Building a Recommender System

Ananya Bhattacharjee
Department of Computer Science
University of Toronto
Kaggle Team ID: 5896500
ananya@cs.toronto.edu

1 Introduction

Recommender systems (Jannach et al. (2010)) have become an integral part of all forms of online business platforms. Recommender systems analyze the past behavior of users and their interaction with different products, and try to predict which product should be recommended to an individual user based on the analysis. An important part of the recommender systems is rating prediction - predicting how a user will rate a product.

2 Assignment Task

In this assignment, we were asked to construct a recommender system which would predict ratings if a user-item pair is given. We were provided with a training file named "train.json", which contains 200,000 lines of review data. A typical line in the file provides information about how a particular user rated a specific product (in a scale of 1.0 to 5.0) along with other metadata (the actual review text, summary of the review provided, date, etc.). Our task was to use this file for training, and then predict ratings for all 10,000 reviews in "test.json" file, which is similar to the aforementioned file, but does not contain the rating.

3 Feature Selection

In order to build the model, I spent a good amount of time looking at the training data to get familiar and decide on features I should use. I initially tried to implement a variant of matrix factorization algorithm because of my prior familiarity with the algorithm. Through careful inspection, I noticed that the numbers of distinct users and items are quite high (more than 15,000 in both cases) and the "baseline.py" file gave me the idea that the test data may contain users and products not in the train data. However, after a weak submission and some more inspection into the data, I realized that the review text and the summary provide useful information about the ratings (e.g., some of the summary texts are "Five stars", meaning the actual rating provided is 5.0). This inspired me to look into the sentiment of the reviews. This time I used a library named **TextBlob** (https://textblob.readthedocs.io/en/dev/) to analyze the sentiment of the review texts and summary and then predict ratings based those. However, I found this library's performance underwhelming, and even inaccurate in many cases. I built a matrix containing three columns - one column containing sentiment of the review texts, one column for the sentiment of the summary, and the last one for the actual rating. I fed this matrix into an autoencoder (Goodfellow et al. (2016)) based model, but I could only reach a mean squared error (mse) of 0.8. I view this as an unsuccessful attempt too.

Finally, I tried implementing a multi-class classifier, where I assumed the ratings as discrete values (previously I assumed ratings as continuous values). I again used summary and review texts as fea-

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Table 1: Information about an example review

Summary	reviewText	Overall
Three Stars	good	3.0

tures, but I built a deep learning based model with layers of **Convolutional neural network** (CNN) (O'Shea and Nash (2015)) and **Long-Short Term Memory** (LSTM) (Hochreiter and Schmidhuber (1997)). The whole process followed for developing this model was inspired from two sources - a repository on multiclass text classification (https://github.com/tensorflow/examples/blob/master/community/en/text_classification_solution.ipynb) and a tutorial on basic text classification using TensorFlow (https://www.tensorflow.org/tutorials/keras/text_classification). I could beat the strong deadline with this model (0.52880 mean squared error).

4 Data Preprocessing

I only used the **reviewText** and **summary** features to analyze the data. Following the sources mentioned in previous section, I created a folder named "train2" and 5 sub-folders named 1, 2, 3, 4, and 5. Each sub-folder contained the summary and review texts of that rating only (i.e., sub-folder 2 contained only those reviews which had an overall rating of 2.0). In a particular sub-folder, each of the files were named according to its line number in the training data, and contains the summary and review text of that review. For example, the review in line number 13 of "train.json" file contains the information shown in Table 1. Consequently, this review is stored in the "3" sub-folder in a file named "13.txt". The content of the file will be *Three Stars. good*. Each review statement is stored following this way in one of the five sub-folders in "train2" folder.

I used the 80:20 split to divide the divide the training data into training set and test set. In order to prepare the dataset for training, I removed the punctuation symbols and HTML tags from the training set using the function *custom standardization()*. I also split the sentences into words for tokenization. Then I converted the resulting tokens into numbers for using them as inputs to neural network. For this process, known as **vectorization**, I set the sequence length to 500.

5 Model Architecture

Figure 1 illustrates the architecture of my proposed sequential model. Layer 1 is an embedding layer, which takes encoded texts (summary and review text merged in a single text for each review) as inputs and for each word index finds the embedding vector. This output of this layer, a three dimensional array (dimension = (batch size = 32, sequence length = 500, embedding = 8)) is fed to a convolution layer, although I applied a dropout rate of 10% to the output of Layer 1 first. In the 1D convolution layer, the parameter, **filters** were set to 512. I used convolution layer to go through embeddings for multiple words. When this layer is applied on different combinations of similar words, it is likely that the output will be similar too. Layer 4, a Max Pooling layer, is added for increasing generalizability. This layer does not involve learning of new parameters. An extra dropout layer was added to avoid overfitting as well.

Layer 6 consists of a Bidirectional LSTM layer. Many of our review texts are long, so using RNN might struggle with vanishing gradient, eventually giving less importance to certain words. LSTM layers can solve this problem and learn long term dependencies. Bidirectional LSTM uses both the original sequence and the reversed copy of the original sequence for training. Some previous works have been successful in text classification by using using bidirectional LSTM layers, as instead of treating the text sequence linearly, it considers the whole context (Zhou et al. (2016); Nowak et al. (2017)).

Layer 7 averages over the sequence dimension and gives a fixed length output vector. This is useful for dealing with texts of various lengths. A dropout layer was again added after that. The final layer is a dense layer of 5 nodes, as we have five output classes.

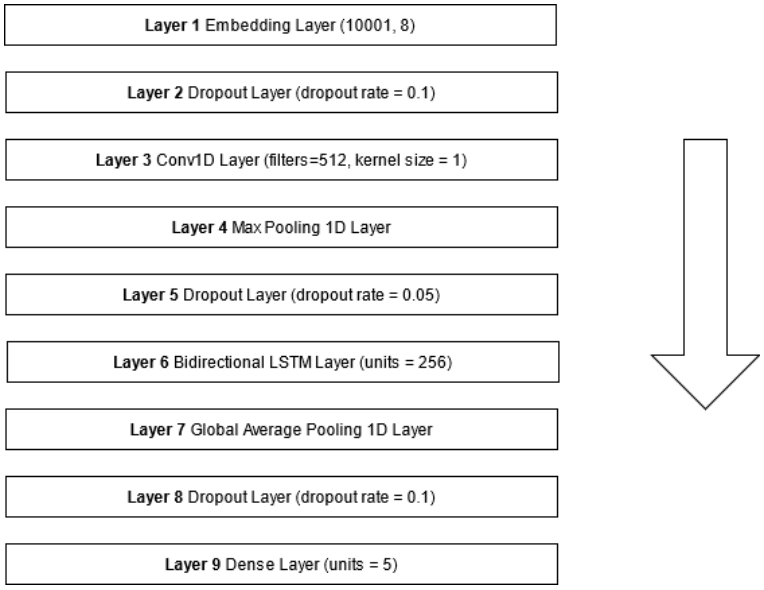

Figure 1: Architecture of the proposed model. The arrow at the right indicates that the layers appear sequentially from top to bottom.

6 Performance Metric

The performance of the model is supposed to be measured with mean squared error (MSE). However, as loss function for training, I used **Sparse Categorical Cross Entropy** because of its widespread use in classification models (Covington et al. (2016)). Then I measured the performance in validation data using accuracy and submitted the model that was the most accurate. The motivation behind this choice was my use of discrete values as output. As I achieved acceptable performance using these metrics, I did not attempt using MSE as a performance metric on validation data.

7 Results and Discussion

My proposed model has an MSE of 0.52880 in the seen portion of the test data, which is significantly less than the strong baseline (0.64545). I trained my model using different number of epochs and observed their performance on validation data. Figure 2 illustrates the performance of my model after different number of epochs. It is evident that the accuracy drops after 8 epochs, so I used the model trained with 8 epochs on test dataset.

I gave an overview of the methods I tried in Section 3. With Matrix Factorization algorithms, I got an MSE of 7.7 on the public leaderboard. However, I believe that method had some flaws, and I later turned to deep learning based methods. With autoencoder and TextBlob, I could beat the weak baseline (0.8 MSE). If the TextBlob library were better in detecting emotions, I think that the MSE might go a bit lower.

In my attempts to work on the model I finally built, I first use the same model used in the aforementioned github repository and only used summary as Text. I made slight improvement (0.77 MSE). Then I merged summary and review text and added CNN and LSTM layers as well, which eventually resulted in the final model.

Although use of matrix factorization algorithm is prevalent for recommender systems, I think for this particular dataset, I believe that approach would not be of much help because the very few number of representations of distinct users and products in the training dataset. Focusing on the sentiment of the review text and summary is much more useful, as some review texts or summary explicitly provide the information about the rating. I applied a basic text classification approach, but using sophisticated pre-trained model like BERT (Tenney et al. (2019)) may improve the performance even further.

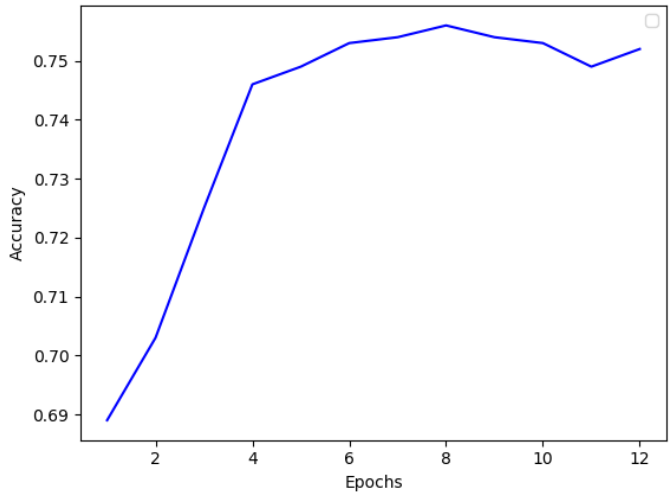

Figure 2: Model Accuracy in Validation data for different number of epochs

8 Conclusion

I used the summary and review text features to predict the ratings for a given user-item pair. Inspired by basic text classification models, I developed a model with CNN and LSTM layers to correctly associate the texts with the rating provided. My model could beat the strong baseline in the public leader-board, and I believe it would continue to do so in the private leader-board.

References

P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

J. Nowak, A. Taspinar, and R. Scherer. Lstm recurrent neural networks for short text and sentiment classification. In *International Conference on Artificial Intelligence and Soft Computing*, pages 553–562. Springer, 2017.

K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

I. Tenney, D. Das, and E. Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*, 2016.