Gurjus Singh

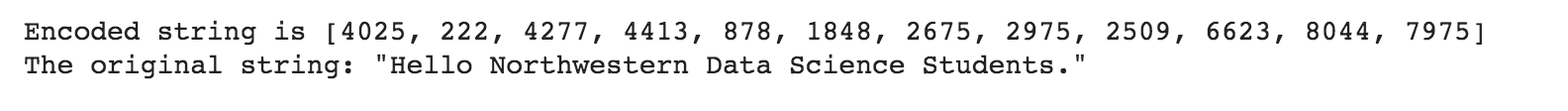
November 8th, 2020

MSDS 422 Practical Machine Learning

Assignment #8 Language Modeling with an RNN

**Data preparation, exploration, visualization**

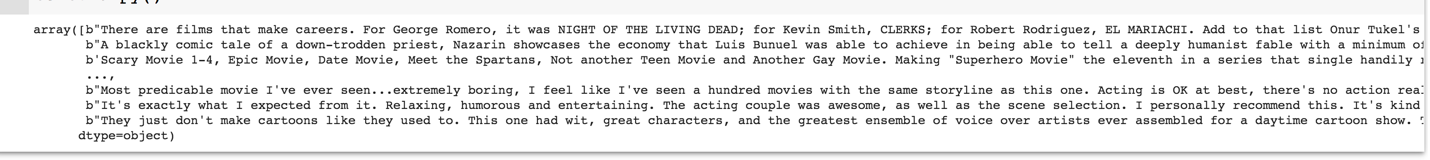
The goal of this Data Analysis was to use two types of Neural Networks specifically Recurrent Neural Networks and Long short-term memory Neural Networks for **Binary Classification** on IMBD Movie Reviews. This is considered a supervised learning method as there was a target variable in the Movies dataset. Before implementing the models, I first had to download the dataset and split it using tf.load function from TensorFlow package. The dataset had its own encoder that was used to encode the text to numbers. I got an idea of how it encodes by showing encoding a sample string shown in output 1-1. I then saw how the encoded string was mapped to each text pattern in output 1-2. Next I wanted to seem of the text and get an idea of the types of text the encoder encodes from the movie reviews dataset. I saw this my looking at the plain text dataset seen in Output 1-3. From the text negative reviews were seen through the words ‘boring’, ‘worst’, ‘bad’ while the positive reviews were dictated by the words of ‘good’, ‘best’, ‘great’ , and ‘entertaining’.

*Output 1-1*

Table

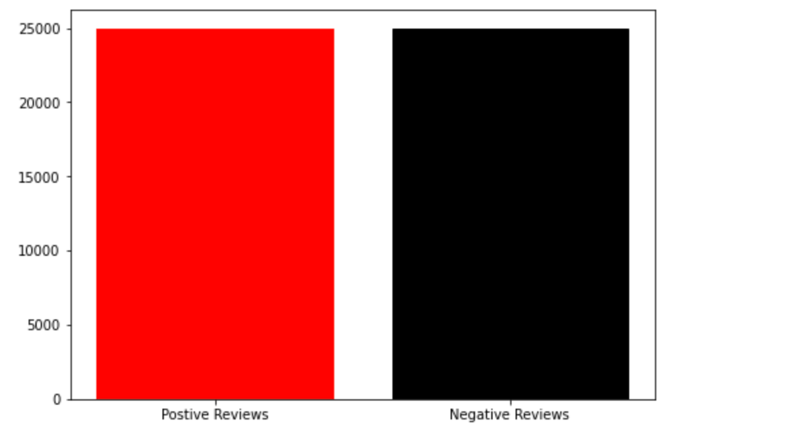
Description automatically generated

*Output 1-2*

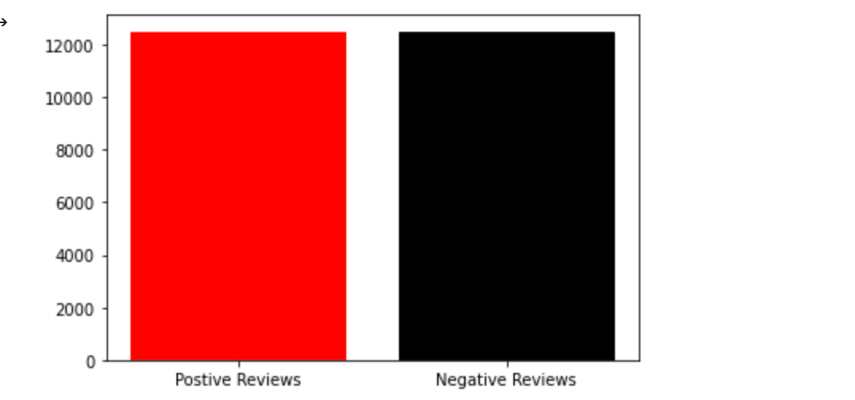
**

*Output 1-3*

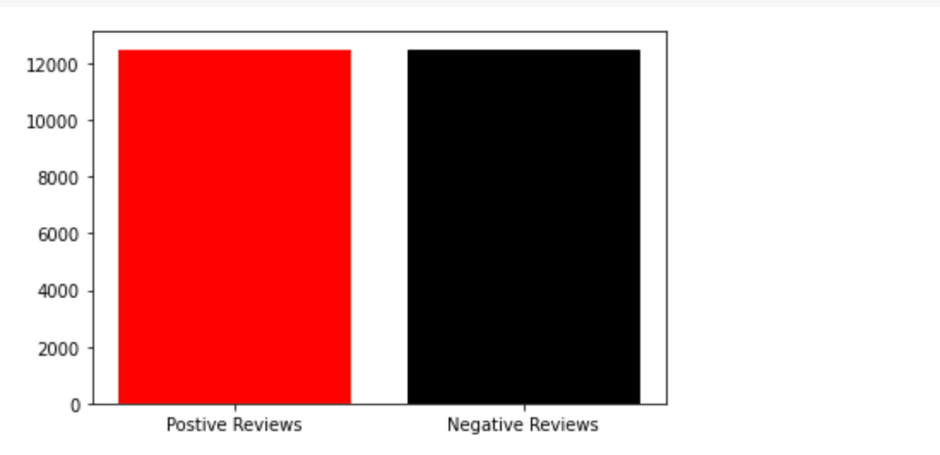
I then tried to create a bar plot to see how many positive and negative reviews there were in both datasets. What I found that there were equal number of negative and positive reviews in both. I also saw there were equal number in total between all the dataset splits shown in bar plots 1-4 through 1-6.



*Shows number of Positive and Negative Reviews in total 1-4*

**

*# of pos and neg reviews in train data 1-5*

**

*# of pos and neg reviews in test data 1-6*

Before creating the Neural Networks for this training, I had to make sure inputs of text had were of the same size, so I had to do padding of 64 words to get shorter words to the same size of the longer texts that included 64 words. The algorithm adds 0s on the end of these shorter encoded texts for padding [1].

**Review research design and modeling methods**

In this analysis I used a type of Deep Learning method which is called Recurrent Neural Networks. This type of Neural Network is used for sequential data [2]. The thing that is special about Recurrent Neurons is that it receives previous data with new data as shown in Figure 1-7 from the Geron book where x(t) represents present data input and y(t-1) represents previous days output [2]. With x(t) and y(t-1) we can create a formula represented by formula 1-8 which is a linear combination of x(t) and y(t-1) , and this creates a new output for a present day’s neuron [2].

In the case of the analysis of NLP used in this analysis it is necessary to have a Bidirectional Layers as they look in both directions in both past and future to predict a target variable. In the case of text, it needs to look ahead to figure out what the combination of words builds up to, so the model can properly be encoded. There is also an embedding layer which transforms the encoded words that are put into vectors so that the models can train on them [2]. These vectors represent similar words or category of words. This particularly important for NLP.

Two models I am going to build is simple RNN and LSTM. LSTM is different to RNN in that it converges faster than RNN. Another thing about LSTM it has different types of gates for its neurons, for example, it has input gates which controls what are important inputs that the neuron can take in, it also has forget gates which is used to preserve long term information and erase information that is not needed, and it has output gates which distinguishes information that it should read and output [2]. In summary this is why the Long Short-Term Memory Neural Network got its name.

Diagram

Description automatically generated

*Figure 1-7 from Gerons*

*Text

Description automatically generated*

*Formula 1-8*

For my Simple RNN, my layers are one Embedding Layer, one Bidirectional Layer, one Dense Layer all with 64 neurons, and one output layer with the sigmoid function which is used for Binary Classification. For LSTM model there is one Embedding Layer, Two Bidirectional Layers, one Dense Layer all with 64 neurons, one Dropout Layer for regularization, and one output layer with the sigmoid function used for Binary Classification. These are represented in 1-9, and 1-10. As you can see LSTM and Simple RNN are specified in the Bidirectional Layers which is the special parts of a RNN.

Text

Description automatically generated

*Simple RNN Model 1-9*

Text

Description automatically generated

*LSTM RNN 1-10*

**Review Results and Evaluate Model**

After implementing the models, it was time to review the performance for each of the models. For the Simple RNN in 1-9, I got an underfitting problem, with a Training Accuracy of 0.6970 on 10 epochs and Test Accuracy of 0.6420. For the Test Data I got the Confusion Matrix in 1-13. After I found the confusion matrix, I computed the Precision and Recall scores as well as F1 Score and found Precision and F1 to be very low seen in 1-14 through 1-15. The F1 score is was 0.559 and precision score was the highest at 0.72, but recall score was the lowest around 0.454. I was unsatisfied with Simple RNN for the most part.

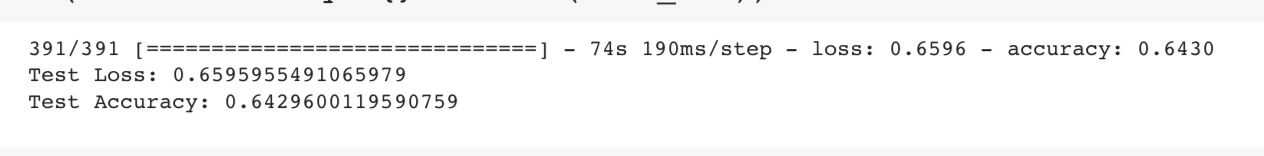
F1 score takes into account both precision and recall where 0 is the worst score and 1 is the best score [2]. Precision is a fraction of correctly identified positive results over all the instances identified as positive, while Recall is a fraction of correctly identified positive result over all the real positives. For Recall and Precision, 0 is bad, and 1 is the best score, so one could see they were all very good scores but could be unsatisfied with the Recall score.

In the LSTM model I got a pretty good Training Accuracy at 0.8386 and Test Set was around the same accuracy at 0.8224 which means this model was just about right. I then looked at the Confusion Matrix, F1 Score, Recall and Precision to see if they were underperforming as well, and I saw that they were all performing well compared to Simple RNN. Precision score was around 0.83, Recall score around 0.812 and F1 score around 0.82 all better than the scores for the Simple RNN model.

A picture containing text

Description automatically generated

*Epochs for Simple RNN 1-11*

**

*Test Set Accuracy for RNN 1-12*

*Text

Description automatically generated*

*Test Data Confusion Matrix for Simple RNN 1-13*

*Graphical user interface, text, application

Description automatically generated*

*Recall and Precision score RNN 1-14*

*Text

Description automatically generated*

*F1 Score RNN 1-15*

*A picture containing table

Description automatically generated*

*Epochs for LSTM 1-16*

**

*Test Set Accuracy LSTM 1-17*

*Text

Description automatically generated*

*Test Data Confusion Matrix for LSTM 1-18*

*Text

Description automatically generated*

*Recall and Precision score LSTM 1-19*

*Graphical user interface, text, application

Description automatically generated*

*F1 score LSTM 1-20*

**Implementation and Programming**

For implementation of the code, the first step is to import the packages as seen in code 1-21. The most important packages that I used below are the **Tensorflow** package, **Keras** package, **numpy** package and **matplotlib**. I also checked the versions of Tensorflow and Keras which were 2.3.0 and 2.4.0 respectively. The first major step was to load the IMBD dataset with 8,000 words. This was done with the function **tfds.load() from** Tensorflow package, using the path to dataset. It was easy to split up the dataset by using the load function. I split it up as there was a dictionary in the dataset which had the keys train and test set. I also had metadata on the file which was saved in the variable info.

Next I wanted to see how the encoder provided worked. This was done by saving the encoder from the metadata in a variable named ‘encoder’ as written by this line **encoder = info.features[‘text’].encoder**. I also saw how many words the encoder could encode which was 8,185. Before implementing the models, I needed to pad to the longest text which was 64 words. Padding is useful so that it is ready for training, and all texts is of the same size. This was done by **.padded\_batch().**

After adding padding, I then created a few barplots to see if negative and positive reviews were equal, and my plots confirmed they were. This was done by plt.bar() function from matplotlib package. After creating an EDA, I then implemented the model using keras and tensorflow package. I created the first model as seen in 1-9 and second model in 1-10. The model was created using **tf.keras.sequential() with Embedding() layer and Bidirectional() layers SimpleRNN or LMST explicitly stated.** I then used .compile to compile both models, and I used **.fit()** function to train the model which showed all the epochs and accuracy at each epochs between training and validation set.

The accuracy and loss were shown. I then got the accuracy of test data using **evaluate()** function. I then plotted the losses and training accuracy for both models using a custom made function which utilized Matplotlib. I then created confusion matrix using **SKLearn** package. I first used **.predict()** on test dataset, and then utilized **numpy.where** function to get the classes. I used **tensflow.concat** to get the true classes at axis =0 of tensor dataset object and saved true classes and predicted classes in a variable and passed it into confusion matrix function. Later I also imported **F1score, Precision, and Recall score functions** from **SKLearn** package as well. See in appendix below for more details.

Text

Description automatically generated

*Code Cell 1-21*

**Exposition, problem description, Management recommendation**

As examined above, I made a Simple RNN model and a LTSM Model. I then examined the results and I found that the scores were higher for the LTSM Model. I thought LTSM model would perform better as I realized it converges better to predictions. In my LTSM Model I got a Test Accuracy of 0.8224, a Precision Score of 0.83, a Recall Score of 0.812, and a F1 Score of 0.82. These were all relatively high. Therefore, **for my recommendation to management the LTSM Model should be chosen with one Embedding Layer, Two Bidirectional Layers, one Dense Layer all with 64 neurons, one Dropout Layer for regularization, and one output layer with the sigmoid function used for Binary Classification** as seen in model 1-10. 1-16 through 1-20 show the results of my model and justify my recommendation as indicated here.

For future analysis, I would like to figure out how make Simple RNN run better as I thought the scores were very low. I also want to figure out how I can make the LTSM accuracy go into high 90’s along with Precision, Recall and F1 Scores. I think I would also want to play around with making the model fit faster as GPU/TPU took over 2 hours for both of these models to run. I used Google Colab for this analysis, but it still ended up taking forever.

References

[1] Srinivasan, S. (2020b, November 8th). *Sync Session 8* [Slides]. Canvas.

<https://canvas.northwestern.edu/courses/125893/modules/items/1698004>

[2] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O’Reilly Media.

**Appendix Next Page:**