

Text Classification using AI and non-AI approaches

CSE518 Artificial Intelligence

Varun Deliwala

AU1940034

Abstract

The project revolves around how text classification can be implemented using both AI and non-AI approaches. The dataset used here is “Sentiment140 dataset with 1.6 million tweets” from Kaggle in which the tweets are denoted a Boolean value for positive (1) and negative (0). For the AI approach, the model is developed using LSTM (long short-term memory) which is a special type of RNN (Recurrent Neural Network) which are capable of learning long-term dependencies, which is intricately present in this problem. Another AI model that is based on DenseNets is also implemented here, to provide a comparative analysis between the AI approach.

A non-AI approach is also implemented here as essentially, if a problem can be solved with a non-AI approach and can also provide a substantial level of accuracy, then the need for an AI approach becomes moot. For the non-AI approach, we ranked the words which might be considered as positive text as they might be repeated in phrases e.g., “Looks Good.” The words then are ranked using TF-IDF vectorizer. This gives us the understanding that if there is a positive phrase, we can directly classify any text containing the positive phrase as a positive text.

What is Text Classification

When working on a large amount of unstructured data that is in the form of texts, be it emails, tweets, conversation, analyzing the data and extracting insights from the data can be monotonous and time consuming. Using Natural Language Processing (NLP) and Machine Learning, we can sort textual data much faster and in an easier manner.

Text Classification organizes, structures, and categorizes text in any format into appropriate field buckets. It is the base of sentiment analysis and can also be used for other applications like topic labeling, spam recognition etc. A text classifier takes a phrase as an input, analyzes it and then automatically assigns it a relevant field.

Twitter as a use-case of Text Classification

More than 500 million tweets are sent every day and twitter being a free-speech platform also faces a lot of cases of hate driven tweets. When scouring over 500 million tweets, a program might not be able to classify the tweet efficiently and accurately into hate-speech or any other forms of speech. This can be because 80% of the tweets are unstructured, messy and difficult to analyze. Text classification can automatically categorize all manner of text in an efficient, accurate and most importantly, cost-effective manner.

Non-AI approach

For the non-AI approach, we used the TF-IDF vectorizer to make a TF-IDF sorting array. This array contains the ranks of the words based on their current usage in a positive or a negative sentence. Then we implemented an N-Gram rule Engine, which is based on the TF_IDF vectorizer. The rule engine which would take as input, the text and compare it with the respective n-grams, derived from the datasets. The engine essentially compares to classify a particular text for the 2 cases, being positive and negative. The interested and not_interested dataset is created because, the phrases associated with the respective datasets might be different. A bigram, a trigram and a quad gram rule engine have been implemented. This is a heuristic approach, which utilizes, non-AI methods to identify the sentiments in the text.

AI Approach

The Artificial Intelligence model is based on the concept of Long Short-term memory. This is a form of an artificial neural network, more particularly, recurrent neural network. The second model that is implemented in the AI section is the Dense neural network, which essentially utilizes the connection between layers through dense blocks in a feed-forward nature. Here each layer obtains additional inputs from all preceding layers and the feature-maps are passed on to all upcoming layers.

Steps in implementing Text Classification

The dataset that is being used here requires preprocessing as some of the columns need to be dropped and at the same time the sentimental values need to be encoded.

Stemming and Lemmatization

Tweets that are being used here consists of @, hyperlinks, punctuations and other hindrances which need to be removed in order to use the tweets to learn using the LSTM model. So, the data here requires various cleaning and preprocessing methods. The tweets here have words which belong to the same root, such as drink, drinks and drinking. At the same time, there are related words which derived from the root and have a similar meaning. The main goal of Stemming and Lemmatization is to squeeze the words to their original root form via different methods.

Stemming, as the word suggests is reducing the object from the ends. For words it is the process of chopping off the ends to achieve the original root form. For example, drinking can be stemmed in a manner to provide the word drink. This is a more robust approach as it does not really care if the new formed word, is a word or not and just focuses on removing the ends, sometimes the derivational affixes of the words, like 'ing', 'es', 'ly' etc. Lemmatization on the other hand focuses on using morphological analysis and the use of vocabulary to analyze the word and then removes the endings to return the dictionary root form of the word.

Train/Test Split

Just like any other dataset on which a machine learning model must work, this data set is also required to be split into train_set and test_set. We are using the 80-20 ratio of train to test in splitting.

Tokenization

For a sequence of characters in a defined unit, tokenization chops the document into smaller pieces called tokens, sometimes omitting characters like punctuations. This can help in assigning meaning in an easier fashion. This essentially is the process of converting a complex sequence into smaller lesser complex sequence which can be processed by the model.

Word Embedding

Word embedding is a form or word representation of the document vocabulary in a n-dimensional space. It has the capability to capturing context of a word in a document and at the same time produce a semantically and syntactical relation of one word with other words. It essentially is a feature vector representation which can be better understood by computers.

Model Training

LSTM

LSTM has the capability of processing not just one data point but also the entire sequence of data with the help of feedback connections. It has the features of both long-term and short-term memory and as a result this provides the short-term memory of RNN which can last for substantial amount of time steps. These networks can work well with classification, prediction and time-series problems. Originally developed to deal with the vanishing gradient problem, as the network is insensitive to gap length, the model works better than RNN, HMMs and other sequence models. The LSTM unit essentially can maintain the error in its unit cell and this can continuously feed error back to the LSTM, till the model learns the cut off value for the gradient.

Here the input goes through a pipeline of multiple functions in the model.

Dropout -> Convolution Network -> LSTM -> DenseNet1 -> Dropout -> DenseNet2 -> Output

Then during the process of training using Adam as a gradient descent algorithm which is based on adaptive estimation of the initial order moments, and learning rate controller which reduces the learning rate near the plateau, which is like hill climbing to find the closest maxima and at the same time prevents overshooting.

Model Evaluation

The model performance can be judged over the accuracy and model loss for every epoch.

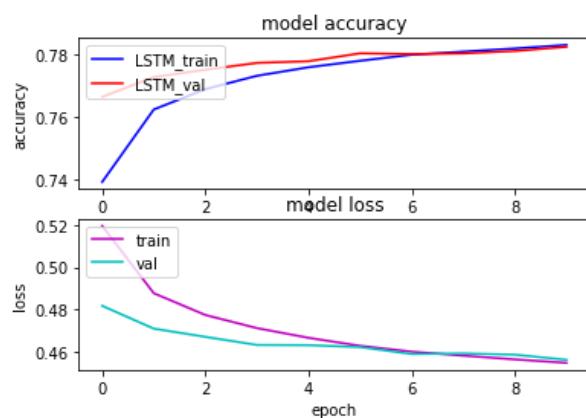


Figure 1 Model Accuracy and Model Loss for LSTM

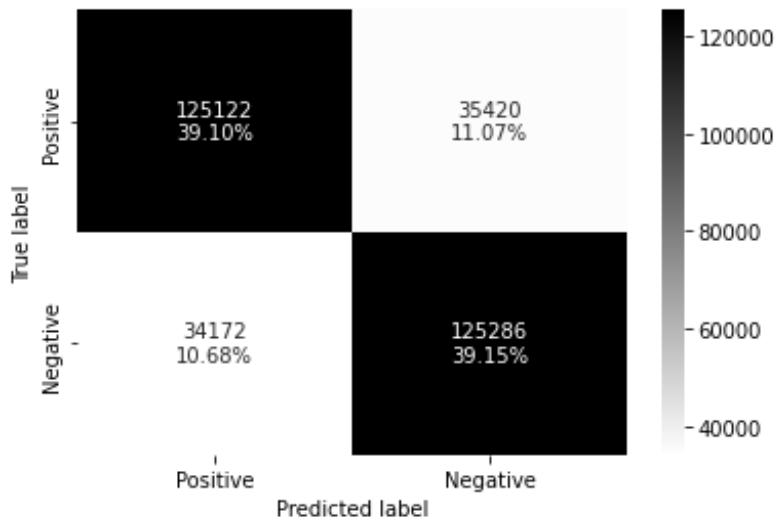


Figure 2 confusion matrix for lstm

	precision	recall	f1-score	support
Negative	0.79	0.78	0.78	160542
Positive	0.78	0.79	0.78	159458
accuracy			0.78	320000
macro avg	0.78	0.78	0.78	320000
weighted avg	0.78	0.78	0.78	320000

Figure 3 Classification Score for LSTM

DenseNet

The DenseNet architecture is a type of CNN that uses the connection between the multiple layers which are connected directly with each other. DenseNet can be used to improve the accuracy which might be already affected by Vanishing Gradient in high-level Neural Network. Here we have implemented a much shorter and lesser complex version of DenseNet as we are just trying to compare the performance output of both the models.

The Dense Net here is in the following format.

Dense (1024) -> Dense (512) -> Dense (128) -> Dropout -> Dense (512) -> Output

Model Evaluation

The model performance can be judged over the accuracy and model loss for every epoch.

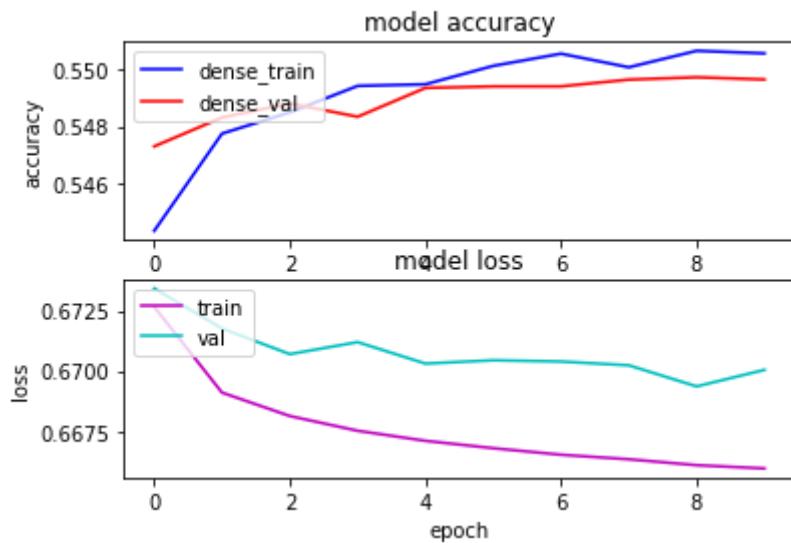


Figure 4 Model Accuracy and Model Loss for DenseNets

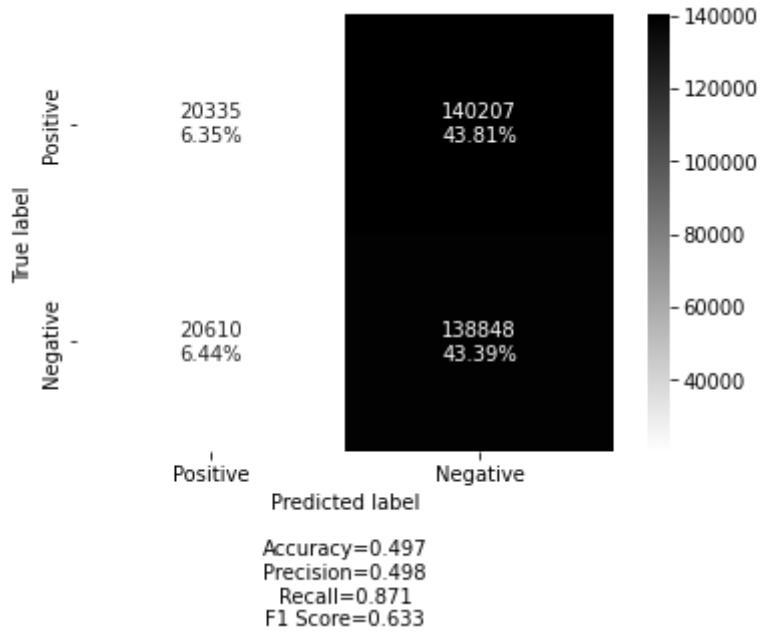


Figure 5 Confusion Matrix for DenseNet

	precision	recall	f1-score	support
Negative	0.50	0.13	0.20	160542
Positive	0.50	0.87	0.63	159458
accuracy			0.50	320000
macro avg	0.50	0.50	0.42	320000
weighted avg	0.50	0.50	0.42	320000

Figure 6 Classification Score for LSTM

LSTM vs DenseNet

After implementing both the models we can extrapolate the following insights.

- 1) The training time for the LSTM model is substantially higher than the training time of the DenseNet model.
- 2) On the other hand, LSTM provides us with much better accuracy and much lower loss over the Epochs, than DenseNet and has a much better precision, recall and f1-score.

Overall, if trained better DenseNet, might be able to perform better than LSTM in terms of Resource Usage, but is lesser accurate than LSTM. So, it would different for special cases in which DenseNet would be preferred over LSTM because of lack of resources.

Bibliography

- 1) Data set <https://www.kaggle.com/datasets/kazanova/sentiment140>
- 2) Word Embedding <http://nlp.stanford.edu/data/glove.6B.zip>
- 3) <https://www.kaggle.com/code/kredy10/simple-lstm-for-text-classification>
- 4) <https://www.kaggle.com/code/arunrk7/nlp-beginner-text-classification-using-lstm>
- 5) <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- 6) https://keras.io/api/layers/recurrent_layers/lstm/
- 7) <https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a>
- 8) <https://doi.org/10.48550/arXiv.1608.06993>
- 9) <https://medium.com/analytics-vidhya/sentiment-analysis-on-amazon-reviews-using-tf-idf-approach-c5ab4c36e7a1>
- 10) <https://arxiv.org/ftp/arxiv/papers/1806/1806.06407.pdf>