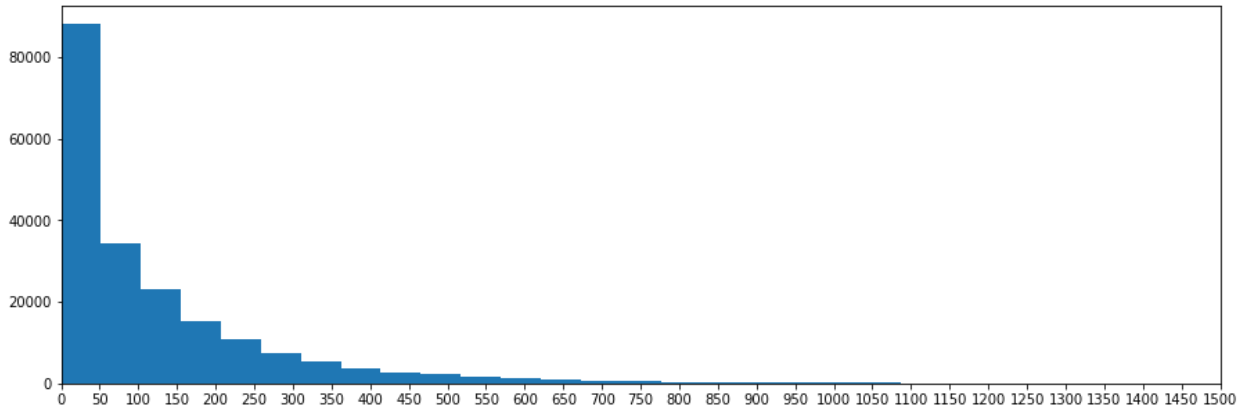
JUNXI XU (TEAMID :5899985)
12/6/20

Project Report - Amazon Music Rating Prediction (Kaggle Competition)

The datasets given were formatted in JSON type, which consists of key-value pairs using a dictionary. This is not convenient for data preprocessing and model implementation. Therefore, first of all, I converted the JSON format to CSV format using my local Jupyter Notebook. Then I uploaded the CSV file to Google Colab to complete the rest of the project.

The training set has 11 columns and the “overall” is the one we use as labels. Since we are expected to do some Natural Language Processing to build a recommender system model, I took the “reviewText” column as feature. Another column containing text is the “summary” column. I planned to simply use the “reviewText” to build features and if it is not enough (eg. The reviews are too short), then I would consider using the summary as well. The following plot shows the histogram for the review length.



As shown, about 80% of reviews have a length of less than 200, and about 60% of reviews have a length of less than 100. Based on this, I think these reviews could contain enough information for prediction. Then I found out how many NAN values for the “reviewText” and “summary” columns (just in case I need to use the summary). The results are shown below.

overall	0	reviewTime	0
reviewTime	0	reviewerID	0
reviewerID	0	reviewText	1
reviewText	36	summary	3
summary	37	unixReviewTime	0
unixReviewTime	0	category	0
category	0	price	0
price	0	itemID	0
itemID	0	reviewHash	0
reviewHash	0	image	9977
image	199537		

Since the NAN values will raise a problem when generating features, I simply replace these NANs with a single word “Ignore” (The records in the test set cannot be dropped since we have to predict ratings for all the 10000 records).

Additionally, I analyzed the proportion of each rating in the training set and found that about 85% of records have 4-5 ratings which means the overall feedback is pretty good for this dataset and we can guess most of the predictions should have high ratings.

The model I used is a pre-trained Bidirectional Encoder Representations from Transformers (BERT) base. There is a BERT large model, but it is more computationally intensive. With the base model, there are 12 stacked encoders, 768 hidden units, and 12 heads. The reason I chose this model is that it is less computationally intensive than BERT large but more complex than traditional LSTM since BERT utilizes the transformer architecture. We use the transformer blocks (only the encoder part of a transformer architecture). This is very helpful since it takes into account self-attention in both directions, meaning that we consider the relationship between a token and all the other tokens within the same sentence. This is feasible since BERT is used for classification as well as next sentence prediction, instead of next word prediction. We are analyzing at the sentence level. In this project, we are predicting the ratings instead of the words within the review and therefore, BERT is feasible and useful. Also, there are 3 embeddings in this model: Token embeddings, Sentence Embedding, and Positional Embedding. These take into account the position of each word and separate different sentences using the [SEP] token and the sentence embedding. Another reason I want to use this model is that I have completed other NLP projects using LSTM and this time I want to try something new. And obviously, BERT is a good alternative.

Since our task is to predict the label, and the fine-tuning procedure has added a linear layer with Softmax activation function to the [CLS] token, which is the first token at the beginning of a sentence, we can complete a multi-class classification. If I use this model for classification, the results will only be an integer (1 -5). This will affect the performance since even in real life, sometimes it is difficult to distinguish between 4 and 5. If we think this product is good, sometimes it is difficult to determine if we should rate it 4 or 5. When we use machine learning to predict the rating if the machine considers this product is good and rates it 5 but actually the user is also satisfied with the product but only rates it 4, which will generate a higher error using MSE. Consequently, I change the task to a regression task to optimize the performance by reducing the output dimension from 5 (5 classes) to 1, which results in floats instead of integers. Another thing is scalability and overfitting. I first trained the model in 5 epochs. It took long and the MSE was reduced to <0.10 in the training set. However, the test performance was much higher (0.38). Definitely, this is overfitting. I found that the loss has been converged at epoch 3 with a decent MSE. So I tried to change the number of epochs from 5 to 3 and got better results.

Other models that I considered are the general Recommender System models such as Probabilistic Matrix Factorization (PMF) and Collaborative Filtering (User-user and Item-item), as well as deep learning model bidirectional LSTM (with attention). For the

general Recommender System models, they only consider the ratings between users and items. Although the PMF takes into account the latent variables, the matrix size is huge due to a large number of users and items, which could potentially result in the computational problem. These methods only take into account the ratings and do not consider other useful information such as user reviews or summaries. Therefore, I used deep learning models. The bidirectional LSTM with attention is feasible but it is not as good as the transformer architecture since the latter allows the model to jointly attend to information from different representation subspace at different positions and the position embedding captures the position of words in a sentence. The pre-trained model is well structured in terms of the transformer blocks, hidden units, etc.

Overall, the traditional Recommender System Collaborative Filtering methods can predict the ratings very fast, but they completely neglect the useful review information. And PMF might work better but it also only considers the ratings and takes much longer to train compared with Collaborative Filtering. The LSTM is a good alternative, and it is less compute-intensive than BERT. Although it is faster than BERT, we need to consider the different hyperparameters such as the number of units. BERT then becomes a better choice. Although it takes much longer to train, the performance looks good. After going through the transformer blocks, the token [CLS] strongly represents all the useful pieces of information in a sentence which makes a good prediction.

During the training process, the MSE for the last 500 iterations (each iteration has 64 samples) in the last epoch is shown below.

```
Iteration 2700 of epoch 3 complete. Loss : 0.16851893067359924
Saving at ./drive/My Drive/CSC2515/Models/Part1_Model_1.1
Iteration:  2800
Iteration 2800 of epoch 3 complete. Loss : 0.11518970876932144
Saving at ./drive/My Drive/CSC2515/Models/Part1_Model_1.1
Iteration:  2900
Iteration 2900 of epoch 3 complete. Loss : 0.09614542126655579
Saving at ./drive/My Drive/CSC2515/Models/Part1_Model_1.1
Iteration:  3000
Iteration 3000 of epoch 3 complete. Loss : 0.22527149319648743
Saving at ./drive/My Drive/CSC2515/Models/Part1_Model_1.1
Iteration:  3100
Iteration 3100 of epoch 3 complete. Loss : 0.18151114881038666
Saving at ./drive/My Drive/CSC2515/Models/Part1_Model_1.1
```

Finally, I got an MSE of about 0.34 with 50% test data (which is shown on the Public Leaderboard), but the overall performance has to consider the other 50% of the test data as well. During the training process, I recorded the MSE during every 100 batches in each epoch and observed how it was oscillating at the beginning and starting to converge as the training process continues. In the first epoch, the MSE was oscillating between 0.2 and 0.6. In the second epoch, it seems that the loss found a god path to converge with less fluctuation. Then during the last epoch, the MSE changed between 0.10 and 0.25.

Conclusively, the BERT base model works well on the Recommender System to predict ratings when we use regression for the last layer and set the appropriate

hyperparameters. Compared with other alternatives, this BERT base model significantly improves performance. And this makes sense since as mentioned above, the distinct architecture of BERT has many characteristics which take into account different aspect of a sentence and a word. During the training process, we can see that the BERT model has a lower loss. This result is meaningful and shows how strong the bidirectionality and multi-head attention are. On the other hand, it demonstrates that BERT can do well on classification/regression and also sentence prediction.

I did not try other feature representations such as N-gram, TF-IDF, Bag of Words, etc. But depending on what models we use; we can use different feature representations. For example, I did an NLP project using normal ML models and feature representations such as N-gram and TF-IDF work well. In the case of BERT, the above representations are not appropriate since we use the pre-trained BERT model, and the sentences should be processed with certain embeddings. For example, we have to represent each token with its id from the embedding table. Also, we have to separate each sentence and insert [CLS] and [SEP] tokens to indicate the beginning and the end of a sentence. These are necessary for the transformer to process the input data in BERT. The model has 110 million parameters in total. During fine-tuning, all the pre-trained layers with the specific parameters are trained simultaneously, and all the parameters are tuned with the same learning rate. These parameters are very important when we transform the word vectors in the Transformer blocks (eg, the multi-head attention part). The success of this model is not accidental. For this specific task, considering only the ratings is not enough since we miss lots of information for the review text and therefore, the traditional methods such as collaborative filtering and PMF cannot work well. Also, the review words and sentences contain lots of important information and significant relationship. This indicates that BERT is appropriate for this task and it is more suitable than LSTM or other general ML algorithms.

Reference:

https://albertauyeung.github.io/2020/06/19/bert-tokenization.html

http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

https://medium.com/dissecting-bert/dissecting-bert-part2-335ff2ed9c73#:~:text=As%20such%2C%20BERT%20base%20has,%3D1024%2C%20A%3D16

https://medium.com/analytics-vidhya/fine-tuning-bert-for-amazon-food-reviews-32e474de0e51

https://towardsdatascience.com/bert-to-the-rescue-17671379687f

https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0