Twitter tweet emotion sentimental analysis

DATASET: The Emotion Dataset is imported using the nlp package. The dataset is already divided into test, training and validation sets. Each set has text and label features. There are 16,000 tweets in the training set, 2,000 tweets in the test set and 2,000 tweets in the validation set.

ABSTRACT:

Millions of people use Twitter and other social media sites to share their everyday thoughts in the form of tweets. It is a short and straightforward way of expressing oneself, which is a hallmark of tweeting. As a result, we concentrated on sentiment analysis of Twitter data in our project. Sentiment Analysis is a subset of natural language processing and text data mining. It is feasible to investigate sentiment analysis using Twitter data. It is performed in a number of different circumstances. The technique of finding valuable patterns from textual data is referred to as sentiment analysis. Using particular analysis tools, these valuable patterns include evaluating and categorizing feelings as neutral, positive, or negative.

Twitter's network structure, event dispersion across the network, and impact identification. There have been several ways described for exploring semantics for sentiment analysis, which can be classified into contextual semantic and conceptual semantic approaches.

One of the most important disciplines of natural language processing is sentiment analysis. The technique of finding valuable patterns from textual data is referred to as sentiment analysis. Sentiment analysis applications will continue to develop in the future, and sentiment analytical approaches will become more standardized across systems and services. Because of the vast amount of data available, Twitter is one of the best virtual environments for tracking and monitoring information.

INTRODUCTION:

Emotions are considered of utmost importance as they have a key responsibility in human interaction. Nowadays, social media plays a pivotal role in the interaction of people all across the world. Such social media posts can be effectively analysed for emotions. Twitter is a microblogging service where worldwide users publish and share their feelings. However, sentiment analysis for Twitter messages (‘tweets’) is regarded as a challenging problem because tweets are short and informal. With the use of Recurrent Neural Networks, a model is created and trained to learn to recognize emotions in tweets. The dataset has thousands of tweets each classified in one of 6 emotions – love, fear, joy, sadness, surprise and anger. Using TensorFlow as the machine-learning framework, this multi class classification problem of the natural language processing domain is solved.

Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine. Sentiment analysis is the process of retrieving information about a consumer’s perception of a product, service or brand. Social media sentiment analysis applies natural language processing (NLP) to analyse online mentions and determine the feelings behind the post. Social sentiment analysis will tell whether the post was positive, negative, or neutral. A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level—whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as enjoyment, anger, disgust, sadness, fear, and surprise.

In this project with the use of Recurrent Neural Networks, a model is created and trained to learn to recognize emotions in tweets. The dataset has thousands of tweets each classified in one of 6 emotions love, fear, joy, sadness, surprise and anger. Using TensorFlow as the machine-learning framework, this multi class classification problem of the natural language processing domain is solved.

METHODOLOGY

The RNN model is built using Google Collab environment, which runs the Jupyter notebook in the cloud. It uses the TensorFlow as the machine-learning framework. First, all the necessary libraries are imported. Then the dataset is imported and assigned to the corresponding data object. The text pre-processing functions are done using the built-in tokenizer of TensorFlow and all the words in the dataset are assigned to a specific token. Next, the tokens are padded and truncated so that the model is input of fixed shape. Then we create a dictionary for converting the name off the classes to their corresponding index. The text labels for the different classes are passes to get them as numeric representations. The sequential model is created using four different layers. The model is then trained and evaluated.

STEPS INVOLVED:

Installing Hugging Face's nlp package

The Hugging Face's nlp package is installed using the following command: -

```
!pip install nlp
```

Importing the libraries

The following libraries are imported: -

```
import tensorflow as tf
import numpy as np
```

```
import matplotlib.pyplot as plt
import nlp
import random
import warnings
```

Further some modules from the Tensorflow library are also imported.

Importing the Dataset

The Emotion Dataset is imported using the nlp package. The dataset is already divided into test, training and validation sets. Each set has text and label features. There are 16,000 tweets in the training set, 2,000 tweets in the test set and 2,000 tweets in the validation set. Each tweet also has its corresponding emotion with it. These three sets are assigned to their respective objects. A function is also defined to get the text and label keys from the dataset. The first tweet and its corresponding label of the training set on which the model is going to be trained is displayed.

```
tweets[10], labels[10]
```

```
('i feel like i have to make the suffering i m seeing mean something',
 'sadness')
```

Tokenizing the Tweets

The TensorFlow comes with a built-in Tokenizer library which is imported from its text pre-processing module. Tokenization randomly generates a token value for plain text and stores the mapping in a database.

The words of tweets need to be tokenized so that each word can be represented as a number to feed into the model and the model is able to train on the data. The tokenizer basically creates a corpus (collection) of all the words that exist in the dataset and give each unique word a unique corresponding token. A limit is also set to how many most frequently words are to be organized and the rest less commonly used words are given a common token called out of vocabulary which is basically an unknown word token.

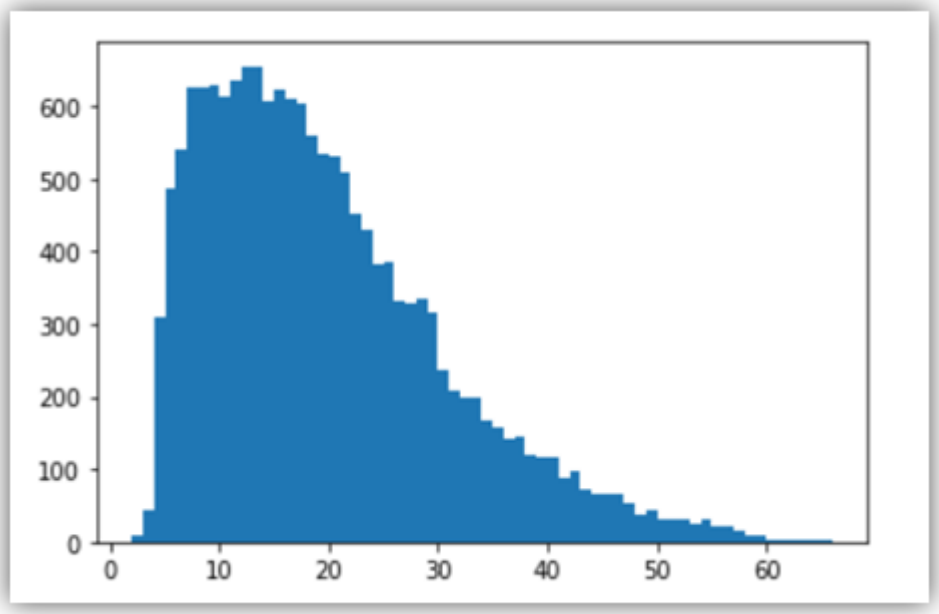
An object tokenizer is created which tokenizes the most frequently used 10,000 words from the text corpus and assigns an unknown token () to the remaining words. Then the words from the tweets from the training set are mapped to the numeric tokens using fit_on_texts function. Using the texts_to_sequences function we can see that the tweets have been tokenized.

```
print(tokenizer.texts_to_sequences([tweets[10]]))
```

```
[[2, 3, 14, 2, 21, 5, 80, 6, 733, 2, 93, 544, 304, 84]]
```

Padding and Truncating Sequences

The sequences generated from the Tokenizer need to be padded and truncated because the model requires a fixed input size. The tweets in the dataset are of different length of words and thus it is required for them to be padded or truncated. The length of the tweets is calculated by counting the number of words separated by a space. A histogram is plotted to get the most common lengths of tweets in the dataset.



Most of the tweets in the dataset are about 10 to 20 words long. There are very few tweets which are less than 4 words and also very few tweets of length 50 words or more.

A maximum length of 50 is set to truncate any tweets over the length of 50 words. Any tweet which has less than 50 words is padded with '0' in its token sequence. This is done using the pad_sequences function from the TensorFlow library. Both truncating and padding is done 'post' which means that the function will remove or add words from the end of the token sequence to get the sequence length to 50. This will get all the tweets to a fixed input size.

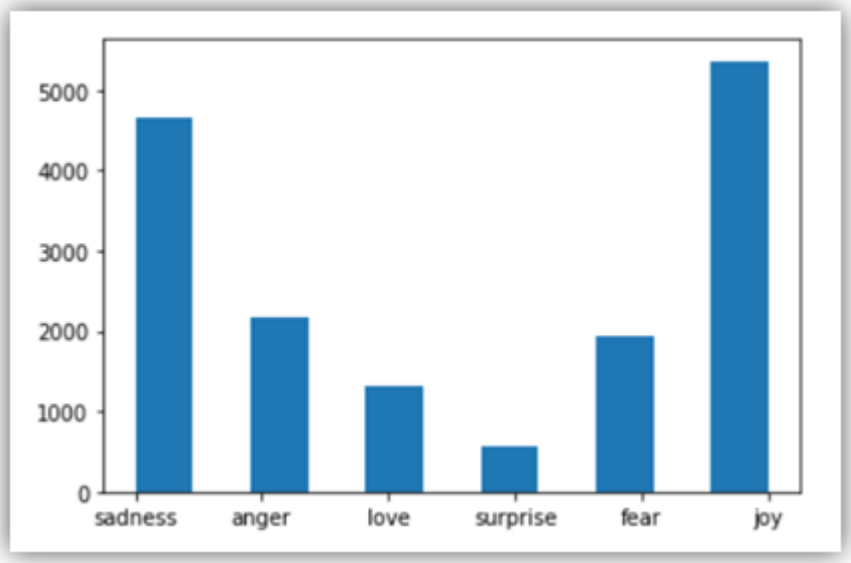
```
padded_train_sequences[10]

array([  2,   3,  14,   2,  21,   5,  80,   6, 733,   2,  93, 544, 304,
        84,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0], dtype=int32)
```

Preparing the Labels

There is a need of different numeric values for the different classes for multi class classifications. The classes are created using the labels from the training set. The six classes which represent the different emotions are - anger, joy, love, surprise, fear and sadness.

A histogram is plotted to see the number of tweets for the different classes.



Two dictionaries are created to convert the names of the classes to their corresponding numeric values. A lambda function is also created to convert the labels of the tweets of the training set to numeric representations.

Creating the Model

A sequential model is created using keras. Recurrent Neural Network (RNN) is a deep learning algorithm that is specialized for sequential data. In a RNN the neural network gains information from the previous step in a loop. The output of one unit goes into the next one and the information is passed.

But RNNs are not good for training large datasets. During the training of RNN, the information goes in loop again and again which results in very large updates to neural network model weights which lead to the accumulation of error gradients during the update and the network becomes unstable. At an extreme, the values of weights can become so large as to overflow and result in NaN values. The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1 or vanishing occurs if the values are less than 1.

To overcome this problem Long Short-Term Memory is used. LSTM can capture long-range dependencies. It can have memory about previous inputs for extended time durations. There are 3 gates in an LSTM cell – Forget, Input and Output Gate.

• Forget Gate: Forget gate removes the information that is no longer useful in the cell state.

• Input Gate: Additional useful information to the cell state is added by input gate.

• Output Gate: Additional useful information to the cell state is added by output gate.

Memory manipulations in LSTM are done using these gates. Long short-term memory (LSTM) utilizes gates to control the gradient propagation in the recurrent network’s memory. This gating mechanism of LSTM has allowed the network to learn the conditions for when to forget, ignore, or keep information in the memory cell.

The fist layer of the model is Embedding layer. Its input dimension is 10,000 (most commonly used words in the dataset) and output dimension is 16 which will be the size of the output vectors from this layer for each word. The input length of sequence is going to be the maximum length which is 50. LSTM preserves information from inputs that has already passed through it using the hidden state. Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past.

Bidirectional LSTM will run the inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backwards information from the future is preserved and using the two hidden states combined it is able in any point in time to preserve information from both past and future.

The second is a bidirectional LSTM layer. This means that the contents from the LSTM layer can go for both left to right and right to left. Its 20 cells (each cell has its own inputs, outputs and memory) are used and return sequence is set to true which means that every time there will be an output which will be fed into another bidirectional LSTM layer it is sent as a sequence rather than a single value of each input so that the subsequent LSTM layer can have the required input.

The final layer will be a Dense layer with 6 units for the six classes present and the activation is set to softmax which returns a probability distribution over the target classes.

The model is compiled with loss set to ‘sparse_categorical_crossentropy’ as it is used for multi class classification problems as the classes are not one-hot encoded (for binary classes). The optimizer used is ‘adam’ as it is really efficient for working with large datasets. The training metrics used is accuracy which calculates how often the predictions are equal to the actual labels. The model summary is generated.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 16)            160000
_________________________________________________________________
bidirectional (Bidirectional (None, 50, 40)            5920
_________________________________________________________________
bidirectional_1 (Bidirection (None, 40)                9760
_________________________________________________________________
dense (Dense)                (None, 6)                 246
=================================================================
Total params: 175,926
Trainable params: 175,926
Non-trainable params: 0
```

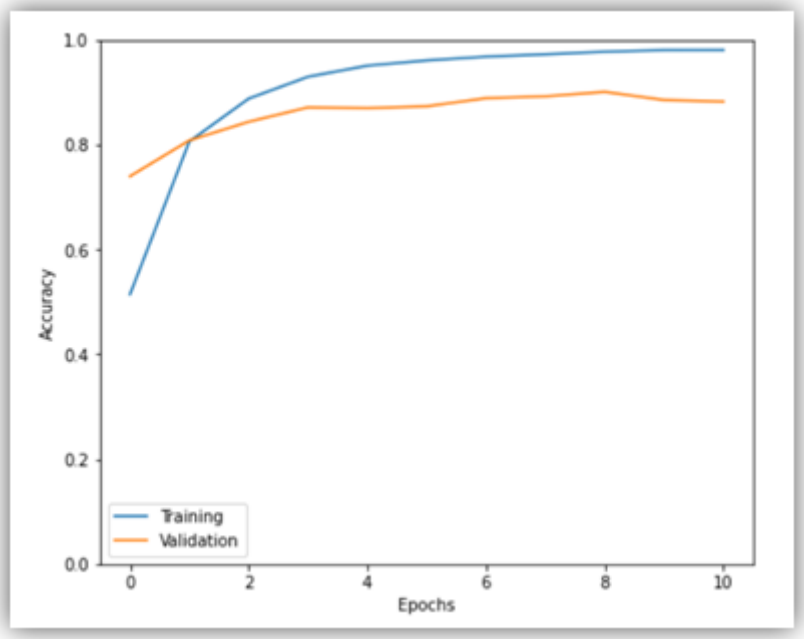
Training the Model

The validation set is prepared and its sequences are generated. Its labels are also converted to their corresponding numerical representation. The model is then trained for 15 epochs. The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset. An early stopping callback is also set which stops the training if the model does not see any improvement in the validation accuracy for over 2 epochs. The model training to be completed takes only about 4 minutes as the Jupyter notebook service is hosted on Google Colab which is using a GPU for accelerated computation.

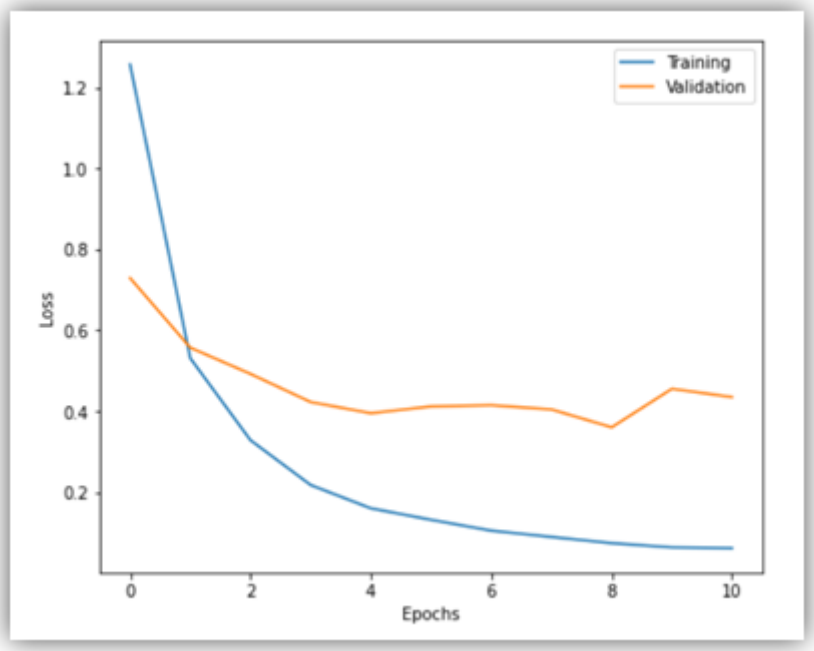
```
h = model.fit(
    padded_train_sequences, train_labels,
    validation_data=(val_sequences, val_labels),
    epochs=15,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2)
    ]
)
```

Evaluating the Model

Plots are generated for the accuracy and loss for the training and validation set over epochs.



Accuracy per epoch plot



Loss per epoch plot

The training accuracy increased consistently and the validation accuracy plateaued and that's when the training stopped. The training and validation loss are both decreasing gradually.

The test set is also prepared and the model is evaluated over it. Some predictions are also checked manually from the test set.

```
eval = model.evaluate(test_sequences, test_labels)
```

```
63/63 [==============================] - 0s 6ms/step - loss: 0.4190 - accuracy: 0.8880
```

The model achieves 88.80% accuracy on the test set which is very similar to the accuracy achieved on the validation dataset.

Results

Some predictions are checked manually from the test set against the actual class label. The tweet with its actual and predicted emotion are printed. About 9 out of every 10 predictions are correct which matches with the model accuracy rate.

```
Tweet: i left feeling absoloutely devastated
Actual Emotion: sadness
Predicted Emotion: sadness
```

```
Tweet: im sorry i feel so uncertain about it
Actual Emotion: fear
Predicted Emotion: fear
```

```
Tweet: i still sit back and feel amazed by the whole thing
Actual Emotion: surprise
Predicted Emotion: surprise
```

```
Tweet: i feel useful and valued and that is fundamental for me
Actual Emotion: joy
Predicted Emotion: joy
```

```
Tweet: i am feeling outraged it shows everywhere
Actual Emotion: anger
Predicted Emotion: sadness
```

CONCLUSION

In this project, a RNN model is constructed to recognize the emotions in tweets. The Model produces an accuracy rate of about 89%.

	Training	Validation
Accuracy	98.50%	89.35%
Loss	0.0538	0.4302

All the predictions are also evaluated against all the ground truths using the test set. A confusion matrix is generated for the test labels in the against the actual classes.