Twitter Sentiment Analysis
Kaggle Challenge

40.016 The Analytics Edge

Team 07

Lee Min Shuen (1004244)

Sim Wei Xuan, Samuel (1004657)

Muhammad Hazwan Bin Mohamed Hafiz (1004122)

12th December 2021

1. Methodology

Our initial approach was to consider several base models to determine which type of classification model to focus on. We mainly considered 3 models: a Classification Tree model pruned using the lowest cross-validation error, Random Forest, and Naïve Bayes Classifier. The validation accuracy of each model was 0.827, 0.853, and 0.741 respectively. Our group suspected that the accuracy was not higher due to the sequential nature of textual data, whereby information provided by text is reliant on word order, e.g., "work to live" and "live to work". Hence, we needed a model that could retain such information and decided to consider more advanced models such as neural networks. We identified Recurrent Neural Networks (RNNs) as the appropriate neural network due to their ability to work with sequential data. Since basic RNNs suffer from the long-term dependency problem, where the desired output depends on inputs presented at times far in the past, we specifically implemented a Long Short-Term Memory (LSTM) RNN model for the challenge, as LSTMs were explicitly designed to avoid this long-term dependency problem.

2. Text Preprocessing

Text preprocessing was performed under the following considerations: (i) Lower case: Since our corpus is written in the English language, our model will treat similar words of different capitalization as different terms. (ii) Remove word elongation: Some words might be elongated to emphasize certain emotions due to the informal nature of tweets (E.g. Noooo!). Therefore, we removed any word elongations. (iii) Replacing internet slang: Many tweets contain internet slang (E.g. "Gr8", which is the internet slang for Great). Hence, we replaced the internet slang with their relevant formal English words. (iv) Replacing emoticons: Tweets contain emoticons with sentiment meaning in the form of symbols (E.g. ":-)"). Hence, we replaced them with corresponding expressions like smiley. (v) Remove URLs, email addresses, Html tags: These elements only add noise to the data and do not offer much relevance to sentiment analysis.

3. Long Short-Term Memory (LSTM) Model

We put together a basic LSTM network using TensorFlow with the help of the Keras and Reticulate libraries in R. The architecture is simple: Input – Embedding – LSTM – Dense – Output layers. As per standard machine learning practice, we split our train data into a train and validation set using an 80:20

ratio such that we can use the validation set as an unbiased estimator of our test set accuracy. Here is a non-exhaustive list of models we have tried and our final choice:

1. One layer LSTM with no pre-trained embeddings and two-layer deep neural network.
2. One layer LSTM with one pretrained embedding layer and two-layer deep neural network.
3. Two-layer LSTM with two pretrained embedding layers and a two-layer deep neural network.

4. Model Layers

(i) Input: Takes indices of tokenized words as input and passes them to the embedding layer to be transformed into a word vector representation before passing through the LSTM layer.

(ii) Embedding: Firstly, there was too little provided data to train the word embeddings accurately. For comparison, pre-trained word embeddings created from Stanford's GloVe[1] model are trained on 2 billion tweets to create embeddings for 1.2 million words, whereas we only had a word count of 19974 after pre-processing 30000 tweets. Our word embedding resulted in our initial model overfitting the training data with a training accuracy of 0.96, while validation accuracy was only 0.89. Hence, we decided to utilize pre-trained word embeddings trained on Twitter data to provide a starting point for the embedding weights of the embedding layer. The GloVe pre-trained word embeddings consisted of 1.2 million words, each represented by a 200-dimensional vector. We matched our tokens to the pre-trained embeddings and assigned tokens not found in the embedding matrix to be represented by a zero vector. Eventually, we sourced for additional word embeddings to add more dimensionality to our input data. However, it was important to use embeddings trained on Twitter data for pre-trained word embeddings, given that the model was to be used on tweets. We learnt that the hard way when our models performed worse after using as many pre-trained embeddings as we could add. Eventually, we decided to use just two embedding layers which two embedding matrices provided weights pre-trained on Twitter data using Stanford's GloVe and Facebook's fastText[2] models.

(iii) LSTM: For both LSTM layers, we used the default hyperbolic tangent activation function (tanh) for the final output and the sigmoid function for the gates' recurrent activation function. All these are to

---

[1] Jeffrey Pennington, Richard Socher, & Christopher D. Manning (2014). GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543).

[2] Godin, F. (2019). *Improving and Interpreting Neural Networks for Word-Level Prediction Tasks in Natural Language Processing*. (Doctoral dissertation, Ghent University, Belgium).

introduce non-linearity into our model. The dropout values are 0.4 and 0.5 for the activation and recurrent activation respectively to prevent overfitting. Since we are only concerned with the LSTM cell's final output, we used the default, "FALSE", for the return_sequences argument that the configuration is "many to one". Therefore, the final output of the two LSTM layers is a rank one tensor of length 32 each. We then concatenated both rank one tensors into a single one-dimensional tensor with 64 neurons as the last step.

(iv) Dense and Output: The output from the LSTM layers is a one-dimensional tensor of size 64, which will be passed on to a dense layer with 128 neurons. A rectified linear activation function (ReLU) is used here, which performs better for sparse inputs. The last output layer is a layer with three neurons representing each sentiment class. Therefore, we used a SoftMax activation function at the end that normalizes the results into a probability distribution across three classes whose sum add up to 1. In between each dense layer, we included a dropout of 0.5 to once again account for overfitting.

5. Model Hyperparameters

The choice of the optimizer is Root Mean Squared Propagation (RMSProp), which performs well on sparse datasets like our tweets. We chose a high number of 100 epochs, as we have added an early stopping function that stops the training if our specified validation loss does not improve across 10 epochs. Despite knowing how a larger batch size will result in a faster and better fitting, we observed that a smaller batch size produced a better training and validation accuracy for us. Hence, we chose a smaller batch size of 128. We also played around with various learning rates and settled down on a learning rate of 0.001 that has the best performance. After training the model for the first time, we unfroze the weights of the embedding layer and trained further with a lower learning rate of 0.0001 to better fit the embedding weights to our problem in hopes of a better optimal solution.

6. Description of Results

Our model results with the final set of weights: Training Loss: 0.1815, Training Accuracy: 0.9553, Validation Loss: 0.2458 and Validation Accuracy: 0.9338. To evaluate our model, we had to look at the losses and accuracies over the training, shown in Figure 1.0. By comparing both performance variables between the training and validation sets, we will be able to determine the extent of overfitting or underfitting. As observed from the graphs above, our model's overfitting begins around the 20^{th} to 30^{th}

epoch, where the training accuracy continues to increase while there is no significant improvement in validation accuracy. It is important to note that while the model "sees" the validation set, no training is done on it. Likewise, we also see that the training loss decreases as we further train while the validation loss sees no improvement. Nevertheless, the extent of overfitting is not very large, as we notice that the deviation between both lines for both plots is not significant. However, for our given hyperparameters and model architecture, the sweet spot would be training it till the 20th epoch.

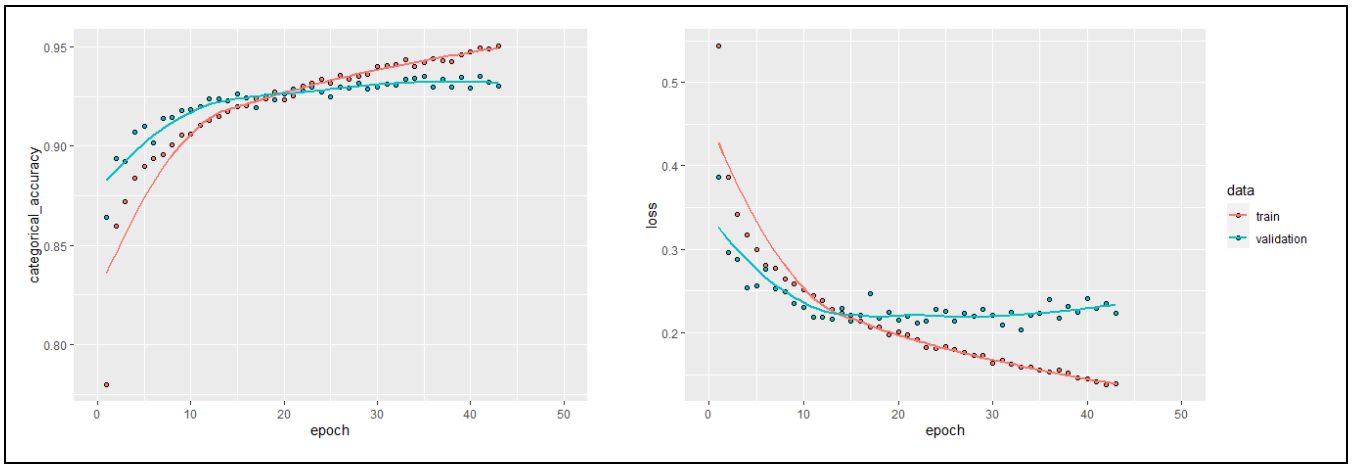

Figure 1.0 Plots of the training and validation, accuracies, and losses over each epoch

7. Limitations

(i) Data augmentation: Ideally, we would have preferred to implement text augmentation techniques such as back translation and synonym replacement to get more training data. However, we were limited by the libraries available in R. Looking through the train set; we realized that some tweets seem to be mislabeled, probably due to the crowdsourcing of data labelling. Therefore, relabeling the trainset could be done; however, we felt it might not be acceptable in the name of competition fairness.

(ii) Alternative Models: Use state-of-the-art transformer-based models such as Bidirectional Encoder Representations (BERT), a pre-trained deep learning model developed by Google AI. However, the BERT model in R[3] inputs pre-processed tweets into their pre-trained and defined layers, which does not benefit our learning. Therefore, we decided to forgo the BERT implementation even though it requires us to follow the step-by-step setup of the correct environment in R and run it.

[3] Abdullayev, T.. (2019). RStudio AI Blog: BERT from R.