

Recommender System Report

Kaggle team ID: Zi Han Zhao

Student ID: zhaozih3

1 Describe how you processed your data and what features you used. Your exploratory analysis here should motivate the model you use in the next section.

The data processing method is based on IMDB dataset loading from TensorFlow package [1]. The project is a text classification program. Various review strings are related to the rating stars. I pick the most used words for each review sentence and encode them as the input dataset of my model. The training/validation dataset processing is as following: After setting random seed, the raw dataset from `train.json` is shuffled to avoid the correlation between neighbor data when collecting from Amazon. After shuffled, the raw dataset is split into training data and validation data. The ratio between training data and validation data is 8 : 2. The reason of choose the ratio is that let model be trained by enough training data to check the model performance. All of dataset will be regraded as training data after finding the best performance under some epoch. In this way, not only the model is well-trained based on such all of dataset but also overfitting is prevented. I focus on only `reviewText`, `summary` and `overall` entries from the dataset because `reviewText` and `summary` features are the sentences and words to describe the items, which are the closest to the label `overall`. Other entries about `reviewerID`, `itemID`, etc are not directly related to the label and my choosing of model is based on text classification models. first the number of each word is counted and sorted through `reviewText` and `summary` entries in `train.json`. The label is `overall`. Through testing, top 20000 most-frequent words in dataset are stored in a word list. The words after 20000 ranking are ignored since their frequency helps nothing when building good model. The index of each word in word list is its rank position. Second, looking into each single data, if a word in the data is found in word list, I encode the word to a number representing the index of it in word list. E.g, if "`the`" is found in word list and it is in index 0 of word list, it means the word is the 1st most frequent in training data and index 0 is the encoding output of "`the`". The words not in the word list are abandoned. Finally, in order to make each data width the same, zeros are padded at the end of the index sequence. The data sequence length is set as 500. Any sequence over 500 is forced to cut aligned. Through testing, 90% of data sequence length is within 500 and 400 in average. 500 efficiently contains top used words and avoid the negative impact of

too many padded zeros as well. Therefore, the input training and validation dataset is the sequences of indexes with labels. Next, a text classification model should be selected by experiments.

2 Describe your model. Explain and justify your decision to use the model you proposed. How will you optimize it? Did you run into any issues due to scalability, overfitting, etc.? What other models did you consider for comparison? What were your unsuccessful attempts along the way? What are the strengths and weaknesses of the different models being compared?

My main layer is Gated Recurrent Unit (GRU) neural network model. GRU is a improved version of Long Short-Term Memory (LSTM) [2]. GRU and LSTM are both insensitive to gap length. Compared with naive RNN, GRU has cell gate structure to remember the further past information [3].

The reference of my GRU design is TensorFlow RNN Text binary classification for IMDB review [4]. Similar with LSTM model, GRU model is capable of long-sequence prediction. Looking into raw dataset, the length of the review strings is usually over 200. RNN cannot handle and remember long sequence data. GRU and LSTM will result in lower loss and higher accuracy. Compared with LSTM, the number of parameters of GRU is reduced so that the occurrence of overfit reduces and the speed of training increases.

The layers of my base model are input layer, GRU layer and Dense layer. The output is multi-class corresponding to **overall** five rating stars. In order to avoid overfitting situation, I add Dropout layer ahead of Dense output layer. In order to shorten input data size, embedding layer replaces with simple input layer to encode data into multi dimension [8]. The similarity among sentences or words stands out. Through experiments, it is found that 32 encoded dimensions acquires the lowest validation loss. Therefore, I choose the output dimension of embedding layer is 32. The table 1 is the overview of the model.

One alternative model is Transformer TFBertModel. TFBertModel is also a popular Transformer model for next-line prediction [5, 7]. The model provides a effective attention-based neural language translation. The input dataset is the string sequence of combining **reviewText** and **summary** entries. The label is **overall**. The sequences are encoded into **input_ids** list. The layers are Input Layer, TFBertMainLayer, Dropout layer, Dense output layer. The output is also multi-class.

TFBertModel makes use of Transformer model structure to learn the relations among words or sub-words. The accuracy of prediction is higher under large enough dataset and iterations [6]. However, the computation resources are heavy and it is time-consuming.

The other alternative model as my design baseline is Turi Create official recommender system from Apple Inc. [9]. It is fast and flexible aiming at IOS, macOS, etc. Turi Create recommender system is a item-item similarity model. Its goal of the model is recommend item IDs to the given user IDs. It also predict the rating given item ID

and user ID. Different from the models above, the input data is `reviewerID`, `itemID` with label `overall`. The recommender model is not released but according to its input data type, it indicates that the model focuses on user-item similarity rather than the review strings. With enough individual reviewer information and item information, It is capable to predict more accurate rating stars. But the number of the dataset from *train.json* for particular persons and items is limited. The number of data for the same reviewer is less than 100. Underfitting might occur without training review strings.

Layer	Output Shape	# of Parameters
embedding	(500, 32)	640000
GRU	(32)	6336
Dropout	(32)	0
Dense	(6)	198

Table 1: Model structure

3 Describe your results and conclusions. How well does your model perform compared to alternatives, and what is the significance of the results? Which feature representations worked well and which do not? What is the interpretation of your model’s parameters? Why did the proposed model succeed why others failed (or if it failed, why did it fail)?

I set a callback function which will be triggered when validation loss raises up after current epoch training. The training steps stops at epoch 4.

As shown in Figure 1, the training loss and accuracy increases and validation loss reaches to minimum point at 3rd epoch. The minimum validation loss is 0.6038 and the corresponding validation accuracy is 0.7595. The total training and validating time is around 1200 secs. As shown in Figure 2, under 2nd epoch, TFBertModel minimum validation loss is 0.5996 and the corresponding validation accuracy is 0.7579. The total training and validating time is around 10800 secs (3 hrs) since the number of parameters are over 1 million.

Turi Create recommender system performs worse. Its minimum validation loss is 0.8964. The training time is within 5 secs. The training process is unpublished so no medium loss can be found. Since it is regression model rather than class classification, we cannot tell the accuracy. The reason of high loss results is no enough data for individual person and items to train. It is underfitting.

In terms of validation loss and accuracy comparison, GRU performs much better than baseline Turi Create recommender system and takes a slight lead to TFBertModel. Compared with TFBertModel, the time of training is less than half hour. It is efficient to apply GRU model in other practical recommender system to save training time.

References

- [1] `tf.keras.datasets.imdb.load_data`
https://www.tensorflow.org/api_docs/python/tf/keras/datasets/imdb/load_data
- [2] Understanding GRU Networks
<https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>
- [3] Long Short-Term Memory (LSTM): Concept
<https://medium.com/\spacefactor\@m{}kangeugine/long-short-term-memory-lstm-concept-cb3283934359#:~:text=LSTM%20is%20well%20suited%20to,and%20other%20sequence%20learning%20methods.&text=The%20structure%20of%20RNN%20is,hidden%20Markov%20model.>
- [4] Text classification with an RNN
https://www.tensorflow.org/tutorials/text/text_classification_rnn
- [5] Multi-Label, Multi-Class Text Classification with BERT, Transformers and Keras
<https://towardsdatascience.com/multi-label-multi-class-text-classification-with-bert-transformer-and-keras-c6355eccb63a>
- [6] BERT Explained: State of the art language model for NLP
<https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>
- [7] What is a Transformer?
<https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>
- [8] Deep Learning #4: Why You Need to Start Using Embedding Layers
<https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12>
- [9] Apple release 'Turi Create' Machine Learning Framework on Github
<https://todobi.com/apple-release-turi-create-machine/>
<https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12>
<https://apple.github.io/turicreate/docs/api/generated/turicreate.recommender.create.html#turicreate.recommender.create>

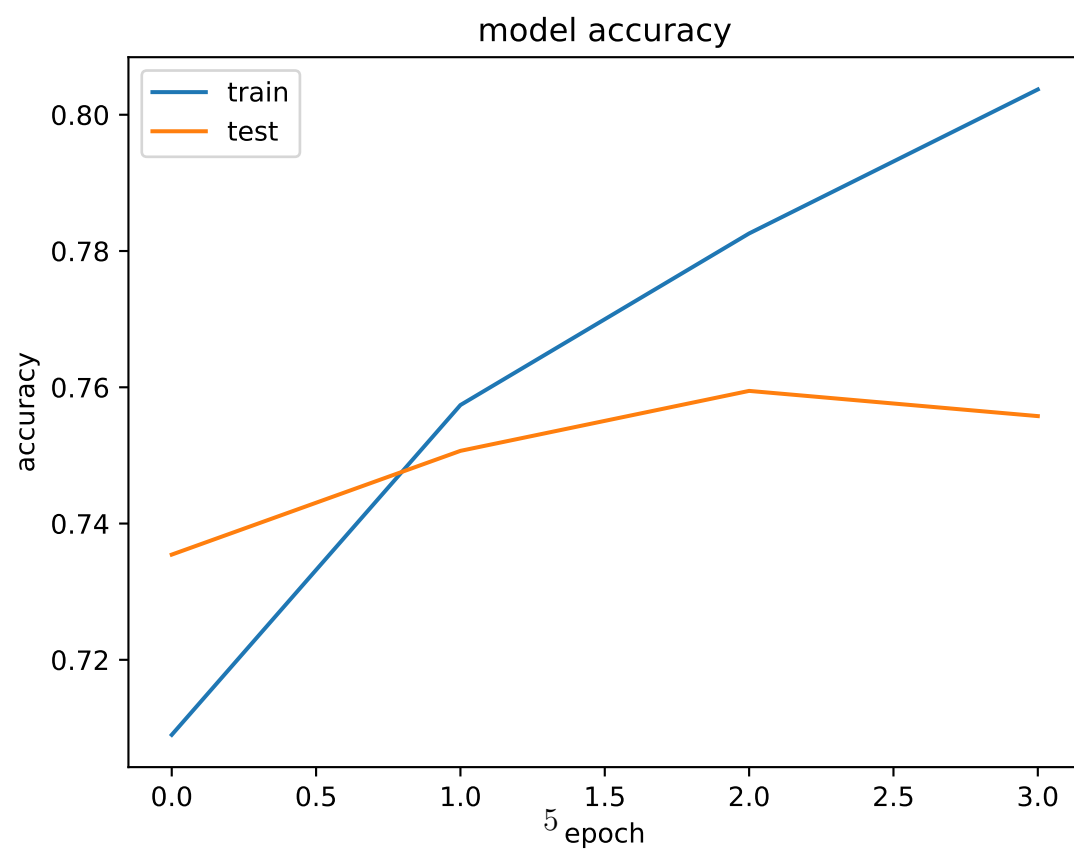
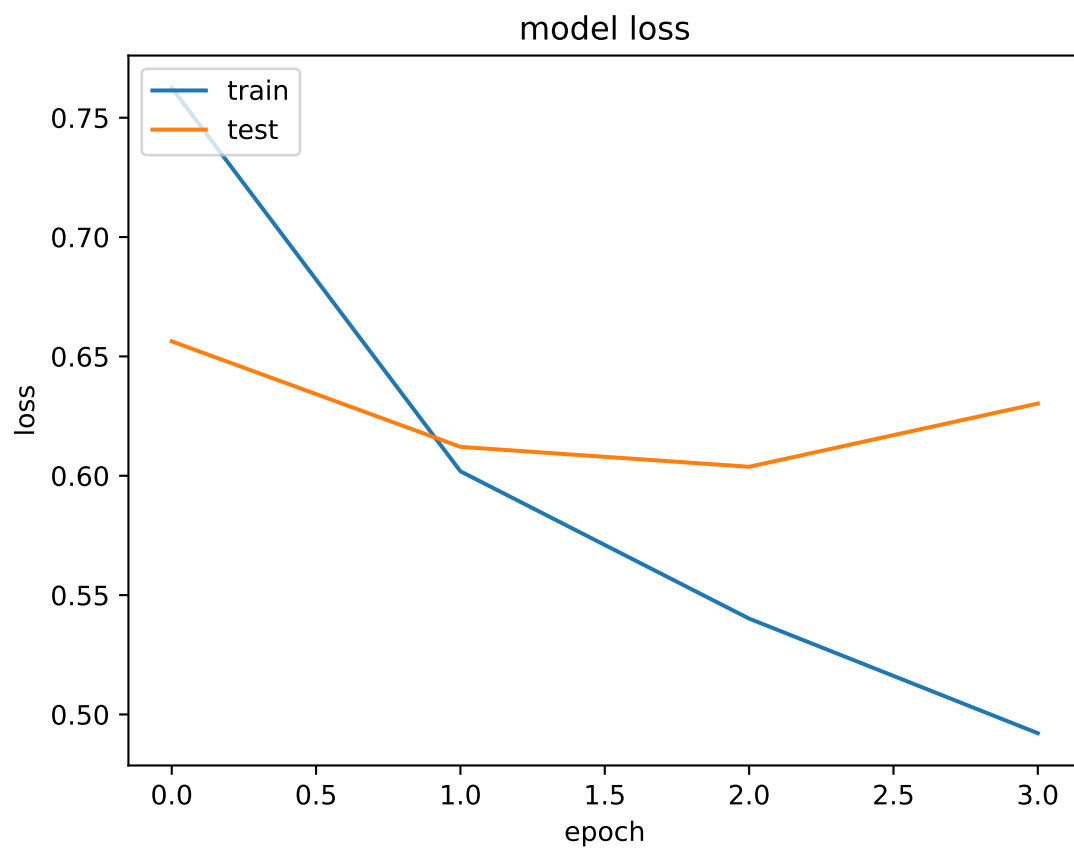


Figure 1: GRU model loss and accuracy

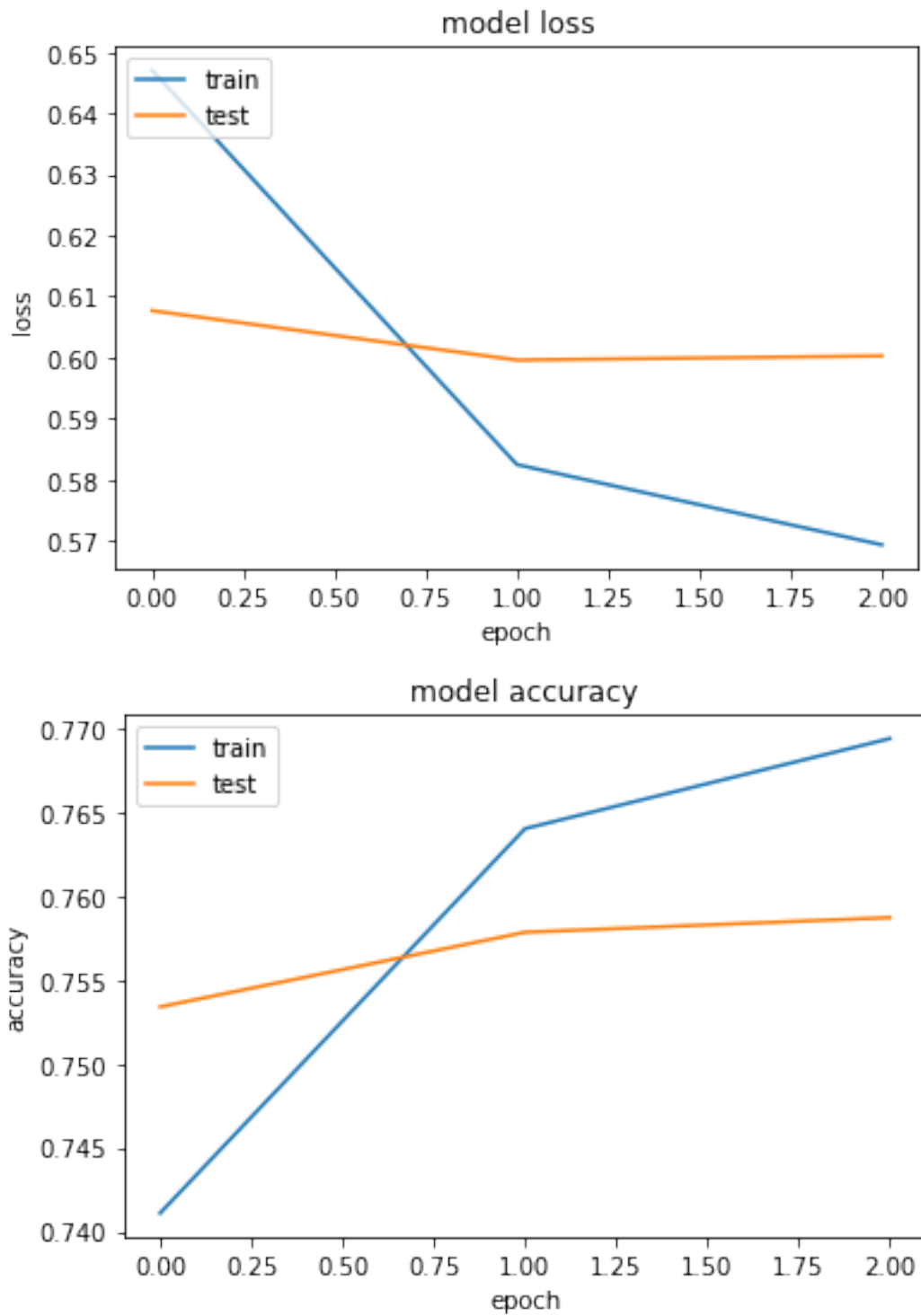


Figure 2: TFBertModel model loss and accuracy