TWEET SENTIMENT EXTRACTION
| DEEP LEARNING |

UNIVERSITÉ JEAN MONNET
MLDM

Mostafa Hajj CHEHADE

Mohamad Badia ALFATHI

January 2023

Abstract

Every second, thousands of tweets from various users with different thoughts are seen on Twitter. Therefore, it is challenging to detect the sentiment underlying these tweets and their effects on specific individuals, businesses, and brands. In order to determine if a tweet has a positive or negative effect on the users, it would be interesting to investigate the sentiment of the tweets. Thus, we would extract keywords and phrases from tweets to determine the sentiment. On the Tweet Sentiment Extraction[1] Kaggle-provided dataset, analysis and extraction are performed. The dataset that we considered contains a wide set of tweets that are classified into 3 different types of sentiments, namely, positive, negative, and neutral.

In this report, we discuss the sentiment analysis of tweets using different deep-learning models and compare their performance.

Keywords: *Sentiment analysis, Tweet extraction, Tweets sentiment, Deep learning, NLP, Neural network, MLP, CNN, RNN, LSTM.*

[1] Maggie, Phil Culliton, Wei Chen. (2020). Tweet Sentiment Extraction | Kaggle. (n.d.).

Contents

1 Introduction

On the social networking site Twitter, millions of tweets are posted every day covering a wide range of subjects. As a result, a large dataset of tweets that contains important information about the public's perception of many subjects may be developed every day. Tweet sentiment extraction using deep learning and NLP is a method of analyzing the sentiment, or emotion, expressed in a tweet using deep learning techniques. Deep learning is a subfield of machine learning that involves training neural network models on large datasets to recognize patterns and make predictions.

In the context of tweet sentiment extraction, deep learning models can be trained on a dataset of tweets and their corresponding labels (positive, neutral, or negative) to classify new tweets based on their sentiment. The models can learn to recognize patterns in the tweet text that are indicative of positive or negative sentiment, such as the use of certain words or phrases. There are a number of deep learning architectures that can be used for tweet sentiment extraction, including convolutional neural networks (CNNs), Multilayer perceptron (MLP), and long short-term memory (LSTM) networks. Depending on the characteristics of the dataset and the objectives of the research, these models can be trained using a variety of methods, including supervised learning, unsupervised learning, or semi-supervised learning. In our case, we have a labeled dataset by the sentiment of the tweets.

In this research, we investigate several deep learning algorithms, particularly neural network models that may be used to extract words from tweets and classify tweet sentiment. Here, we cover a number of topics, beginning with an explanation of the data, followed by an overview of the exploration of the data, an application of natural language processing, and finally the algorithms and neural networks that were used in the project, their structures, and the implementation. We have a hypothesis that the specific data gives better results than those that contain the full text of the tweet, so we built the models and trained them on two types of inputs in the training data, which enables us to compare between the models and understand the impact of the inputs on them and confirm or refute this hypothesis.

2 Overview of dataset

The dataset is part of the "Tweet Sentiment Extraction" completion on Kaggle. It consists of train, and test files. The data being processed is labeled with the sentiment of tweets. Thus, each tweet of the 27481 tweets contains up to 4 data points:

- ***textID:*** which is a unique ID for each piece of text
- ***text:*** the full text of the tweet
- ***selected_text:*** the part of the tweet that strongly exemplifies the sentiment. It is found only in the train data.
- ***sentiment:*** Finally, our label column that represents the sentiment of the tweet as either positive, negative, or neutral.

2.1 Data Exploration

We have 27481 tweets in the train set and 3534 tweets in the test set. One Null value in the train data was removed. As discussed, the data includes 3 types of sentiments. So, in Figure 1 we can see the distribution of the label classes.

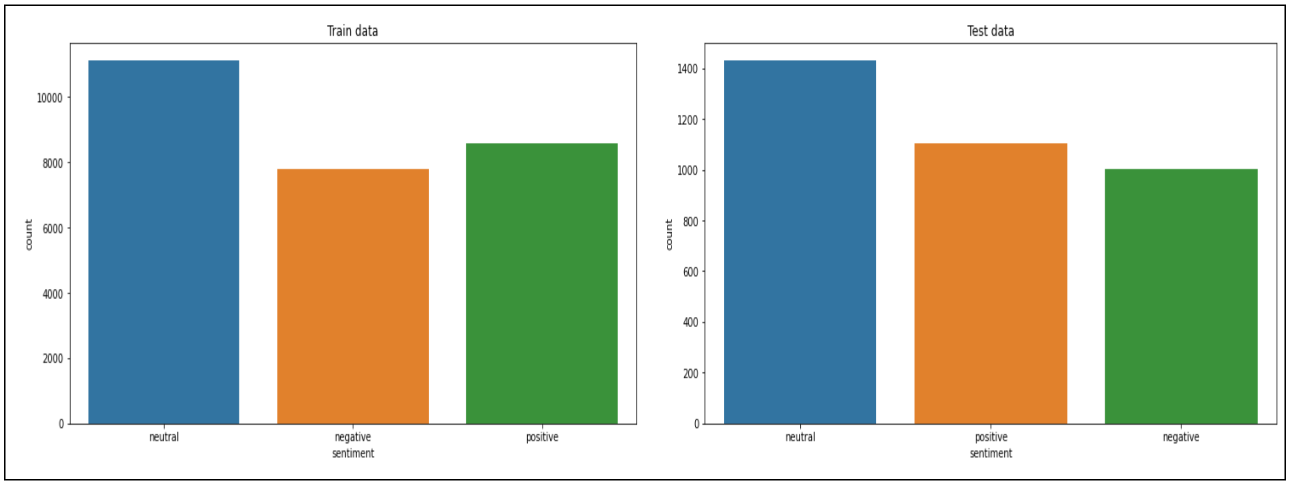


FIGURE 1: SENTIMENT CLASSES IN TRAIN AND TEST.

Then, we thought that it would be interesting to see the difference in word count distribution for the full tweet and the selected text. Because there are so few tweets with more than 25 words, the number of words distribution plot in Figure 2 is right-skewed. In most cases the number of selected words in range [0-5].

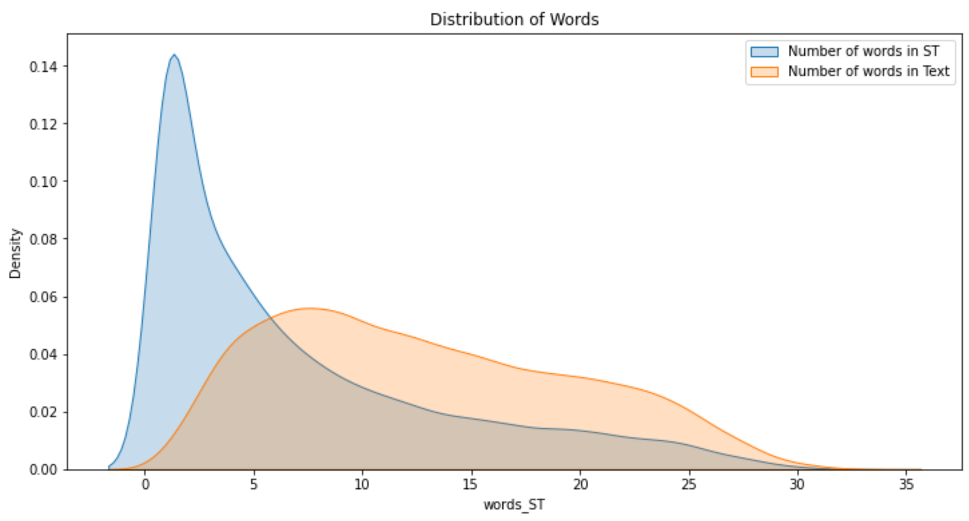


FIGURE 2: NUMBER OF WORDS DISTRIBUTION

From Figure 3 it is obvious that most neutral tweets have the same text and selected text. On the other hand, the Figure 4 shows the distribution of the difference in the number of words for other two labels. The density of the number of words in the full tweet and selected text is shown for each sentiment in Figure 5.

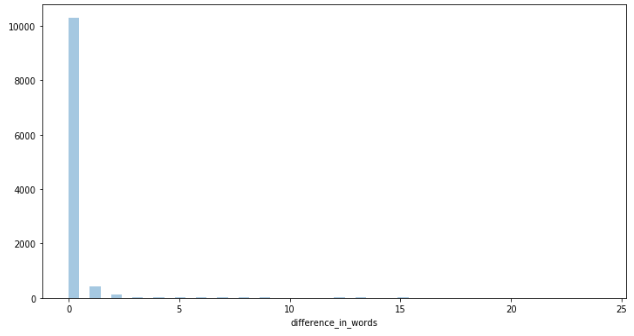


FIGURE 3: DIFFERENCE IN NUMBER OF WORDS [NEUTRAL]

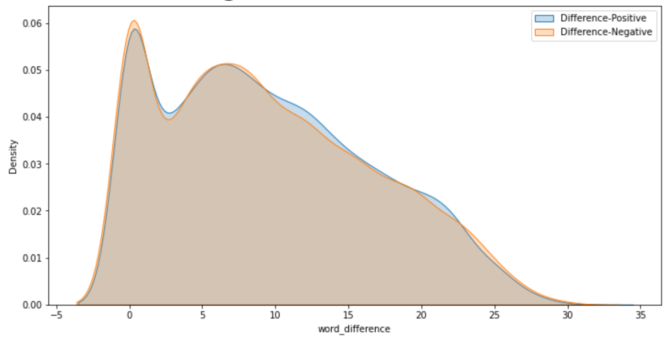


FIGURE 4: DIFFERENCE IN NUMBER OF WORDS [POSITIVE & NEGATIVE]

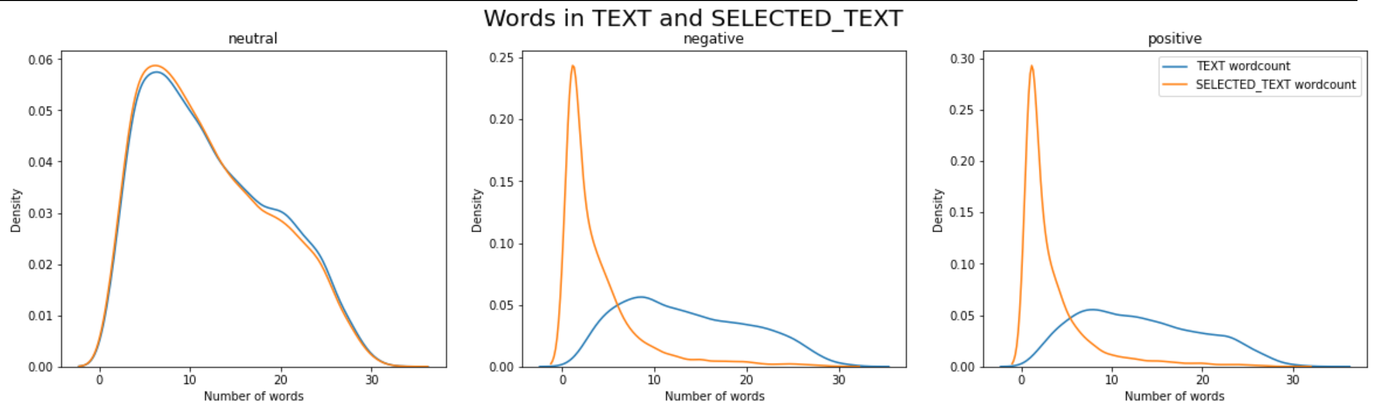


FIGURE 5: NUMBER OF WORDS

2.2 Data processing

Data preprocessing is the process of cleaning, transforming, and preparing data for further analysis or modeling. It is an important step in the data analysis before start models implementation. There are types of data processing based on the type of data. In the tweets experiment the data is texts. So, we deal with text processing[2] which is an important step in natural language processing (NLP) to clean and transform unstructured and noisy data in order to be useful for analysis or modeling. We employed the following preprocessing methods in our study:

1. Convert the entire text to be written in lowercase letters.
2. Removing any data that won't help with sentiment classification. (numbers, URLs, Https, punctuations, special characters).
3. Tokenization: This involves splitting the text into individual tokens, such as words or phrases.

As a first step before building the model and extracting the sentiments, we cleaned the data (text and selected text) by the preprocessing techniques mentioned above.

2.3 Data Splitting

The data is divided into three sets: a training set, validation, and test. The test set is reserved for evaluating the final performance of the model. Specifically, 70% of the data is included in the training set, and the remaining 30% is divided equally into the validation and test sets.

2.4 Data Tokenization & Vectorization

To be able to use text data as input for our models, we need to first convert it into a form that the models can process. One way to do this is to define a Tokenizer that converts the text into sequences of token indices. This process involves mapping each token (which can be a character, word, subword, or other element) to a unique index, allowing us to represent the text as an array of indices that the model can understand. In this way, the vectorization helps us to convert our text data into a numerical form that is usable by the model.
We add the Embedding layer[3] that allows you to feed the Tokenized data as input to a neural network model. The Embedding layer takes in a 2D tensor of integer values (the token indices) and maps them to a 3D tensor of embedding vectors.

[2] Reshamwala, A., Mishra, D., & Pawar, P. (2013). Review on natural language processing. IRACST Engineering Science and Technology: An International Journal (ESTIJ), 3(1), 113-116.
[3] Team, K. (n.d.). Keras documentation: Embedding layer.

3 Modeling

In our project, we used several neural networks with different architectures. We divided the study into 2 types of inputs: Full tweet | selected part. The performed deep learning models:

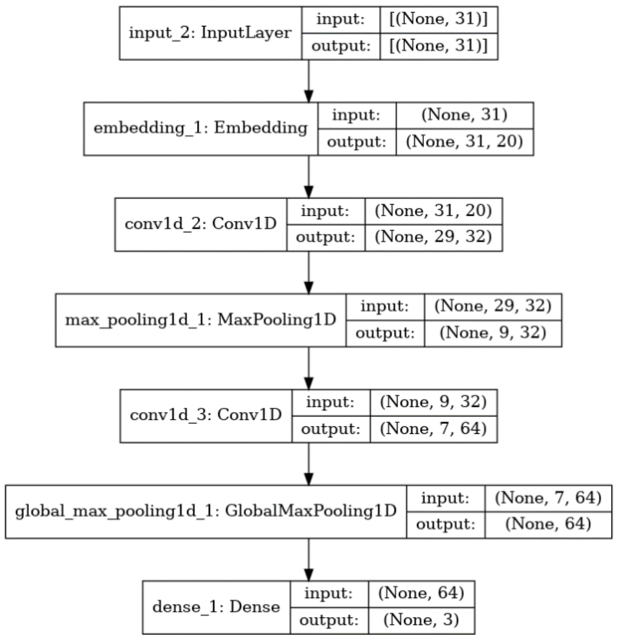
1- The Model class groups layers.
2- Convolutional neural network (CNN).
3- Recurrent neural network (RNN, LSTM).
4- Multilayer Perceptron (MLP).

3.1 Neural networks architectures

This section will discuss the performed neural networks that we used, their structures, layers and do they work. There are many different types of neural network architectures and training algorithms, in our study we work on the most popular networks can be implemented in the multi-classes problem.

3.1.1 The Model class

This model is a neural network model using the tf.keras functional API in TensorFlow. The model consists of an input layer where the data fed followed by an embedding layer, two convolutional layers, two pooling layers, and a dense layer which applies a fully connected operation to the input data and the output of the dense represent the predicted probabilities for each of the 3 classes as shown in Network 1: The Model*Network 1*.



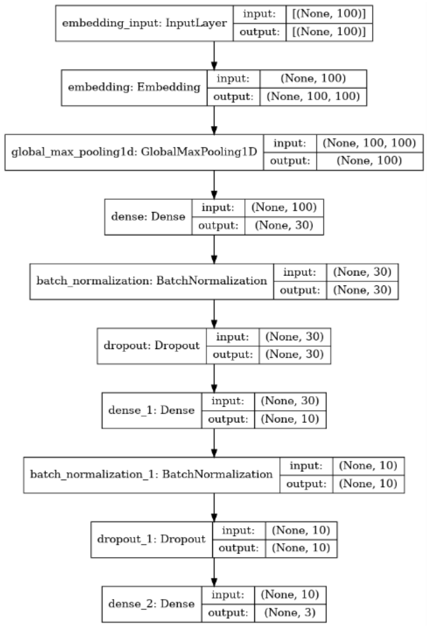
NETWORK 1: THE MODEL

3.1.2 Multilayer Perceptron (MLP)

An MLP consists of an input layer, one or more hidden layers, and an output layer.[4] Each layer is fully connected to the next layer, and the hidden layers typically have non-linear activation functions.

The model defined has an input layer (the embedding layer) and an output layer (the final dense layer with 3 units and a SoftMax activation function) and includes several hidden layers in between (the dense layers with 30 and 10 units).

An embedding layer, which maps the input data (indices of tokenized data) to a dense vector representation (embedding). The input dimension of the embedding layer is the size of the vocabulary, and the output dimension is a fixed size (embedding_dim of 100). The model is then compiled with the fixed optimizer, activation, epochs, and batch to select the best of each one of them.

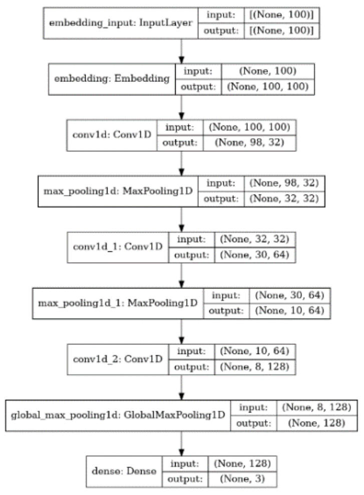


embedding_input: InputLayer | input: [(None, 100)] | output: [(None, 100)]
embedding: Embedding | input: (None, 100) | output: (None, 100, 100)
global_max_pooling1d: GlobalMaxPooling1D | input: (None, 100, 100) | output: (None, 100)
dense: Dense | input: (None, 100) | output: (None, 30)
batch_normalization: BatchNormalization | input: (None, 30) | output: (None, 30)
dropout: Dropout | input: (None, 30) | output: (None, 30)
dense_1: Dense | input: (None, 30) | output: (None, 10)
batch_normalization_1: BatchNormalization | input: (None, 10) | output: (None, 10)
dropout_1: Dropout | input: (None, 10) | output: (None, 10)
dense_2: Dense | input: (None, 10) | output: (None, 3)

NETWORK 2: MULTILAYER PERCEPTRON

3.1.3 Convolutional neural network

CNNs are particularly effective for tasks related to image classification, object detection, and segmentation, as they are able to learn features and patterns from the input data that are important for these tasks.[5] Convolutional neural networks (CNNs) can be used for text classification tasks also, such as sentiment analysis or spam detection. A CNN model for tweet sentiment extraction typically consists of an embedding layer to convert the integer indices of the tweets into dense, continuous-valued vectors, followed by one or more convolutional layers to extract features from the embedded tweets. Pooling layers can be used to reduce the spatial dimensions of the feature maps, and fully connected layers can be used to make the final prediction based on the extracted



embedding_input: InputLayer | input: [(None, 100)] | output: [(None, 100)]
embedding: Embedding | input: (None, 100) | output: (None, 100, 100)
conv1d: Conv1D | input: (None, 100, 100) | output: (None, 98, 32)
max_pooling1d: MaxPooling1D | input: (None, 98, 32) | output: (None, 32, 32)
conv1d_1: Conv1D | input: (None, 32, 32) | output: (None, 30, 64)
max_pooling1d_1: MaxPooling1D | input: (None, 30, 64) | output: (None, 10, 64)
conv1d_2: Conv1D | input: (None, 10, 64) | output: (None, 8, 128)
global_max_pooling1d: GlobalMaxPooling1D | input: (None, 8, 128) | output: (None, 128)
dense: Dense | input: (None, 128) | output: (None, 3)

NETWORK 3: CNN NETWORK

[4] Jenkins, O. C. (1995). Multilayer Perceptron Networks: A Review. Neural Networks, 8(1), 3-23.
[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

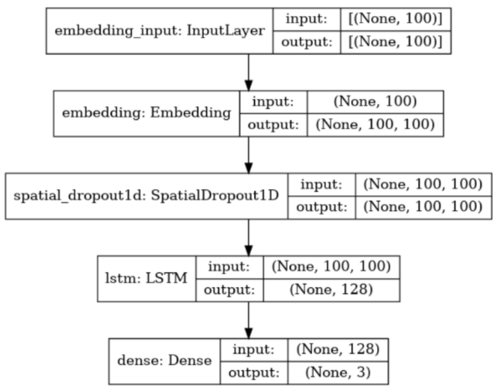
features. So, we built different architectures where we obtained close accuracy results with a slight improvement. Similar to what we have done in the rest of the neural networks we tested different activation functions, optimizers, epochs and batches.

3.1.4 Recurrent neural network (RNN)

Recurrent neural networks (RNNs) are a type of neural network that allows for information to be passed from one step of the network to the next. This makes them well-suited for tasks that involve sequential data, such as natural language processing, speech recognition, and machine translation.[6] RNNs have been used in the field of sentiment analysis, including in the context of Twitter sentiment extraction. We tried a simple RNN model using the Keras API. The function begins by defining the embedding dimension as 100 and creating a Sequential model. It then adds an embedding layer to the model, which converts the input text data into dense vectors of fixed size. Adding a dropout layer after the RNN layer. Dropout is a regularization technique that randomly sets a fraction of the units in a layer to zero during training, which helps prevent overfitting by reducing the complexity of the model.[7] The dropout layer is followed by a simple RNN layer with 64 units, and finally, a dense layer with 3 units and a 'softmax' activation function, which is used for classification. We will try using LSTM layers instead of SimpleRNN layers, where LSTMs are better able to capture long-term dependencies in sequential data and may perform better on this task than SimpleRNNs.

3.1.5 Long short-term memory (LSTM)

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) that is particularly well-suited for modeling long-term dependencies in time series data. However, standard RNNs can struggle to retain information over very long time periods due to the vanishing gradient problem. LSTMs address this problem by introducing a new structure called the "memory cell" that



NETWORK 4: LSTM

[6] Valdés-Sosa, R. E., Corneanu, C., & Shafran, I. (2015). An empirical exploration of recurrent network architectures.

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research.

can store information for long periods of time. Our model consists of an embedding layer, which maps the words in the tweets to a lower-dimensional vector space, a spatial dropout layer, which helps to prevent overfitting, and an LSTM layer with a specified activation function. The model is trained using the categorical cross-entropy loss function and an optimization algorithm like other models will be selected based on the best performance.

4 Neural networks performance

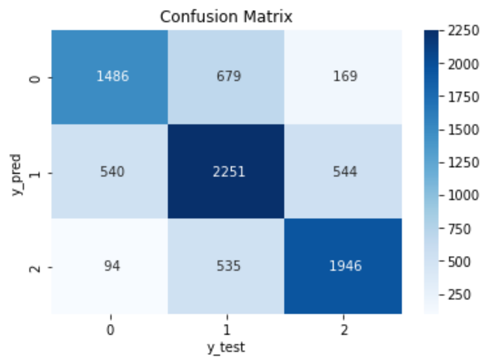
After we discussed the architecture of the performed neural networks, in this section we will analyze the performance of the models and the best hyperparameters selected by the model. We will consider the two cases of our study as mentioned before, the model implemented on the full tweets, and the selected part of the tweet.

4.1 Full text - performance

In this study, sentiment analysis was performed on the full text of a dataset using various neural network methods. The text was vectorized using an embedding layer, where the embeddings were generated during the training process. The performance of the networks was compared by evaluating their accuracies, F1 scores, and time.

4.1.1 The Model class

In this model seems to be performing fairly well, with an overall accuracy of 69%. The f1-score is also fairly high for each class as shown in Figure 6, indicating that the model is making a good balance of true positive and true negative predictions. However, there is room for improvement, as the precision and recall scores are not perfect.



```
              precision    recall  f1-score   support

           0       0.70      0.64      0.67      2334
           1       0.65      0.67      0.66      3335
           2       0.73      0.76      0.74      2575

    accuracy                           0.69      8244
   macro avg       0.69      0.69      0.69      8244
weighted avg       0.69      0.69      0.69      8244
```

FIGURE 6: CONFUSION MATRIX – MODEL [FULL TEXT]

4.1.2 Multilayer Perceptron (MLP)

The accuracy of the MLP model is 0.674, which means that the model was able to correctly predict the class (label) for 67% of the samples in the dataset.

The overall f1-score for the model is shown in the "weighted avg" row as 0.672. A high f1-score is generally desirable, as it means that the model is making a good balance of true positive and true negative predictions and is able to correctly classify a high proportion of the samples in the dataset. The best activation function is ***tanh.*** The best optimizer is ***RMSprop*** With ***best epoch 5*** and best ***batch size 16*** as shown in Figure 7.

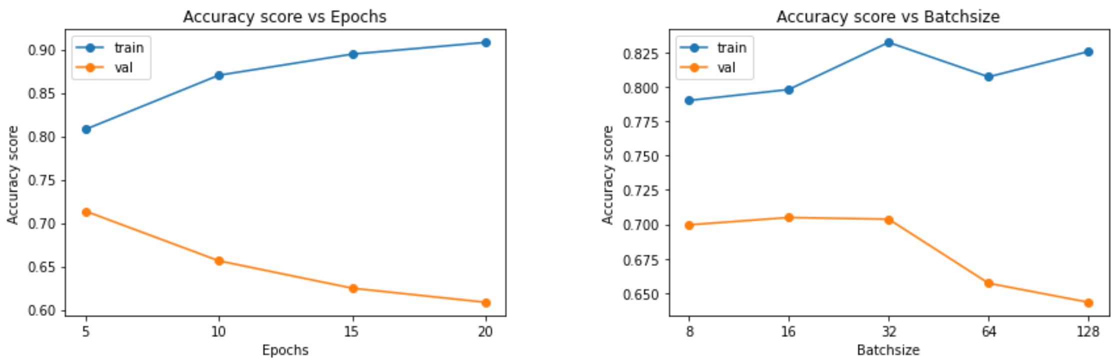


FIGURE 7: PERFORMANCE - MLP [FULL TEXT]

4.1.3 Convolutional neural network

Based on the CNN model, the best activation function for the model was ***selu***, the best optimizer was ***RMSprop***, the ***best epoch was 5***, and the ***best batch size was 16*** see Figure 8 where the train accuracy increases with epochs and almost same for batch size unlike the val accuracy. The final test accuracy of the model was 0.689, and the test f1 score was 0.687 which is slightly less than the using this model on the selected text (as we will see later). Also, the training process took about 86.64 seconds to complete.

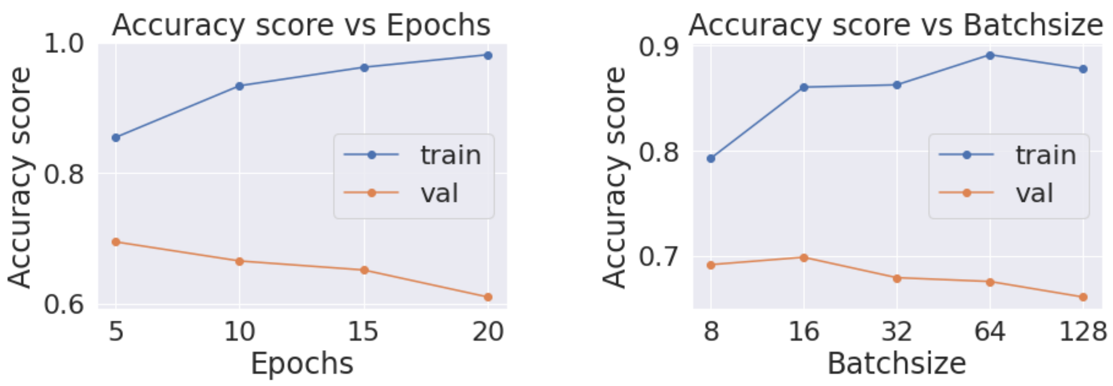


FIGURE 8: PERFORMANCE - CNN [FULL TEXT]

4.1.4 Recurrent neural network (RNN)

The RNN model was trained on text data, with the goal of performing sentiment analysis on the input text. The model was trained with different activation functions, optimizers, epochs, and batch sizes, and the results were compared to find the best combination of hyperparameters for the model. The best combination was found to be the ***elu*** activation function, the ***adam*** optimizer, ***10 epochs***, and ***a batch size of 16***, with a test accuracy score of 0.613 which is the worst accuracy in this experiment even with trying different architectures and a time taken of 1100.7 seconds.

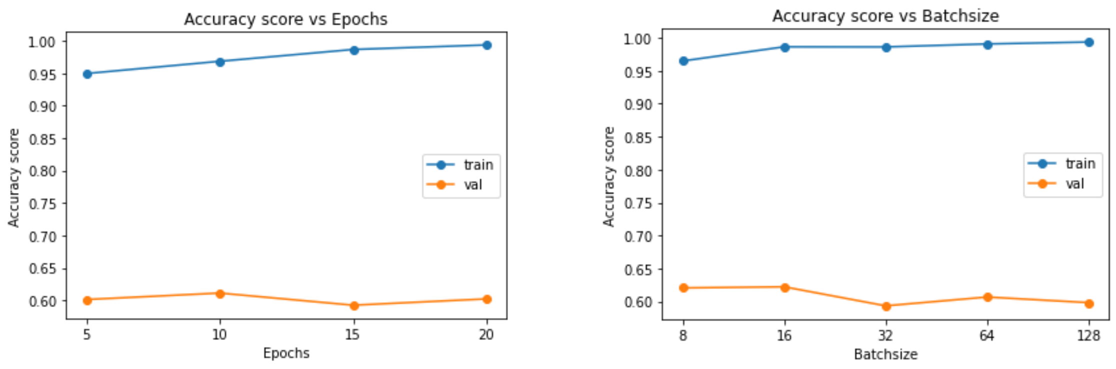


FIGURE 9: PERFORMANCE - RNN [FULL TEXT]

4.1.5 Long short-term memory (LSTM)

The LSTM almost the best model between the networks in both cases. It was better on the selected text. However, the test accuracy score of 0.7239 indicates that the model was able to correctly predict the labels or categories for about 72.4% of the examples in the test set, and the time taken of 345.84 seconds to be trained that is significantly higher. The Figure 10 present the train and val accuracy based on the Epochs=5 and Batch size=64 where the best activation function is ***tanh,*** the best optimizer ***RMSprop.***

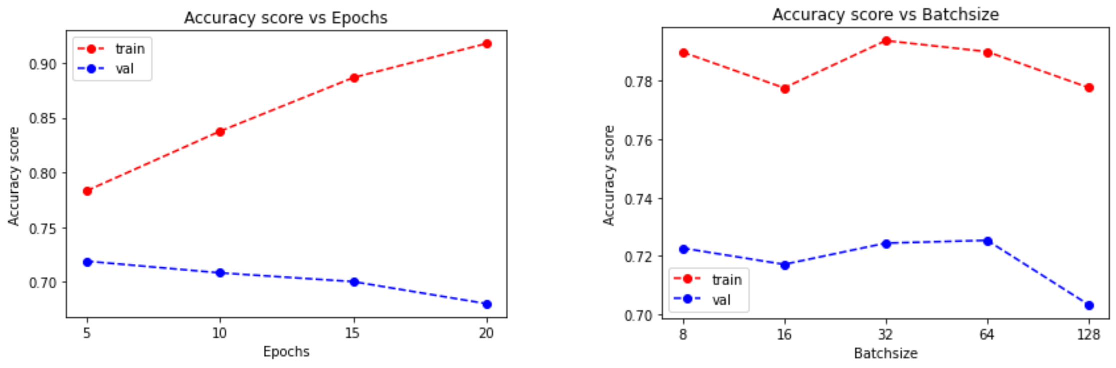


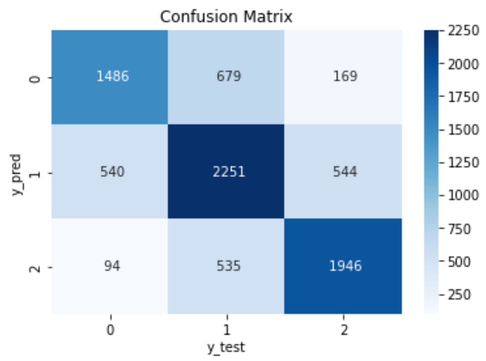
FIGURE 10: PERFORMANCE - LSTM [FULL TEXT]

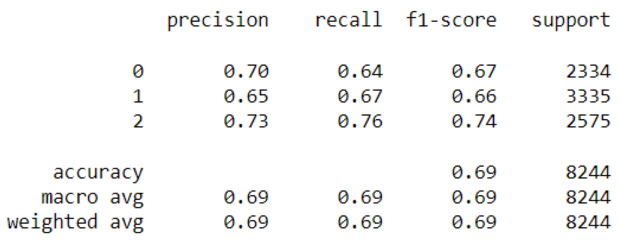
4.2 Selected text - performance

In this case, the sentiment analysis involved using the selected text as the input. The same set of experiments were conducted for both this case and another one (Full tweet) in order to compare the performance of various neural network methods in both scenarios. The goal was to evaluate the effectiveness of these methods for sentiment analysis using selected text versus some other input.

4.2.1 The Model class

In the case of using selected text classification report Figure 11, the overall accuracy of the model is 0.65, which is slightly lower than the accuracy of 0.69 in the first report. This means that the model is slightly less accurate at predicting the sentiment (label) for the samples of selected text in this model with the same architecture. The f1-score for each class is also generally lower in the second report compared to the first report. The overall f1-score for the model, as shown in the "weighted avg" row, is 0.63, which is lower than the overall f1-score of 0.69 in the first report.





```
              precision    recall  f1-score   support

           0       0.70      0.64      0.67      2334
           1       0.65      0.67      0.66      3335
           2       0.73      0.76      0.74      2575

    accuracy                           0.69      8244
   macro avg       0.69      0.69      0.69      8244
weighted avg       0.69      0.69      0.69      8244
```

FIGURE 11: CONFUSION MATRIX – MODEL [SELECTED TEXT]

4.2.2 Multilayer Perceptron (MLP)

MLP for selected text looks like the best activation function for the model was ***relu***, the best optimizer was ***RMSprop***, the best ***epoch was 5***, and the ***best batch size was 16***. The final test accuracy of the model was 0.7193, and the test f1 score was 0.72 which is clearly better than the same one on Full text. It also looks like the training process took about 142.59 seconds to complete. The Figure 12 shows that the increasing with epochs and batch size gives higher train accuracy

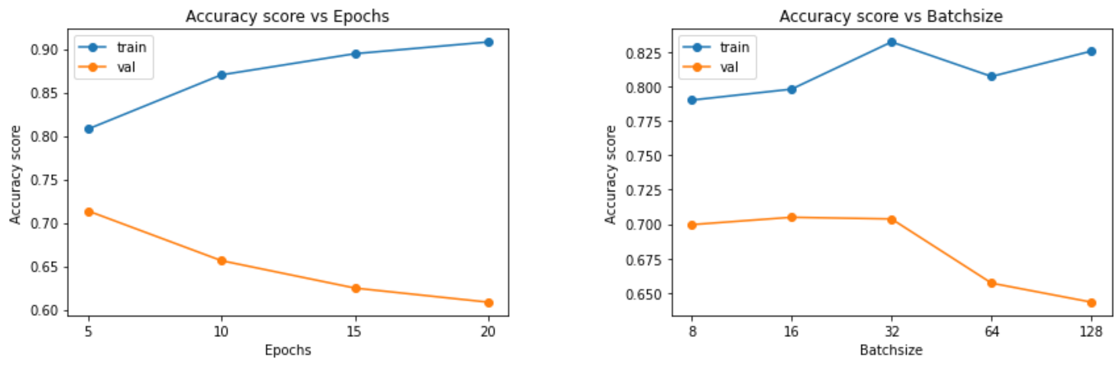


FIGURE 12: PERFORMANCE - MLP [SELECTED TEXT]

4.2.3 Convolutional neural network

The CNN model on the selected text appears that the best activation function was ***tanh***, the best optimizer was ***RMSprop***, the ***best epoch was 5***, and the ***best batch size was 8*** see Figure 13 . The final test accuracy for the first model was 0.71, and the test f1 score was 0.711. It also looks like the training process took about 154.48 seconds to complete.

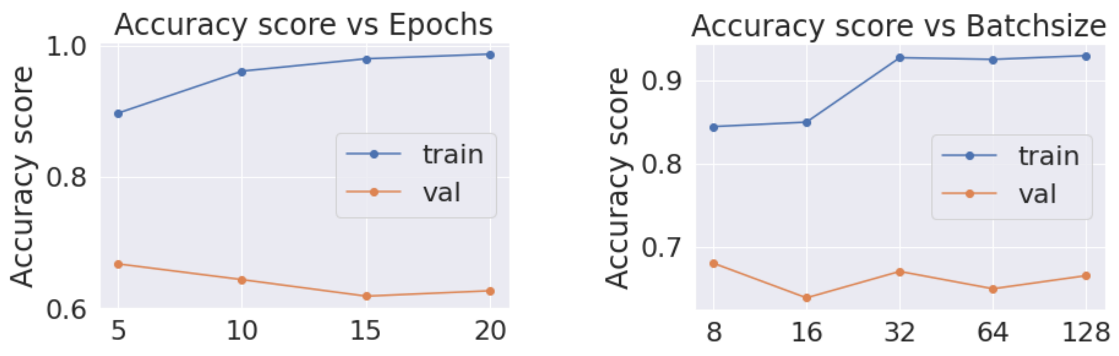


FIGURE 13: PERFORMANCE - CNN [SELECTED TEXT]

4.2.4 Recurrent neural network (RNN)

Applying RNN on the selected text gives us small improvement than the full text but it is still poor. The accuracy we have here about 66% with a big time consuming even using the Kaggle notebooks.

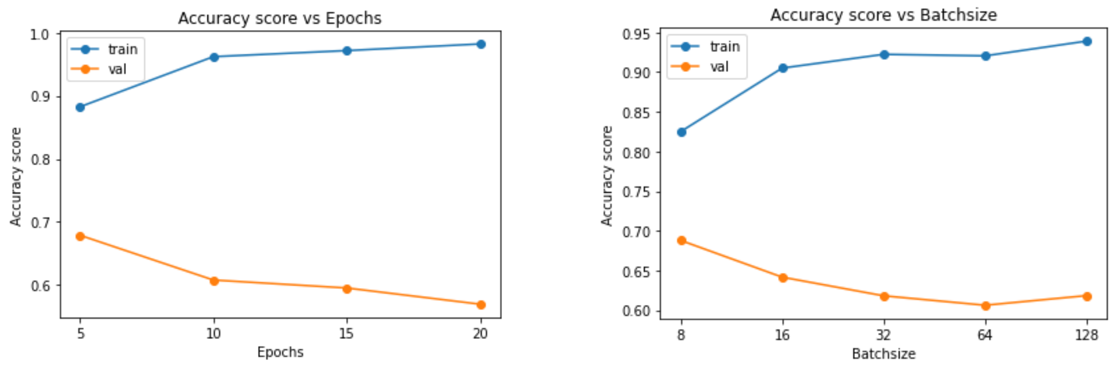


FIGURE 14: PERFORMANCE - RNN [SELECTED TEXT]

4.2.5 Long short-term memory (LSTM)

The best accuracy score performed by LSTM network on the selected text. We get about 73.5% for the same best activation, and optimizer with the first LSTM model.

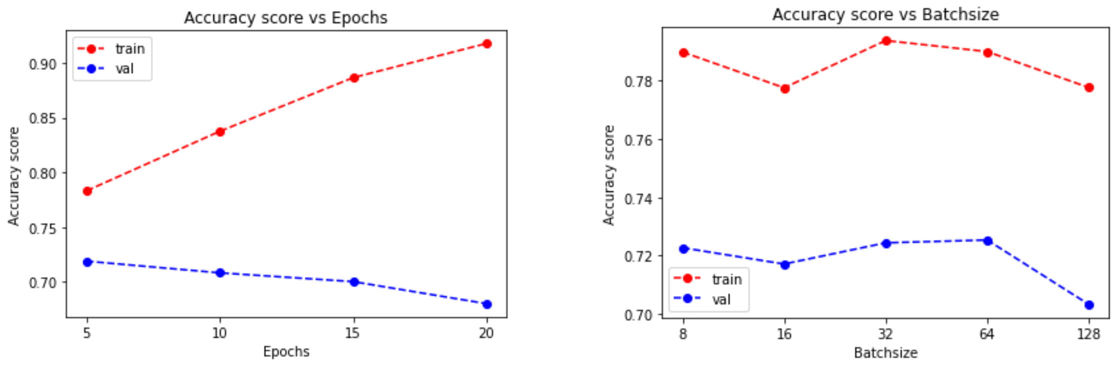


FIGURE 15: PERFORMANCE - LSTM [SELECTED TEXT]

4.3 Improvement using LSTM- GloVe

Thanks to ChatGPT suggest using the GloVe pretrained model with the best performed neural networks which is LSTM. Combining LSTM with pre-trained word embeddings such as GloVe is a common approach to improve the performance of NLP models. The idea is to use the pre-trained word embeddings as the initial weights for the input layer of the LSTM, instead of randomly initializing them. This allows the model to benefit from the information about the semantic relationships between words that is captured by the pre-trained embeddings, and to learn more efficiently from the training data.[8] It appears that the best activation function is 'relu', optimizer is 'Adam', best epoch is 5 and best batchsize is 32. The final test accuracy score is 75%.

4.4 Performance Comparison

In the Performance Comparison section, the results of using different neural network configurations were compared for sentiment analysis tasks on two different input text datasets: the full text and the selected text. The experiments were identical for both cases, allowing for a fair comparison of the performance of the neural network methods. The results in Table 1 showed that, in general, using the selected text as input led to slightly better performance in terms of accuracy and F1 score. However, the difference in performance was not significant, and further experimentation may be necessary to fully understand the impact of using the selected text on the performance of sentiment analysis tasks.

[8] GloVe: Global Vectors for Word Representation.

	Full text			**Selected text**		
Model	Accuracy	F1 score	Time	Accuracy	F1 score	Time
Model class	69	69		65	63	
MLP	67.4	67.2	173	71.9	72	143
CNN	68.9	68.7	87.5	71	71.1	155
RNN	61.3	61.4	1101	66	65.7	1127.5
LSTM	72.4	72.2	346	73.5	73.2	929.5
LSTM-GloVe	73.9	73.7	342	75	74.6	339

TABLE 1:PERFORMANCE COMPARISON

	Full text				**Selected text**			
Model	Best Activation	Best optimizer	Epochs	Batch size	Best Activation	Best optimizer	Epochs	Batchsi ze
MLP	tanh	RMSprop	5	16	ReLU	RMSprop	5	16
CNN	Selu	RMSprop	5	16	tanh	RMSprop	5	8
RNN	Elu	Adam	10	16	tanh	RMSprop	5	8
LSTM	tanh	RMSprop	5	64	tanh	RMSprop	10	16
LSTM-GloVe	Elu	Adam	5	32	ReLU	Adam	5	32

TABLE 2: BEST HYPERPARAMETERS

5 Conclusion

From the experiments conducted, it can be concluded that the performance of neural network models for Twitter sentiment extraction depends on various factors such as the type of model, the choice of hyperparameters, the type and preprocessing of the input data, and the quality of the training data.

In the first case, where the input data was the full text of the tweet, the LSTM model performed the best, with a test accuracy of around 72.4%. The RNN model did not perform well, with a test accuracy of around 61.3%. The performance of models was improved by using the different optimizers and increasing the number of epochs to 5. In the second case, where the input data was the selected text of the tweet, the LSTM model again performed the best, with a test accuracy of around 73.5%.

Overall, it can be concluded that the deep learning models can be effective for Twitter sentiment extraction, with the LSTM model generally outperforming. 5. Also, the RMSprop optimizer was the best among the optimizers we tested (relu, elu, selu, Adadelta and Adam). However, the performance of these models can be improved by carefully choosing the appropriate hyperparameters and ensuring that the input data is properly preprocessed and enough. Moreover, the accuracy of these models can be improved by using pretrained word embeddings such as Glove. Pretrained word embeddings capture the semantics of words in a more meaningful way and can help the model learn better representations of words in the input text.

Note: the project done by Mostafa Hajj CHEHADE, Mohamad Badia ALFATHI almost equally 50% each.

6 References

[1] Maggie, Phil Culliton, Wei Chen. (2020). Tweet Sentiment Extraction | Kaggle. (n.d.).

[2] Reshamwala, A., Mishra, D., & Pawar, P. (2013). Review on natural language processing. IRACST Engineering Science and Technology: An International Journal (ESTIJ), 3(1), 113-116.

[3] Team, K. (n.d.). Keras documentation: Embedding layer.

[4] Jenkins, O. C. (1995). Multilayer Perceptron Networks: A Review. Neural Networks, 8(1), 3-23.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[6] Valdés-Sosa, R. E., Corneanu, C., & Shafran, I. (2015). An empirical exploration of recurrent network architectures.

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research.

[8] GloVe: Global Vectors for Word Representation.

[9] EDA For Sentiment Analysis.

[10] [Full Tutorial] EDA to DNNs, all you need!

[11] RNN baseline model.

[12] Tweet-Sentiment-RNN.

[13] Twitter Sentiment Analysis using CNN DeepLearning