Prediction of Ratings for Yelp Dataset

Sourabh, Zanwar, 391923

Malek, Alhelwany, 391558

Makua, Bernal, 392224

Tobias, Biele, 344413

RWTH AACHEN UNIVERSITY

Report

In this report we will explore the basic steps of predicting the ratings from the review text**,** where we were given a set of reviews written on the Yelp platform. We used the following libraries in our project**: Pandas, nltk, keras, sklearn**

We started with cleaning the text in reviews, we achieved that by changing everything to lower case, then removing the punctuations and stop words, and finally stemming them using Snowball stemmer. All of this data was stored in a list: ***Corpus***. After that we started with building a basic *Naïve Bayes Gaussian Classifier* to predict the ratings from the text in reviews. In order to input the data for it, we had to transform the training data using *CountVectorizer*, we only included 1000 features in it, we tried different values for *max_features* before choosing the one with most accuracy. (500, 800, 1500, 2500). But on getting very low accuracy of just around 38-39% we decided to try using neural network-based architecture.

We then built a LSTM based model, and we had to modify the *training_data* for it, we tokenized the words and used maximum of 100 words per review which was also used as input_length for Embedding layer, other parameters for embedding layer were : *input_dimensions = size of vocabulary, output_dimensions = 100*. After that another layer for LSTM, with drouput of 0.2. In the outpur layer we tried using *sigmoid* activation function but on research we found out that for categorical with 2+ categories it is always better to use *softmax.* For compiling the model we used *cateforical_crossentropy_loss* function, and for this we had to transform out target labels using *to_categorical()* function provided by keras to get 1-of-K type encoding of output labels. The optimizer used was 'adam', and metrics: accuracy. On training the dataset, varying the batch size but it did not have any significant impact. We tried varying various parameters (like changing the number of units LSTM layer, dropout_size) while building the model but they resulted in consuming much more time to train with very little to no improvement in the accuracy or loss.

We finally ended up finalizing a CNN over LSTM architecture. Most of the parameters were same for this and the previous models, the added layers were, *Convolutional1D Layer and MaxPooling Layer* (with pool_size: 4). Though there was not a high difference in accuracy, the time required to train the model was significantly less. That is the reason we decided to choose this model. We only could run 3 epochs in this and the previous LSTM model due to the time constraint. Following are the accuracies that we obtained for the models that we tried:

CNN + LSTM:
```
Epoch 1/3 accuracy: 0.5437 - val_accuracy: 0.6057
Epoch 2/3 accuracy: 0.5924 - val_accuracy: 0.6191
Epoch 3/3 accuracy: 0.6194 - val_accuracy: 0.6172
```
LSTM:
```
Epoch 1/3 accuracy: 0.4401 - val_accuracy: 0.6060
Epoch 2/3 accuracy: 0.5872 - val_accuracy: 0.6162
Epoch 3/3 accuracy: 0.6196 - val_accuracy: 0.6178
```
Classification:
```
Accuracy :  0.38382307692307693
```