Report

Natural Language Processing, or NLP, is a branch of computer science, artificial intelligence, and linguistics that studies how language is used by computers and by humans. NLP aims to make it possible for computers to comprehend, interpret, and produce human language. This may be achieved by a variety of techniques, including sentiment analysis, text summarization, speech recognition, and language translation. NLP employs a variety of methods, including machine learning algorithms, named entity recognition, tokenization, part-of-speech tagging, syntactic and semantic analysis, and named entity recognition, to accomplish these goals. Numerous sectors, including customer service, healthcare, education, finance, and more, have used natural language processing (NLP).

NLP works with text data and necessitates a number of pre-processing actions to get the data ready for machine learning algorithms to use. Cleaning and converting unstructured text data into a format that models can use is known as data preparation. The normalizing of text is one of the most important preprocessing stages. Slang and harmful terms are common in text data, which might impair model performance. In order to standardize the text in this lab, the normalization process entailed eliminating shortcuts and various forms of harmful phrases. Eliminating special characters from the text is another crucial step. Special characters can affect the model's accuracy since they are frequently out of context. Furthermore, we eliminated repeating letters from the text to make it better because they can be unnecessary and have no value for the model. All text was changed to lowercase in order to guarantee uniformity throughout. By doing this, any confusion that would result from the model having to distinguish between words in uppercase and lowercase is eliminated. In addition to being context in-relevant, numbers can also negatively impact model performance. As a result, we eliminated the numbers from the text and the spaces between sentences.

In order to guarantee that the text data is consistent and free of superfluous characters, it is crucial to exclude non-English characters. We may increase model accuracy and make sure the model is trained on relevant data by preparing text data in this way. Apart from the preparation procedures previously indicated, further measures were implemented to get the text data ready for analysis. Lemmatization was used to reduce words to their most basic form, which can aid in lowering the feature space's dimensionality and enhancing the model's accuracy. Stop words were eliminated from the text data since they add nothing to the analysis and may make the feature space less dimensional. A maximum of 200 words could be used as padding, 20% of the data could be used for validation, and there could be no more words in the vocabulary than that. These were some of the parameters used for the analysis. Furthermore, a 300-dimensional Fast Text word embedding

model was employed. For this investigation, the pre-trained word embedding model wiki-news-300d-1M.vec was chosen. The Fast Text word embedding model was used to prepare the text data for analysis by implementing these preprocessing processes and choosing suitable analysis settings.

To categorize the preprocessed text data, an ensemble of CNN and LSTM was employed in the modeling part. There were many versions of the LSTM model, with output counts ranging from 40 to 60 and dense output counts from 30 to 50. The model was trained using the Adam optimizer for two epochs using a ReLU activation function, a sigmoid activation function for the final layer, and a binary cross-entropy loss function. The batch size was set to 32. To avoid overfitting, dropout levels between 0.1 and 0.2 were employed. In a similar vein, the CNN model employed a batch size of 32 and had 64 filter and 3 kernel sizes. Using the ReLU activation function, sigmoid activation function for the final layer, and binary cross-entropy loss function, the model was trained for two epochs using the Adam optimizer. There was also usage of a pool size of three and dropout levels of 0.1 and 0.2. Using various combinations of the aforementioned hyperparameters, the Talos library was utilized to determine the models' optimal hyperparameters.

Hyperparameters: By experimenting with various combinations, we were able to determine the ideal hyperparameters for our model using the Talos library. These are the optimal LSTM hyperparameters that we discovered:.

1. There are 40 LSTM units or cells in per layer.
2. The drip out is 0.1
3. There are 30 neurons in the FCN.
4. Sigmoid and relu activation in the output layers
5. "binary_crossentropy" is the loss function.

Final Model Summary:-

```
model_2.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200)]             0

 embeddings (Embedding)      (None, 200, 300)          44675400

 lstm_layer (LSTM)           (None, 200, 50)           70200

 conv1d (Conv1D)             (None, 200, 64)           9664

 max_pooling1d (MaxPooling1D  (None, 66, 64)           0
 )

 global_max_pooling1d_1 (Glo  (None, 64)               0
 balMaxPooling1D)

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dense_2 (Dense)             (None, 40)                2600

 dropout_2 (Dropout)         (None, 40)                0

 dense_3 (Dense)             (None, 30)                1230

 dropout_3 (Dropout)         (None, 30)                0

 dense_4 (Dense)             (None, 6)                 186

=================================================================
Total params: 44,759,536
Trainable params: 84,008
Non-trainable params: 44,675,528
_________________________________________________________________
```

The model was trained using an ensemble of CNN and LSTM models on the preprocessed and tokenized data for two epochs, as per the model summary supplied. The Talos library was utilized to optimize the model and determine the optimal hyperparameters. The 'adam' optimizer,'relu' activation function,'sigmoid' last activation function, 'binary_crossentropy' loss function, 'batch size of 32,' and dropout values of 0.1 or 0.2 are the selected hyperparameters for the LSTM. The model is functioning effectively on the validation dataset, as evidenced by the validation accuracy of 99.41% during the training phase. As the number of epochs increases, it was discovered that the training and validation losses decreased, suggesting that the model is learning and enhancing over time. As the number of epochs grew, so did the training accuracy, suggesting that the model is becoming more adept at predicting the sentiment labels on the training dataset. The model is not overfitting on the training dataset, either, since the validation accuracy held steady over the course of the epochs.

Lastly, the test.csv file and the labels from the test_labels.csv file were used to test the model. The model is functioning well on the test dataset and is able to generalize to new data, as seen by the test accuracy score of 88.54%.

```
test_sequences = tokenizer.texts_to_sequences(merged_data['comment_text'])
X_test = pad_sequences(test_sequences, maxlen=maxpadlen)  # Extract the actual labels from the combined data. InfoFrame
y_test = merged_data[["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]].values
y_pred_proba = model_2.predict(X_test)  # Use the learned LSTM model to predict the labels.
y_pred = y_pred_proba > 0.5 # The threshold value can be changed to suit your needs.
```

```
2000/2000 [==============================] - 8s 4ms/step
```

```
from sklearn.metrics import f1_score, accuracy_score  # Compute evaluation measures, such as accuracy, F1 score, etc.
f1_micro = f1_score(y_test, y_pred, average='micro')
accuracy = accuracy_score(y_test, y_pred)
print(f"F1 micro score: {f1_micro}")
print(f"Accuracy: {accuracy}")
```

```
F1 micro score: 0.0113801605867105
Accuracy: 0.885429366344681
```

Important Information:

- The model was given two epochs to run.
- To complete this work, we used the Colab Pro environment with the GPU runtime and HIGH RAM option.

Several crashes occurred when using normal RAM during GPU runtime.

MODEL PERFORMANCE :

```
toxicity_level('Hello, How are you?')
```

```
1/1 [==============================] - 0s 42ms/step
Toxicity levels for 'Hello, How are you?':
Toxic:         1%
Severe Toxic:  0%
Obscene:       0%
Threat:        0%
Insult:        0%
Identity Hate: 0%
```

```
toxicity_level('get the fuck away from me @sshole!!')
```

```
1/1 [==============================] - 0s 26ms/step
Toxicity levels for 'get the fuck away from me @sshole!!':
Toxic:         98%
Severe Toxic:  10%
Obscene:       93%
Threat:        3%
Insult:        63%
Identity Hate: 1%
```

```
toxicity_level("I can't believe how rude and obnoxious that person was to me")
```

```
1/1 [==============================] - 0s 32ms/step
Toxicity levels for 'I can't believe how rude and obnoxious that person was to me':
Toxic:         3%
Severe Toxic:  0%
Obscene:       0%
Threat:        0%
Insult:        0%
Identity Hate: 0%
```

The model we developed provided the specific percentages of comment categorization for every class, as can be seen above.