Deep Learning Project

Ahmed JARDAK

Sentimental Analysis on Tweets

Ecole National d’ingénieur de Tunis

Abstract

In this project, a deep learning approach was utilized to conduct sentiment analysis on tweets. The objective of the study was to categorize tweets into positive and negative sentiments. The project served as a practical application and evaluation of the deep learning concepts taught by InstaDeep.

1. **Setting up the environment.**

"requirements.txt", found in the root directory of the project is a text file that contains a list of all the libraries used in a project. These libraries, "dependencies", are necessary for the project. I

The file was generated by using the command "pip freeze > requirements.txt".

To set up their environment, run the command "pip install -r requirements.txt" which will install all the necessary libraries listed in the requirements file.

2. **Data loading:**

The Sentiment140 is a dataset that contains 1.6 million tweets labeled with their sentiment. The dataset is sourced from the popular social media platform Twitter and was created by researchers at Stanford University. The tweets are labeled as either positive or negative. The tweets contain a variety of different languages and cover a wide range of topics and it is available on Kaggle, a platform for data science competitions,.

The dataset includes columns for the sentiment, tweet ID, timestamp, and the text of the tweet.

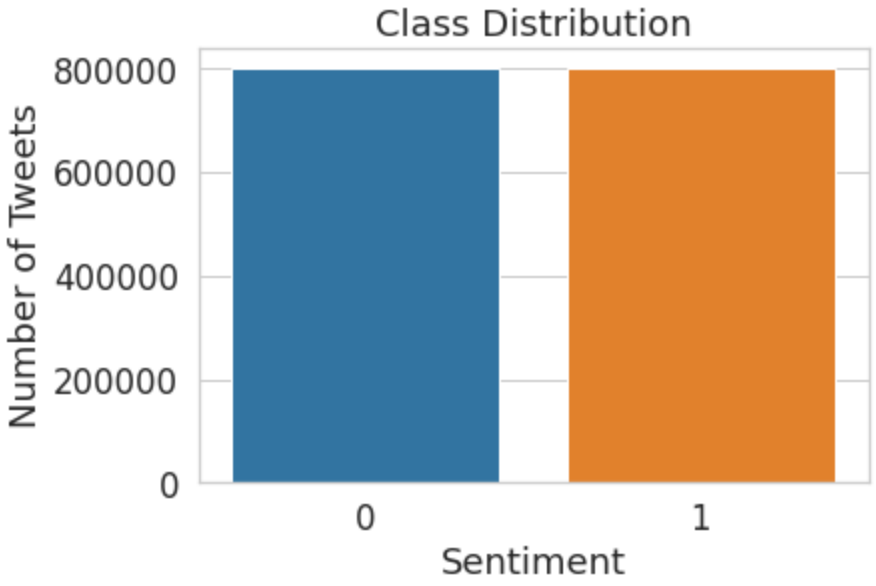
1.6 million rows could be considered as a large volume of data, and it will be relatively computationally expensive to train and it will take time, of course.

For that reason, I wrote a cell to downsample the data, while keeping the same balance between both classes (50% - 50%).

3. Data Analysis:

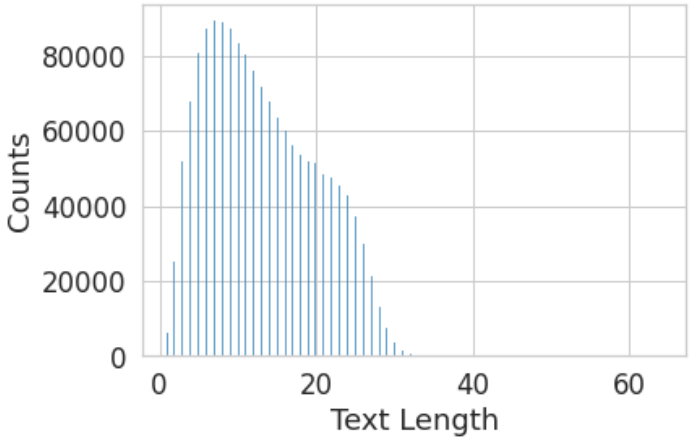
3.1 Class balance:

As it would be a mandatory information for the pre-processing of data, we had to check the classes distribution, for that reason we plotted a bar chart that contained all the information.



3.2 Text length analysis

For this section, we added a column to our dataframe, containing the text length for every tweet, this column was analyzed then to get a better understanding of the distribution of the length.



4. Data pre-processing:

4.1 Text Pre-processing

This part defines a text preprocessing function that takes in a text string as an input and performs several text cleaning and normalization steps. The first step is to convert all the characters in the text to lowercase. The next step is to remove mentions, hashtags and urls from the text. This is done by using a regular expression to search for patterns that match mentions, hashtags and urls and replacing them with an empty string. Then, the text is tokenized into individual words using the word_tokenize function from the nltk library. The next step is to remove stopwords from the text. Stopwords are commonly used words such as "the", "a", "an", "in", etc. that do not carry much meaning and can be removed to reduce the dimensionality of the data. The set of english stopwords is loaded using the stopwords library. After that, the text is stemmed using the SnowballStemmer from the nltk library.

This step reduces words to their base form, so that words that have the same meaning but different conjugations or tenses are treated as the same word. Finally, the cleaned tokens are joined back into a string and returned as the output of the function.

The function is then applied to a specific column ('text') of the dataframe 'df' to create a new column ('text_cleaned') with the cleaned text. With this step, the text is ready for further processing, such as tokenizing and padding, which is important for text classification tasks. Another tokenizing step was taken, using the Keras API, and applied a padding on all the column so all sequences has the same length

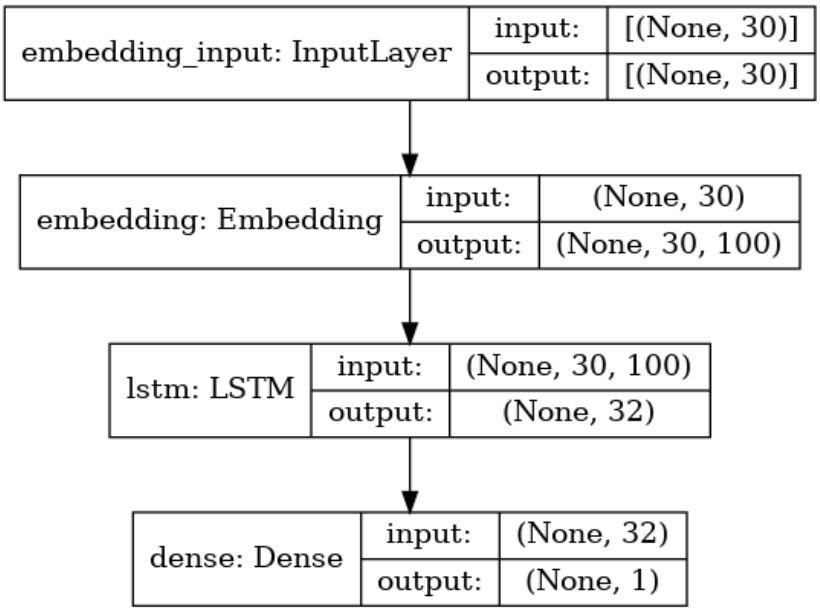
4.2 Data splitting

The data is split here, with 80% being training data and 20% being test data, using sklearn's train_test_split() function.

5. Modelling

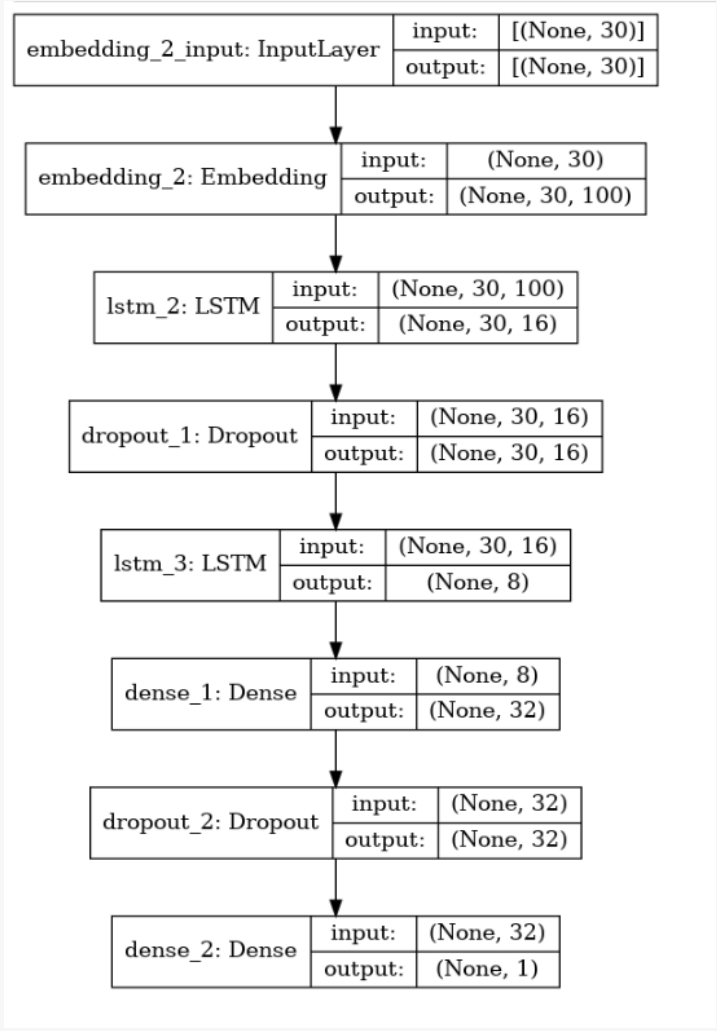
    5.1 First Architecture:

    This is a sequential neural network with an Embedding layer, followed by an LSTM layer, and a Dense layer. The Embedding layer converts text into dense representations for the LSTM layer to process. The LSTM layer allows the model to learn sequential patterns in the data. The Dense layer with sigmoid activation outputs a probability of the input being in the positive class.



5.2 Second Architecture:

This is a sequential neural network with an Embedding layer, followed by an LSTM layer, and a Dense layer. The Embedding layer converts text into dense representations for the LSTM layer to process. The LSTM layer allows the model to learn sequential patterns in the data. The Dense layer with sigmoid activation outputs a probability of the input being in the positive class.



5.3 Evaluation:

The following figure shows the confusion matrix, as an evaluation of the model: