MULTI-CLASS TEXT CLASSIFICATION

Task: Analyse , visualise and build robust models from the given Yelp Dataset to classify customer reviews.

Platform: Jupyter notebook. Tensorflow version 2.6.2

Data: Exploratory Data Analysis

Data consists of customer reviews rated between 1 and 5 stars. Training and test datasets include a total of 650K and 50K samples respectively evenly distributed to each star. I have validated that there is no missing values in the dataset. I've found that the number of words in the reviews varies between 1 and 1052, with an average value of 134. The most common review length is around 60 words. I could not find a correlation between the number of stars and the length of reviews.

Data Preprocessing

Deep Learning algorithms use numbers in their mathematical operations even for NLP tasks. Therefore, text data have to be encoded to numeric values before feeding into DL models. There are several techniques commonly used for this purpose. I preferred to employ word embedding approach, which is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning [1]. Among different word embeddings, I used Glove with the dimension of 50, which means each word is represented by a numeric vector consisting of 50 real-valued numbers.

Before mapping each string-based word in the dataset to its vector correspondence in the Glove, it is advised to perform some data cleaning and preprocessing on the dataset for better match. In this sense, I have removed special characters, punctuation and new line symbols from the text data, and then replaced numbers with # character. Also, I have expanded contractions (e.g. this's → this is), and converted all uppercase characters to lowercase.

Text Vectorization

As mentioned above, the reviews may have different number of words. While feeding data into deep learning algorithms, samples have to be in equal size. So, we have to either pad the short ones or cut them all at a certain place to make them equal length. Note that the longer the text, the longer the training time and bigger memory requirement. For this challenge, I set the sequence length as 60, which means we consider first 60 words of the reviews. If a review is shorter than 60-word, it is padded with empty tokens.

Another parameter we need to set in the word embeddings is the number of tokens, which keeps only a pre-specified number of words in the text. This is helpful because we don't want our model to get a lot of noise by considering words that occur infrequently. I set this parameter to 40K.

Neural Network Architecture

First layer is the input layer that takes tokenized representation of reviews in fixed size.
Second layer is embedding layer that generates word embeddings of the words.
Third and fourth layers are bidirectional LSTM, which process the sequence of the data in both forward and backward directions, and thus extracts patterns and relations from the data.
Fifth layer is dense layer which is used for classification.

Final layer is dense layer used for final classification, where the number of neuron is equal to number of classes.

Model-1 vs. Model-2

Model-1: Model-1 performs classification for 5 categories. Therefore loss function is categorical_crossentropy.
Model-2: Model-2 performs regression that predicts the number of stars between 1 and 5. Therefore loss function is mean_squared_error.

Model-1 and Model-2 are based on identical neural network architecture except the final layer. Since model-1 makes classification, its final layer uses softmax activation function and contains the same number of neurons as the number of classes. On the other hand, model-2 makes regression and therefore there is only one neuron at the final layer with the activation function of linear. The idea behind making regression in model-2 is that there is a ranking relationship between classes, e.g. 3 stars is more closer to 1 star than 5 stars, which is appropriate for regression.

I set the epoch number as 10 during the training process. I've used early stopping callback checking validation loss during training to prevent overfitting.

Results:

According to Loss vs epoch graphs, the training could have continued for a few more epochs because it seems that the validation loss could drop a little bit more.

In Model-1, overall accuracy is around 56%. Confusion matrix provides more insights about the category-based performance. According to the confusion matrix, 1-star has the biggest accuracy whereas 3-star has the lowest accuracy.

In Model-2, overall accuracy is around 53%. According to the confusion matrix, all categories exhibit similar performances in model-2 on contrary to model-1. This is probably due to regression which benefits from order relation between classes.

Improvement Points:

Performance of the models may be improved by doing the followings:
- Increase the number of training epochs until validation loss stops decreasing
- Increase the sequence length (the number of words taken into consideration for each review). This parameter was set to 60 in this setup. I have observed that when we increase this parameter to 100, the accuracy of model-1 reached to %60.
- Optimize the number of tokens, which was set to 40K. The optimal value could be above or below of this number. The optimum value can be found by trial and error.
- Use different neural network architectures. For example, an attention layer can be used as in the transformers. Also, transfer learning using pretrained models could be helpful.

References:

[1] *Jurafsky, Daniel; H. James, Martin (2000). Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River, N.J.: Prentice Hall. ISBN.*