**REPORT\_PART2 : LAB2**

**Pair Programming Group 19 - Harika Boyina, Divya Neelamegam**

**Introduction:**

Developed a Bidirectional-LSTM model capable of classifying textual comments into toxic and non-toxic categories. Such classification is crucial for maintaining the quality of discourse in online platforms, ensuring that they remain safe and inclusive. We use a dataset from the Jigsaw Toxic Comment Classification Challenge, which includes a variety of comments labeled for various forms of toxicity such as threats, obscenity, insults, and identity-based hate.

**Data Preprocessing:**

Identified and removed rows in the dataset where all toxicity labels are marked as -1, indicating unusable data. This preprocessing step cleans the dataset, ensuring that the model trains and evaluates only on valid, properly labeled examples. Fills the missing values in the comment text columns of the training and testing datasets with the string "fillna," and converts all text to lowercase.

Implemented a comprehensive text cleaning pipeline essential for preprocessing text\_comments of the dataset. This pipeline employs a series of well-defined techniques aimed at enhancing the dataset's quality and uniformity, which are critical for effective model training. Below are the key preprocessing techniques used:

1. Normalization: All text is converted to lowercase to maintain consistency and reduce the complexity of the dataset.
2. Special Character Removal: Specific encoding artifacts and special characters are replaced or removed to clean the text further.
3. Punctuation and Numeric Removal: Punctuation and numeric values are stripped away to focus on linguistic content.
4. Whitespace Management: Excess whitespace is eliminated, ensuring that tokens are uniformly separated by single spaces.
5. Stopword Removal: Commonly used words that offer little value in understanding the sentiment or thematic content of the comments are removed.
6. Stemming and Lemmatization: Words are reduced to their base or root form through stemming and lemmatization, making the text more uniform and reducing the vocabulary size the model needs to handle.

Vocabulary class designed to manage a mapping between words and unique indices in a dataset. It initializes with a frequency count of words (word counter) with an optional minimum frequency as 1. Words that meet or exceed this frequency threshold are added to the vocabulary. The class provides methods to convert words to their corresponding indices (lookup\_indices). It uses <unk> as a placeholder for unknown words not present in the vocabulary.

**Data preparation:**

Preparing text data for input into a LSTM model, specifically designed for tasks involving natural language processing. The steps are as follows:

1. Tokenization: Utilizes the word\_tokenize function to split the comments from the training and test datasets into individual words.
2. Vocabulary Construction: Builds a vocabulary from the tokens in the training set, using a frequency counter to ensure each word meets a minimum frequency threshold of one occurrence.
3. Token to Index Conversion: Converts the tokenized text into sequences of indices based on the vocabulary, where each word is represented by a unique integer.
4. Sequence Padding: Ensures that all sequences have a uniform length of 100 by padding shorter sequences with zeros. This step is crucial for batch processing in neural network models. There are 208757 unique vocabulary words.

Prepares datasets for training, validation, and testing of a comment analysis using NLP model. DataLoader objects for each dataset with 512 batch size and collation function was used for facilitating efficient batch processing during model training and evaluation.

### **Model Architechture:**

Splits the data into training and validation sets using an 80-20 ratio and a specified random seed. The model includes an embedding layer, one bidirectional LSTM layer followed by one unidirectional LSTM layer, and two fully connected layers for classification. The activation functions used are ReLU and sigmoid. The model is trained using the Adamax optimizer and Binary Cross-Entropy Loss. The Layers in this BiLTSM model are given below:

1. Embedding layer (Size of vocabulary to 128)
2. One BiLTSM (128 to 512)
3. One LTSM (1024 to 128)
4. Linear Fully connected layer (128 to 16)
5. Linear Fully connected layer (16 to 6)
6. Activation layers - ReLU and sigmoid

**Model Training:**

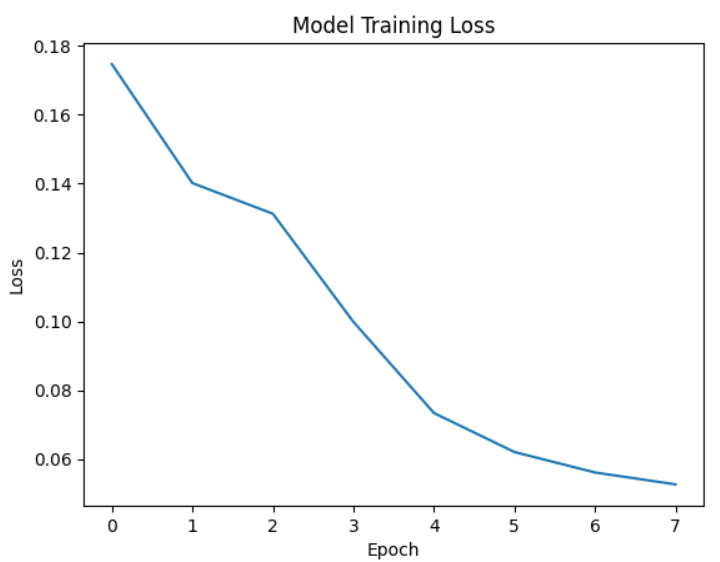
Efficiently trains a neural network model for comment analysis using PyTorch using train\_model function and with essential parameters like the model, data loaders for training and validation, loss function, optimizer, and the number of epochs.

Trains the provided model using the training dataset, evaluating its performance on the validation set. The training process runs for 8 epochs, computing training loss and validation accuracy metrics.

The training process spans 8 epochs, with each epoch displaying the training loss and validation accuracy for batch iterations. After each epoch, the training loss and validation accuracy are printed. After the first epoch, the training loss was 0.1747 and validation accuracy was 0.9629 respectively. The model achieves a progressively improving validation accuracy, peaking at 98.06% after the final epoch and the training loss has also been reduced to 0.0526.

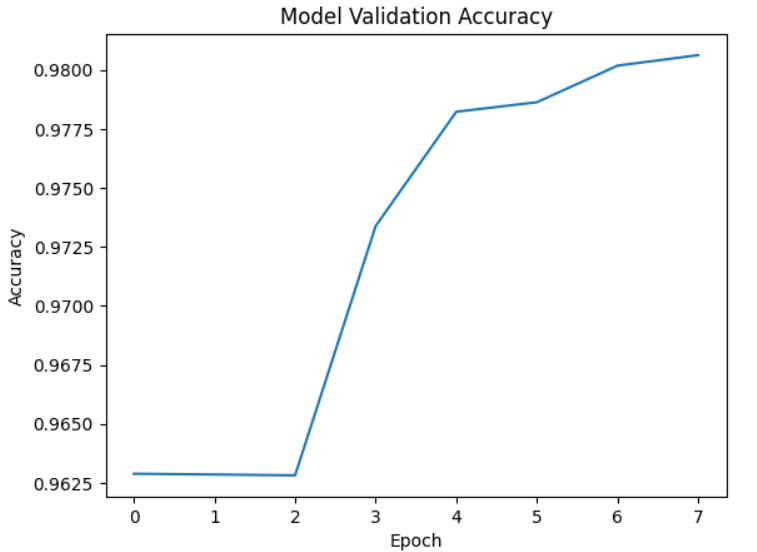
Plotted the model validation accuracy v/s no. of epoch, and model training loss v/s no. of epochs in below graphs.

Graph of Model Training loss v/s number of epochs



With increase in the number of epochs, it is observed that the training loss has been significantly decreasing from the above graph.

Graph of Model Training loss v/s number of epochs



During the initial epoch the validation accuracy was 0.9629, and training loss was 0.1747. With the increase in the number of epochs, the validation accuracy has increased and train loss has decreased. After the final epoch of training, training loss was 0.0526 and validation accuracy was 0.9806.

**Model evaluation:**

Analyzes the trained model's performance on the test dataset. It generates predictions, computes probabilities, and creates a submission file. Additionally, it calculates average test accuracy and label-wise accuracy, offering insights into the model's classification performance. Below are the results of each label and overall test accuracy.

Test Accuracy in toxic label: 92.4130%

Test Accuracy in severe\_toxic label: 99.4311%

Test Accuracy in obscene label: 96.2221%

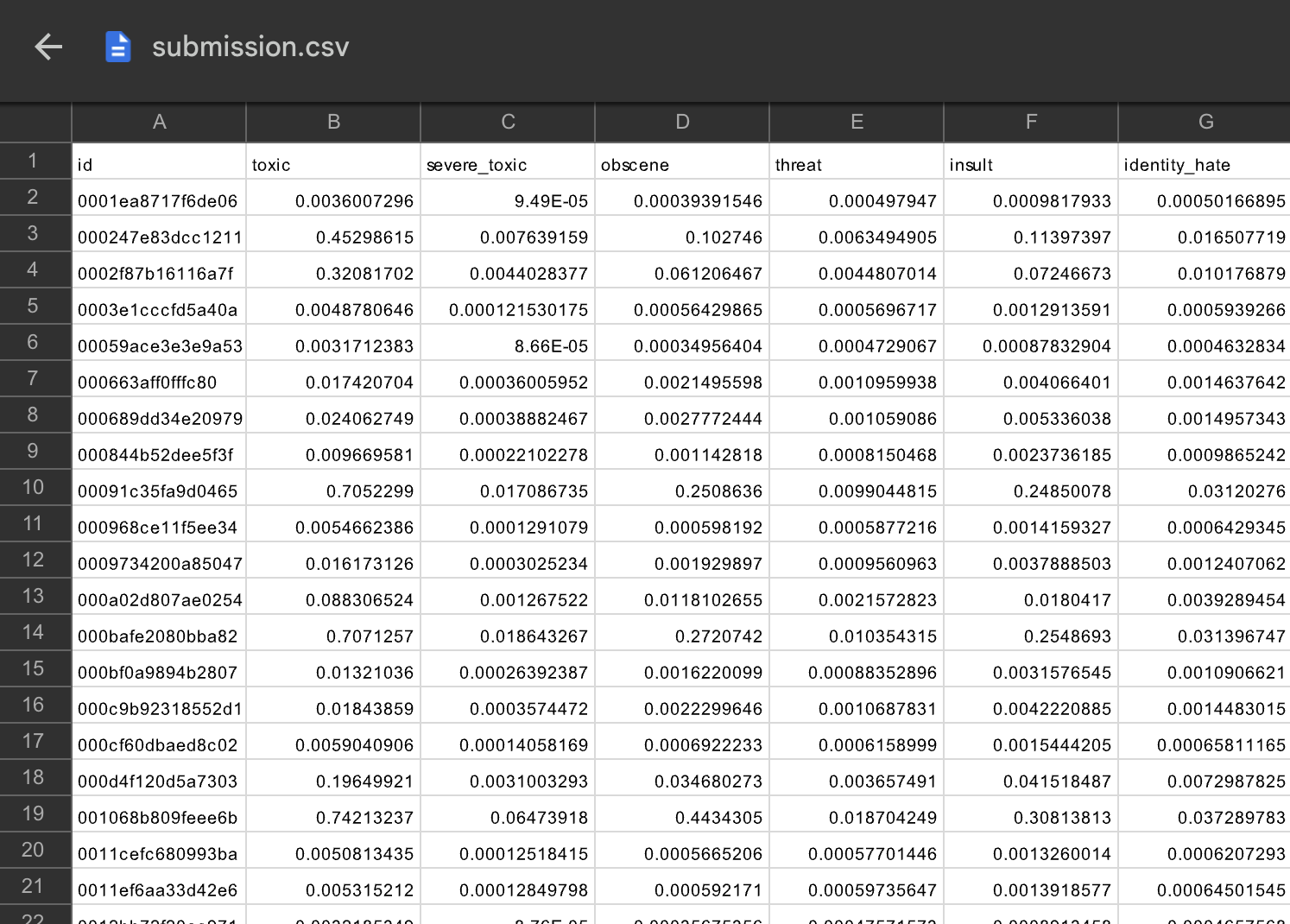
Test Accuracy in threat label: 99.6702%

Test Accuracy in insult label: 95.6626%

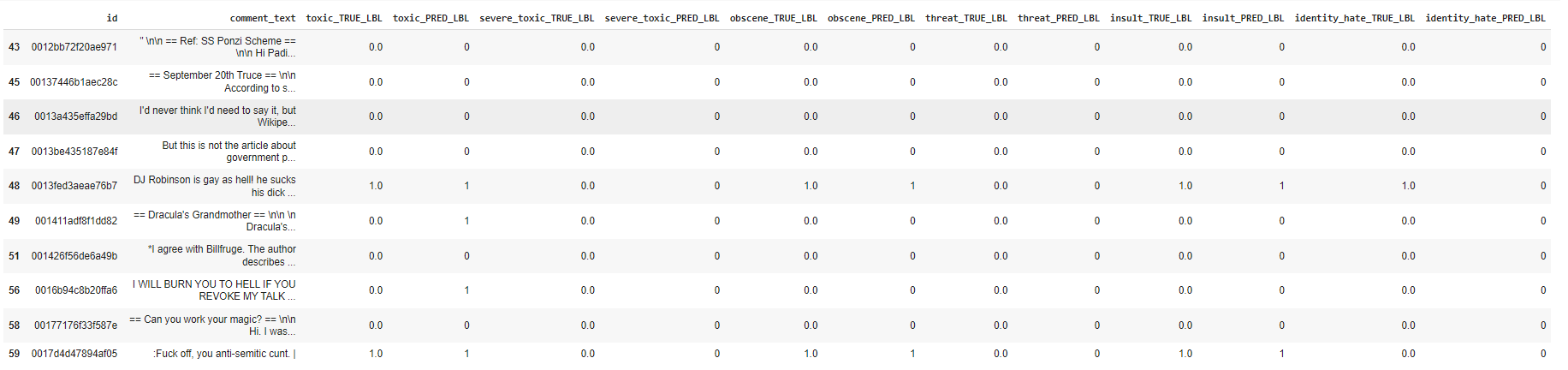
Test Accuracy in identity\_hate label: 98.8871%

Overall Test Accuracy: 97.0477%

**Submission.csv :**



The predicted and true labels of all labels are shown in the jupyter notebook. Displaying the few records in the screenshot below.



The confusion matrix of each labels are displayed in the notebook.

