating. The overall architecture is illustrated in the figure below, including Feature Encoder, Self-Attention Encoder, Attention Pooling Module, and Linear Decoder.



"This hugged in all the right places! It was a perfect dress for my event and I received so many compliments on it. Not to mention customer service was great getting this to me in less than 24 hours!"

Figure 4: Overall Architecture

Word Embedding: We first get plain text from the review text and review summary by splitting and removing all symbols and then concrete them. We then padding or cut out the tokens to the fixed length of 50. We use widely used pre-trained 50-dimensional GloVe word vectors [5] for each token, which provides a representation of a sentence of size $R^{len\_words \times dim}$, where len_words=50, dim=50.

Feature Encoder: We adapt 1D convolution as a linear encoder in order to extract simple patterns in each word vector and encode them to a high-level feature space. The dimension of the word vectors we use is relatively small, so we keep the output dimension the same as the input. The kernel size is set to 1 in order to keep the temporal information.

Self-Attention Encoder: We first adapt a multi-head self-attention layer with a feed-forward network proposed in [6] and we cancel non-linear layer to model the correlation or representation vectors in the word sequence. The core concept can be described below. Given three vectors of Key(K), Query(Q), and Value(V), which are generated by the linear transformation of the input in Eq. (5-7). Where X is the input, $X \in R^{len\_words \times dim}$, $W_Q, W_K, W_V \in R^{dim \times dim_K}$

$$Q = XW_Q \tag{5}$$

$$K = XW_K \tag{6}$$

$$V = XW_V \tag{7}$$

Then, the attention weights are calculated by Eq. (8), and the weighted sum of values is generated by Eq. (9).

$$\alpha = softmax(\frac{Q \times K^T}{\sqrt{dim_K}}) \tag{8}$$

$$Y = \alpha V \tag{9}$$

The feed-forward network maps the outputs from the multi-head self-attention module to a larger feature space and then maps it to the original space. We also add residual connections to the multi-head self-attention module and the feed-forward network, which enables the network to conduct residual learning.

Attention Pooling Module: The output of the Self-Attention Encoder is still a sequence of vectors, it is necessary to transform it into a single vector. Therefore, we calculated the global weight for each representation vector in the sequence and add them up, the method can be described as follow:

$$A = softmax(XW^T) \tag{10}$$

$$L = A^T X \tag{11}$$

where $A \in R^{len\_words \times 1}$, $L \in R^{1 \times dim_K}$ is the output of the Attention Pooling Module and $W \in R^{1 \times dim_K}$, is the trainable parameter matrix.

Linear Decoder: We use the fully connected neural network to map the hidden representation from the Attention Pooling Module to the target value space which is a single value of the rating. And the loss between the predicted valued and the true value is calculated by the MSE function.

The above deep learning method achieves The testing result of MSE=1.5638. And we get MSE=1.4767 when we replace our Self-Attention Encoder with a more complete version of the Transformer Encoder. We achieve the best result using a deep learning model designed by us because the model can make full use of review texts in the dataset which contain abundant information relates to ratings such as users' emotions and attitudes towards the item. The hyper-parameters we use are shown below.

Hyper-parameter	Value
Batchsize	32
Optimizer	Adam
Loss	MSE
Learning-rate	0.001
dim$_K$	50
num_heads	2
Dropout in Self-Attention Module	0.1
Dropout in other Module	0.5
dimention in Feed-Foward Network	100

5. CONCLUSIONS AND FUTURE WORKS

The following graph will show the performance of all the models, we can see the linear regression with sentiment score and the deep learning approach largely over-performed the baseline models and the latent factor models. This is reasonable because those two models utilized the 'reviews' feature, which largely reflects the customers' attitude after buying the product. While the latent factors model does not perform as well as those two, it only depends on customers' previous purchase history and does not require extra information about this current order, so it is more flexible and requires less information to predict. Finally, about the deep learning model, we only let it train no more than 20 epochs to save time and resources, and the dimension of the word vector is only 50, which is less than the commonly used 300-dimensional GloVe word vectors. Therefore, there still has the potential to make a more accurate prediction.

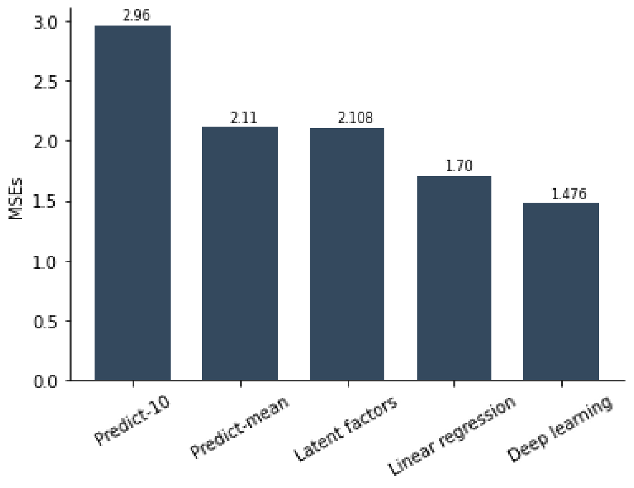


Figure 5: Model Performances

6. ACKNOWLEDGMENT

We would like to acknowledge the contribution of the PyTorch package provider for providing flexible-to-use deep learning model-building tools. We also want to thank our professor Pamela Cosman for providing the structure and algorithm of the latent factors model.