Dylan Govender – 221040222
COMP703 – Assignment 2 Report – 10/05/2024

[1] Long Short-Term Memory (LSTM) Sentiment-Analyser

[1.1] Importing the Dataset
The original dataset encoding was obsolete; hence, it was reconstructed into UTF8 format using the *recode* command API. The dataset contains six components; only two will be used for this application: *target* and *text* components. The data used for the LSTM model is saved as the data frame, *df.*

[1.2] Data-Preprocessing for LSTM:
From the six components, four components were removed from the data frame. The sentiment of the data, the *target* component, in numbers, was mapped back to its actual sentiment. From the *text* component, using *Regular Expressions*, URLs, usernames, symbols, and stopwords were removed. *NLTK*, the Natural Language Toolkit library, was used to download the stopwords dictionary and wordnet, both necessary for preprocessing this data. Wordnet is a **lemmatisation** method that is better than stemming, which considers a word's context to convert it to its meaningful base form, lemma. Each word was lemmatised and lower-cased. The cleaned text was saved as a new component in the data frame named *clean_text*. Below is a sample of the original text and the cleaned text.

	target	text	clean_text
0	negative	is upset that he can't update his Facebook by ...	upset update facebook texting might cry result...
1	negative	@Kenichan I dived many times for the ball. Man...	dived many time ball managed save 50 rest go b...
2	negative	my whole body feels itchy and like its on fire	whole body feel itchy like fire
3	negative	@nationwideclass no, it's not behaving at all....	behaving mad see
4	negative	@Kwesidei not the whole crew	whole crew

Cross-validation was used to split the data frame into a train set, *train_set,* and a test set, *test_set*, using *train_test_split* from the *sklearn* library in an 8:2 ratio, respectively. Each token in the *text* component of the data frame was tokenised and padded, then encoded into numerical data using the *LabelEncoder* provided by the *sklearn—preprocessing* module. **Tokenisation** converts text into a sequence of numerical data. **Padding** was used to ensure that all the tokenised text had the same length, *MAX_SEQ_LEN*, of 42, the average sequence length of the text data. Below is the shape of the *train_set* and *test_set* after tokenisation, padding and encoding.

```
X_TRAIN SHAPE: (1279999, 42)
Y_TRAIN SHAPE: (1279999, 1)
X_TEST SHAPE: (320000, 42)
Y_TEST SHAPE: (320000, 1)
```

[1.3] Configuring and Training the LSTM Model:
The LSTM model was built using *Tensorflow* and *Keras* API. The model was initialised using the *Sequential* class, and each layer required to make the model was then added to the class. Since this application required word embeddings, a pre-trained word embedding, GloVe 6B tokens

with a dimension of 300, was used to create the custom embedding layer for the LSTM, which was used to convert tokens into vectors, which allows the model to capture semantic meaning and context. The NLP Standford Research Group provided the pre-trained word embeddings, which can be downloaded via their website. The Spatial Dropout 1D Layer is applied to the embedding outputs to prevent overfitting by dropping entire 1D feature maps. The LSTM Layer was used to configure the LSTM model and capture long-range dependencies between sequences while mitigating the vanishing gradient problem. The Dropout Layer prevents overfitting, which drops a small fraction of the input units during training, ensuring the model's robustness. The Dense Layer is responsible for the predictions and will give a single output, a percentage, due to using the activation function Sigmoid, which will suppress the output between 0-1. The regularisation techniques, L1 and L2, were included in the dense layer to prevent overfitting by penalising large weights in the model, ensuring robustness.

The model was compiled using categorical-cross-entropy, the standard choice for multiple classification applications, which estimates the difference between the classes' actual and predicted distributions. The Adam optimiser was used to adapt the learning rate during training.

Two callbacks were employed during training to reduce training time and improve overall training results. *EarlyStopping*, *early_stopping*, was used to monitor the validation loss and stop training if it didn't improve during a certain number of epochs (patience). ReduceLROnPlateau, *reduce_lr*, reduced the learning rate when the validation loss stopped improving. The model was then fitted onto the train_set, x_train, and y_train using the *model.fit* function.

[1.4] <u>Evaluating the LSTM Model</u>:
The performance of the LSTM was assessed using metrics such as accuracy, precision, recall, and F1-score by employing the Scikit-Learn library on the test set, *y*. The model achieved an accuracy of 83%, an F1-score of 83%, and a validation loss of 39%. The validation loss measures how well the model performs on unseen data; the smaller the loss, the better the model performs and signifies that the model's predictions are closer to the actual labels. Accuracy measures the proportion of correctly predicted sentiment from the total number of predictions, indicating that 83% of the model's predictions were correct. It is calculated as follows:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

In the above calculation, the *True Positive (TP)* and *True Negative (TN)* denote the number of correct predictions, and the *False Positive (FP)* and the *False Negative (FN)* denote the number of incorrect predictions where the model failed to identify a particular sentiment according to the actual sentiment. These are called discriminator metrics. Precision evaluates the model's ability to identify relevant instances, and recall measures the precision at the gold standard, revealing whether a model could capture relevant instances.

$$Precision = \frac{TP}{TP + FP} \quad \text{and} \quad Recall = \frac{TP}{TP + FN}$$

The F1-score uses precision and recall for computation. F1-score measures the model's performance in identifying and capturing all relevant instances. The F1-score for the LSTM model is 83%, which signifies that the model could identify and capture 83% of the instances. For this report, accuracy, validation loss, and F1-score offer a more pertinent analysis for comparing both models. F1-score is calculated as follows:

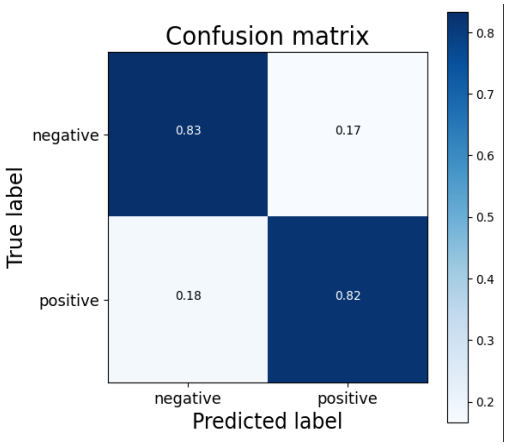
$$F1\text{–}score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

[1.4.1] Classification Report and Confusion Matrix of the LSTM Model:

```
32/32 [==============================] - 4s 123ms/step
              precision    recall  f1-score   support

    negative       0.82      0.83      0.83    159494
    positive       0.83      0.82      0.83    160506

    accuracy                           0.83    320000
   macro avg       0.83      0.83      0.83    320000
weighted avg       0.83      0.83      0.83    320000
```



[2] Gated Recurrent Unit (GRU) Sentiment-Analyser

[2.1] Importing the Dataset

The original dataset encoding was outdated; hence, it was reconstructed into UTF8 format using the *recode* command API. The dataset contains six components; only two will be used for this application. The data used for the GRU model is saved as the data frame, *df.*

[2.2] Data-Preprocessing for GRU:

Every preprocessing technique used for the above LSTM model was used here. The preprocessing methods, code, and data were unchanged.

[2.2] Configuring and Training the GRU Model:

The configuration is the same as the LSTM model, using the *TensorFlow* and *Keras* API and with the customised embedding layer. The only change is that the GRU layer replaces the LSTM layer. Two GRU layers are used in the model, the same as the LSTM model. The first GRU layer returns sequences, allowing it to propagate information through the entire sequence. The second GRU layer doesn't return sequences; it only outputs the final hidden state. The GRU layer is a recurrent neural network layer that selectively updates and retains information over time, allowing it to capture long-term dependencies in sequential data while mitigating the vanishing gradient problem. The configuration has the same number of parameters as the LSTM model.

[2.4] Evaluating the GRU Model:

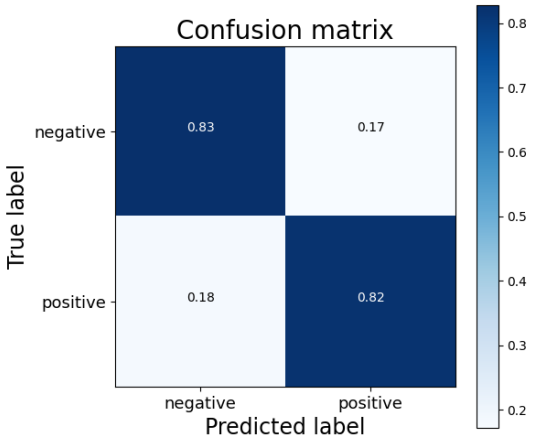
Using the same evaluation metrics used for the LSTM model, this model achieved an accuracy of 82%, an F1-score of 82%, and a validation loss of 40%. The accuracy measures the proportion of correctly predicted sentiment from the total number of predictions, indicating that 82% of the model's predictions were correct. The F1-score signifies that the model could identify and capture 82% of the instances. The slight validation loss means this model's predictions are closer to the actual labels.

[2.4.1] Classification Report and Confusion Matrix of the GRU Model:

```
32/32 [==============================] - 4s 94ms/step
              precision    recall  f1-score   support

    negative       0.82      0.83      0.82    159494
    positive       0.83      0.82      0.82    160506

    accuracy                           0.82    320000
   macro avg       0.82      0.82      0.82    320000
weighted avg       0.82      0.82      0.82    320000
```



[3] Comparison between the LSTM and GRU Model:

[3.1] Model Structure

[3.1.1] LSTM Model: LSTM is an ANN that contains separate memory cells and input/output gates. This allows it to maintain long-term memory and selectively update it over time. LSTM utilises separate input, output, and forget gates to control the flow of information through the memory cell. It generally has a higher computational complexity due to its separate gating mechanisms and memory cells. LSTM can be more robust in capturing long-term dependencies but may suffer from vanishing gradients during training.

[3.1.2] GRU Model: The GRU is an ANN that combines the update and reset gates into a single gate, resulting in a simpler architecture with fewer parameters. The GRU architecture's gate mechanism uses the update to regulate information retained previously and the reset to control the influence of past information on the current state. Due to the simpler architecture, GRUs are more computationally efficient with fewer parameters. GRUs balance complexity and performance, making them easier to train and sometimes more suitable for simpler tasks or smaller datasets.

[3.2] Model Performance

[3.2.1] LSTM Model: Due to computational complexity, LSTM offers a more robust model that can slightly outperform the GRU. This model achieved an accuracy of 83%, an F1-score of 83%, and a validation loss of 39%. The LSTM's performance is stronger than the GRU's.

[3.2.2] GRU Model: Due to a larger quantity of data, computational efficiency, and a simpler overall architecture, the GRU performs exceptionally well but slightly loses to the LSTM. GRU achieved an accuracy of 82%, an F1-score of 82%, and a validation loss of 40%.

[4] Conclusion

Based on the evaluation and comparison between each model, LSTM and GRU offer similar performance, with LSTM having a slight advantage in performance, using the same parameters, dataset, and hyperparameters.