Project 2: Sentiment Analysis

Introduction:

In an era dominated by vast amounts of textual data, the Sentiment Analysis project emerges as a beacon in the realm of data science, wielding the power to decipher and categorize sentiments embedded within diverse textual expressions. Harnessing ML techniques, this report navigates the intricate landscape of language to construct a predictive model adept at discerning whether a given text exudes positivity, negativity, or neutrality. Drawing inspiration from a meticulously curated dataset, the exploration of sentiment intricacies unfolds, guiding the project towards the nuanced understanding of key variables, text structures, and sentiment labels. Through an intricate blend of preprocessing, exploratory data analysis, and model interpretation, Sentiment Analysis not only unveils the sentiments concealed within textual narratives but also unravels the underlying narrative of emotions, enriching our understanding of the language's emotive tapestry.

About the data:

The data is divided in 2 parts: one is for training and one is for testing. The shape of the training data is *(27481, 10)* and that of the test data is *(4815, 9)*, out of which we have extracted the text and the corresponding sentiments as X_train, X_test, y_train and y_test respectively. Other columns seemed irrelevant when compared with sentiment analysis of the texts.

Data Preprocessing:

In the preprocessing or text cleaning phase, the dataset undergoes a meticulous transformation orchestrated by a cleaning function. Each textual entry is first converted to lowercase, homogenizing the case spectrum for uniform analysis. Subsequently, hyperlinks are systematically expunged, ensuring that extraneous web-related information does not influence sentiment classification. Special characters, mentions, and hashtags are methodically excised, leaving behind a purified corpus for analysis. The use of regular expressions and tokenization facilitates the removal of numerical and non-alphanumeric characters, further refining the text. Leveraging the Snowball Stemmer for English, words are stemmed to their root forms, contributing to the normalization of language. A curated list of stopwords is strategically applied, refining the text to its essential semantic content. The culmination of these steps results in a preprocessed, standardized textual representation conducive to the subsequent stages of sentiment analysis.

```
def textprocessing(text):
    text = str(text)
    text = text.lower() #converting all uppercase letters to lowercase
    text = re.sub(r"https\S+|www\S+|https\S+"," ",text,flags=re.MULTILINE) #removing all links from dataset
    text=re.sub("(\\d|\\W)+"," ",text)
    text = re.sub(r'\@\w+|\#'," ",text) #removing # and @ symbols from dataset
    text = re.sub(r'[^\w\s\`]'," ",text) #removing other symbols like ^ except '
    text_tokens = word_tokenize(text)
    lem = SnowballStemmer("english")
    text = [lem.stem(word) for word in text_tokens if not word in stuff_to_be_removed]
    text1 = " ".join(text)
    return text1
```

Feature Engineering:

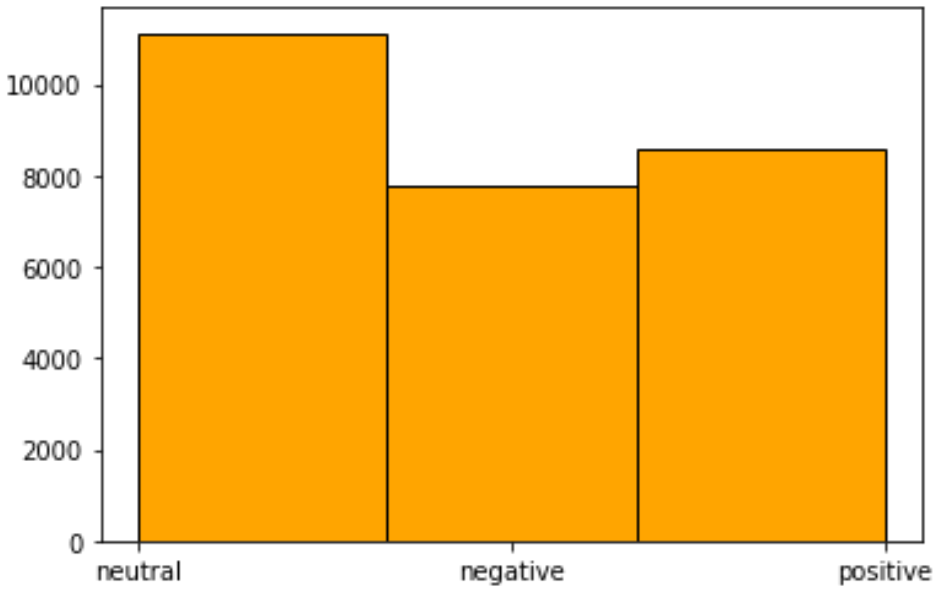
Firstly, we have removed the NaN values from the training and testing cleaned datasets. As few NaN features were present in the training dataset (only 1), hence we decided to go with removing those values and not imputing them. Now since the textual data can be feeded into ML or DL models, we have to convert them into vectors or useful features which can capture semantic relations between words and ultimately which can be fitted into any particular model. We tried out various vectorization techniques like *CountVectorizer*, *TFIDFVectorizer*, *Word2Vec*, etc. But it turned out that ***TFIDF Vectorizer*** turned out to perform best for our dataset where most of the texts are small and thus it is able to capture best relations among words as features.

The TF-IDF transformation code is given below:

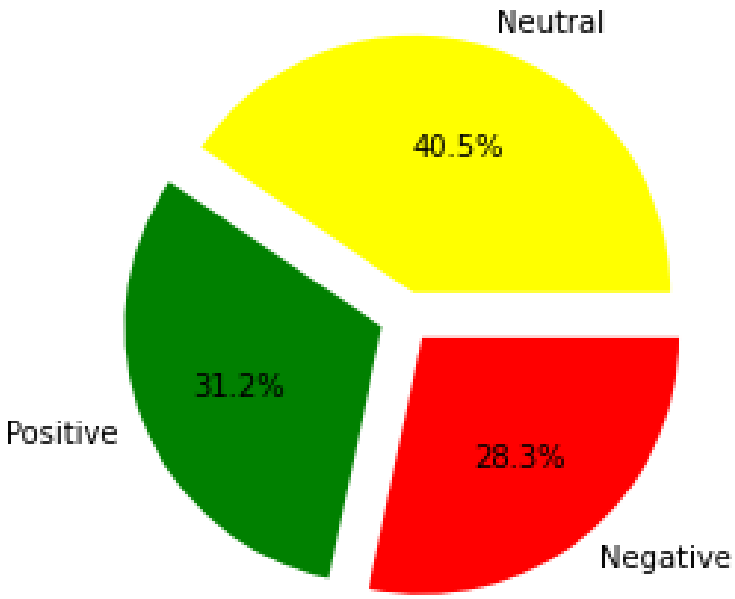
```
## Text vectorization using TF-IDF
tfidf = TfidfVectorizer(max_features=2500)
X_train_tfidf = tfidf.fit_transform(X_train)
X_train_tfidf.shape
```

```
(27480, 2500)
```

Now regarding the labels, there are 3 classes namely *neutral*, *positive* and *negative*, representing the sentiments of the texts. We have simply encoded the labels as 0, 1 and 2. The histogram plot for the distribution of the training dataset labels is shown below:



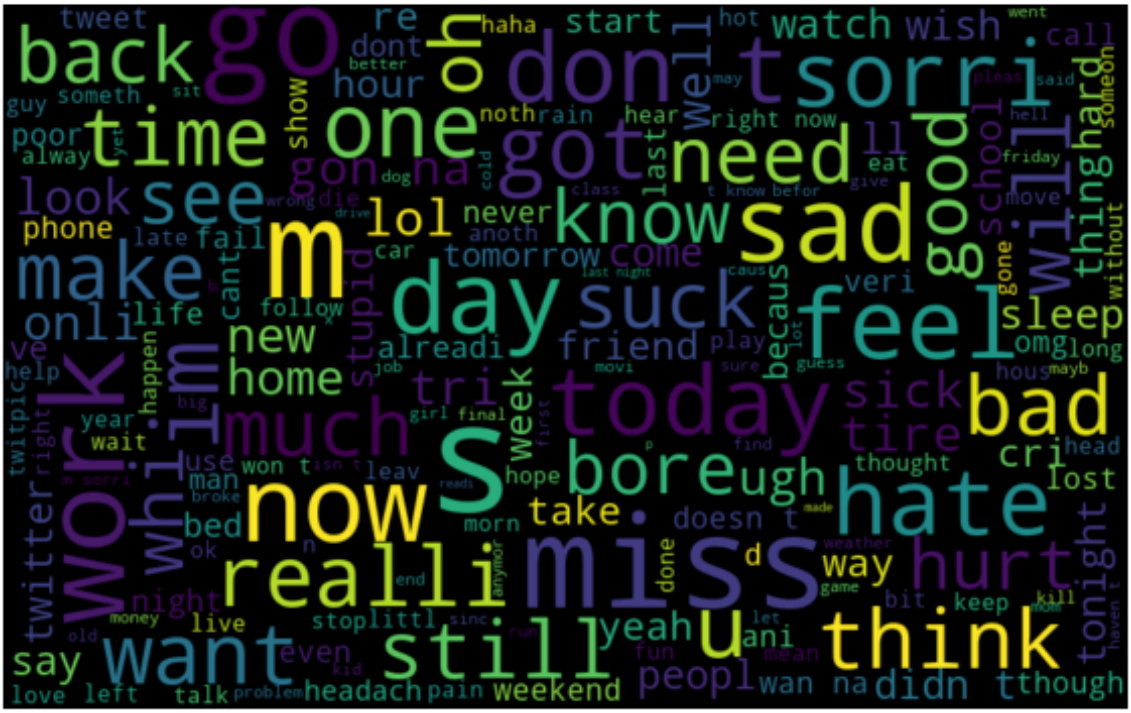
Pie-chart for the same distribution is shown below:



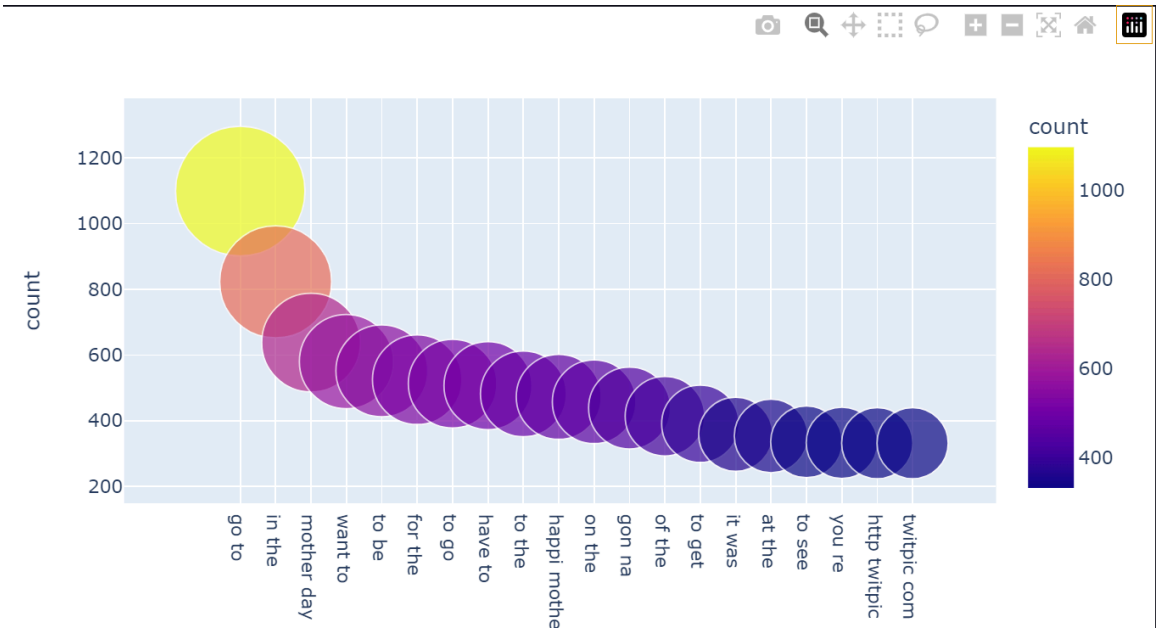
Thus we can say that the training dataset is more or less balanced with 3 distinct labels.
The word cloud for the labels is shown below to better understand the data.

Word cloud for the positive labels

Word cloud for the negative labels

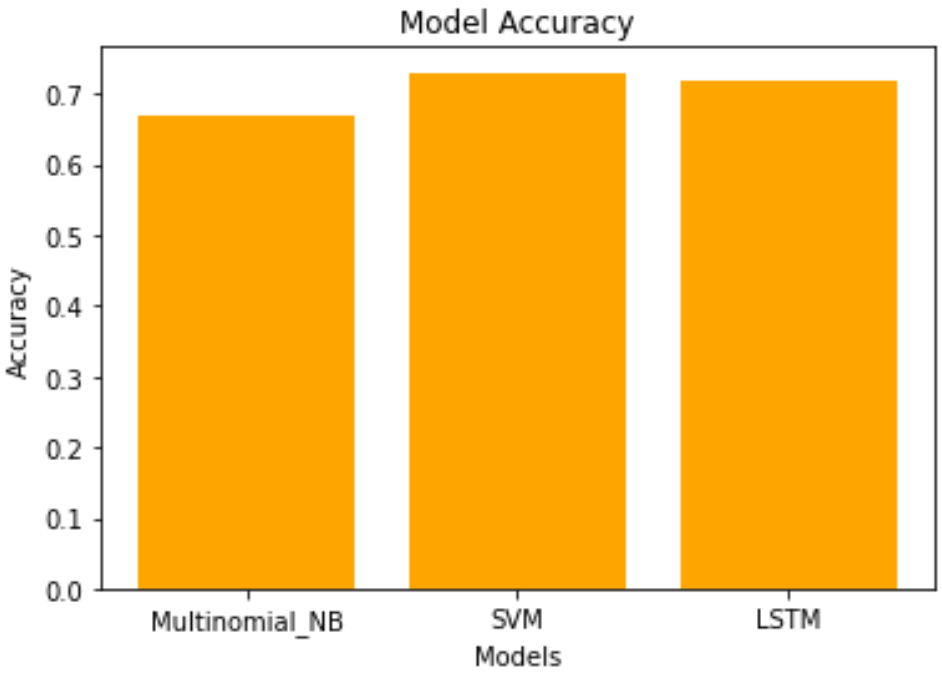


Below is the plot for the top 20 “2-gram” word combinations with their frequencies.



Model Selection:

We have applied various models like *Naive Bayes Classifier, SVM* and *LSTM*. We tested out these models by training them on the entire cleaned training data and tested them on the testing data. Rigorous testing unveiled nuanced performance metrics, with Naive Bayes achieving an accuracy of **0.67**, LSTM closely trailing at **0.72**, and SVM emerging as the frontrunner with a notable accuracy of **0.73** on the test set. The decision to propel SVM to the forefront was underpinned by its superior predictive prowess. To fine-tune and optimize the selected SVM model, an elaborate hyperparameter tuning process was employed, leveraging the exhaustive *GridSearchCV* methodology. This meticulous exploration of hyperparameter space sought to maximize the model's efficacy. Moreover, the implementation of *cross-validation* techniques served as a litmus test for the model's generalization capacity, ensuring resilience against overfitting. This strategic fusion of model testing, performance assessment, and hyperparameter optimization culminated in the discerning selection of SVM as the most adept sentiment analysis model for the given dataset. The histogram of the model accuracies is given below:



The code snippets for the SVM model parameters are also attached:

```
## SVM
classifier3 = svm.SVC(kernel='linear', C=1, gamma=1)
classifier3.fit(X_train_tfidf, Y_train)
## Testing our model
Y_pred = classifier3.predict(X_test_tfidf)
## Classification report
print(classification_report(Y_test, Y_pred))
```

The code for the tokenizer and the LSTM model parameters are attached below:

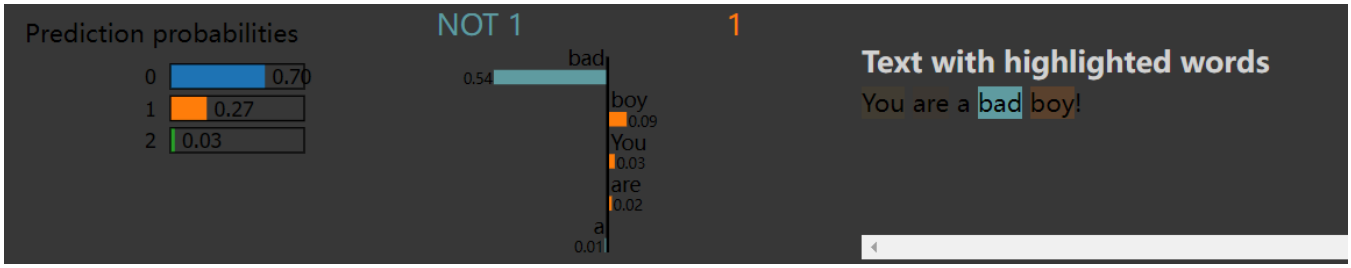
```
## Tokenizing the text
tokenizer = Tokenizer(num_words=1500, split=' ')
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_train = pad_sequences(X_train)
X_train.shape
```

```
## Defining model parameters
embed_dim = 256
lstm_out = 196
batch_size = 32

## Defining the model with gensim vectors as X_train and X_test
model = Sequential()
model.add(Embedding(2500, embed_dim, input_length=X_train.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Model Interpretability:

We know that machines decode the subtleties of human expression, the necessity for model interpretability becomes paramount. It is not merely about predicting sentiments but also comprehending the "why" behind those predictions. Model interpretability acts as the beacon illuminating the black box of algorithms, allowing us to discern the intricate dance of features that shape predictions. Understanding which words or expressions hold the most sway in sentiment determination not only fortifies the trustworthiness of the model but also provides users with valuable insights into the decision-making process. It's akin to unraveling the layers of a conversation, offering a humanized perspective on how the model perceives sentiment nuances. Ultimately, model interpretability bridges the gap between the complex machinations of algorithms and the intuitive grasp of language, fostering a more transparent and relatable interaction between machines and the intricacies of human expression. We have used *LIME (Local Interpretable Model-agnostic Explanations)* here to understand which words or features contribute most to sentiment predictions. A sample test result is interpreted below:

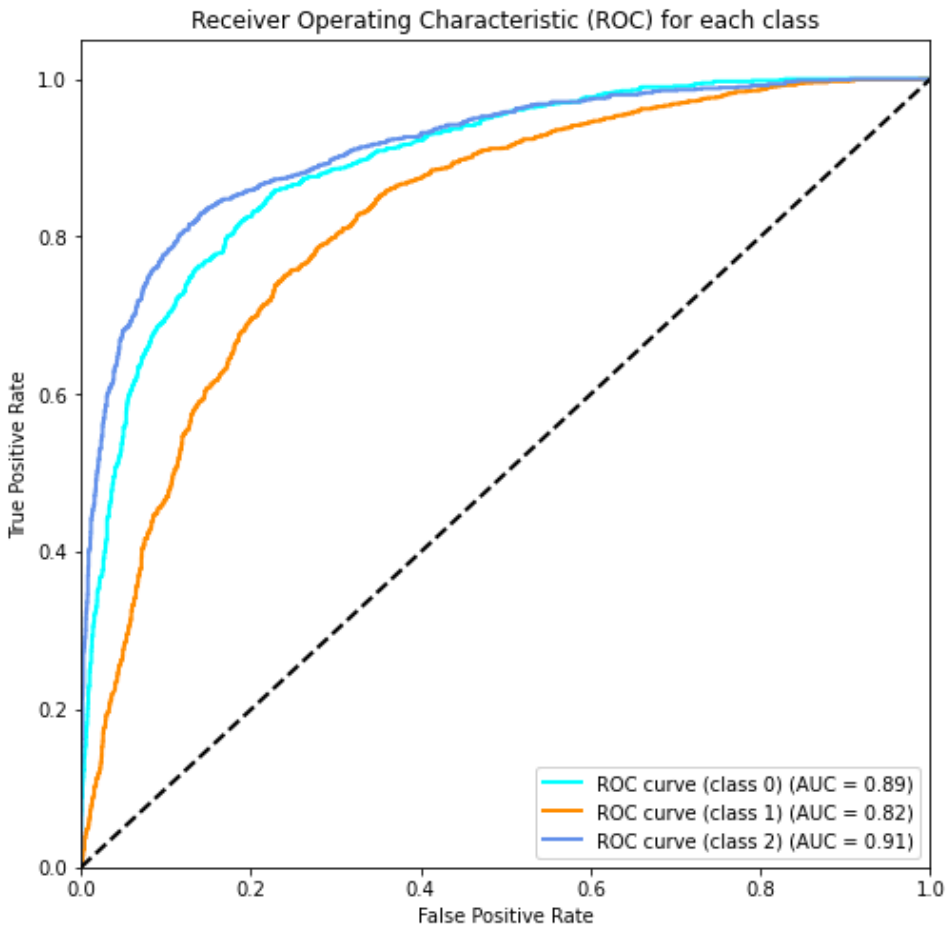


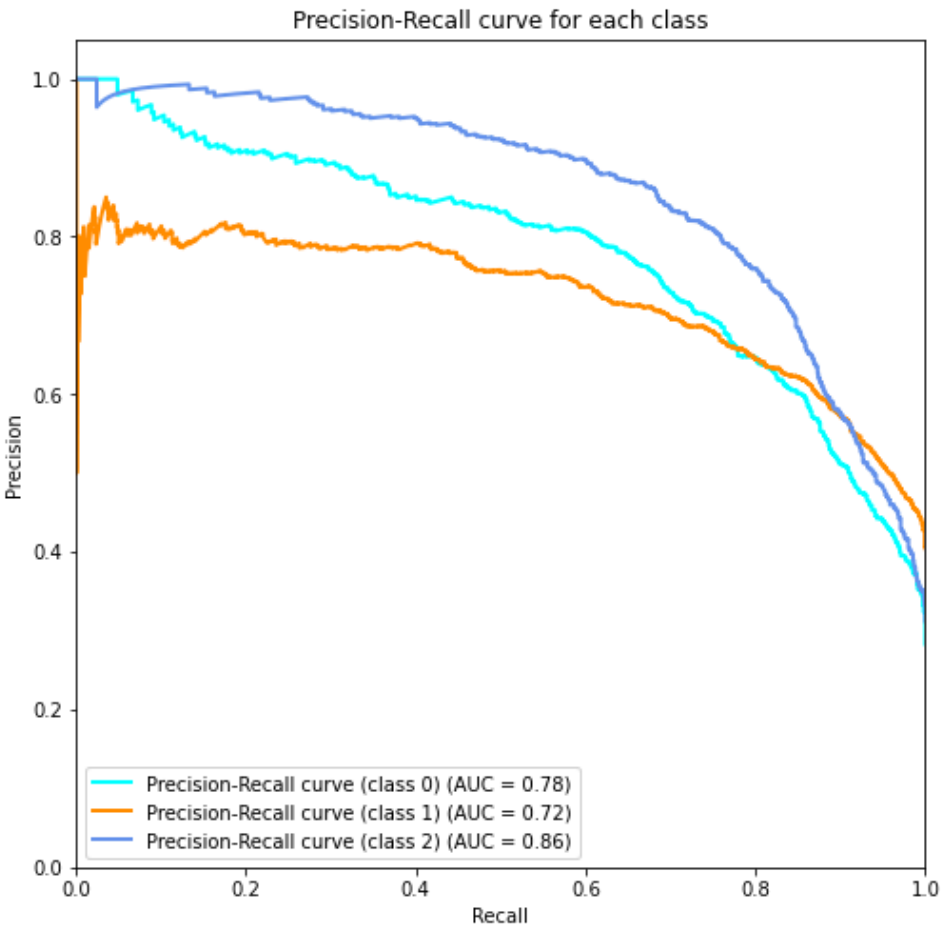
Evaluation Metrics:

Evaluation metrics serve as the guiding compass in the labyrinth of sentiment analysis, unraveling the intricacies of model performance. The Confusion Matrix, akin to a detailed map, not only unveils the model's accuracy but also delineates the nuances of its predictive errors. It distinguishes between true positives, true negatives, false positives, and false negatives, providing a comprehensive snapshot of predictive prowess. Simultaneously, the ROC Curve navigates the delicate balance between true positive rate and false positive rate, offering a nuanced portrayal of a model's discriminatory capabilities across varied thresholds. Concomitantly, the Precision-Recall curve artfully crafts the narrative of precision and recall, elucidating the trade-off between accurate positive predictions and the model's ability to capture all pertinent instances. These metrics, when interwoven, offer a multifaceted lens through which to comprehensively assess and refine the symbiotic dance between predictions and reality in the dynamic domain of sentiment analysis.
The Confusion matrix obtained in our final model is:

```
Confusion Matrix:
[[ 667  303   31]
 [ 175 1111  144]
 [  49  254  800]]
```

It indicates most of the predictions are correct (along the left diagonal line) and some labels are mistakenly predicted by the model, giving it an overall accuracy of **73%**.
The ROC Curves and Precision-Recall Curves are also attached below:


Receiver Operating Characteristic (ROC) for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (class 0) (AUC = 0.89)
ROC curve (class 1) (AUC = 0.82)
ROC curve (class 2) (AUC = 0.91)


Precision-Recall curve for each class
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Precision-Recall curve (class 0) (AUC = 0.78)
Precision-Recall curve (class 1) (AUC = 0.72)
Precision-Recall curve (class 2) (AUC = 0.86)

—X—