# Method

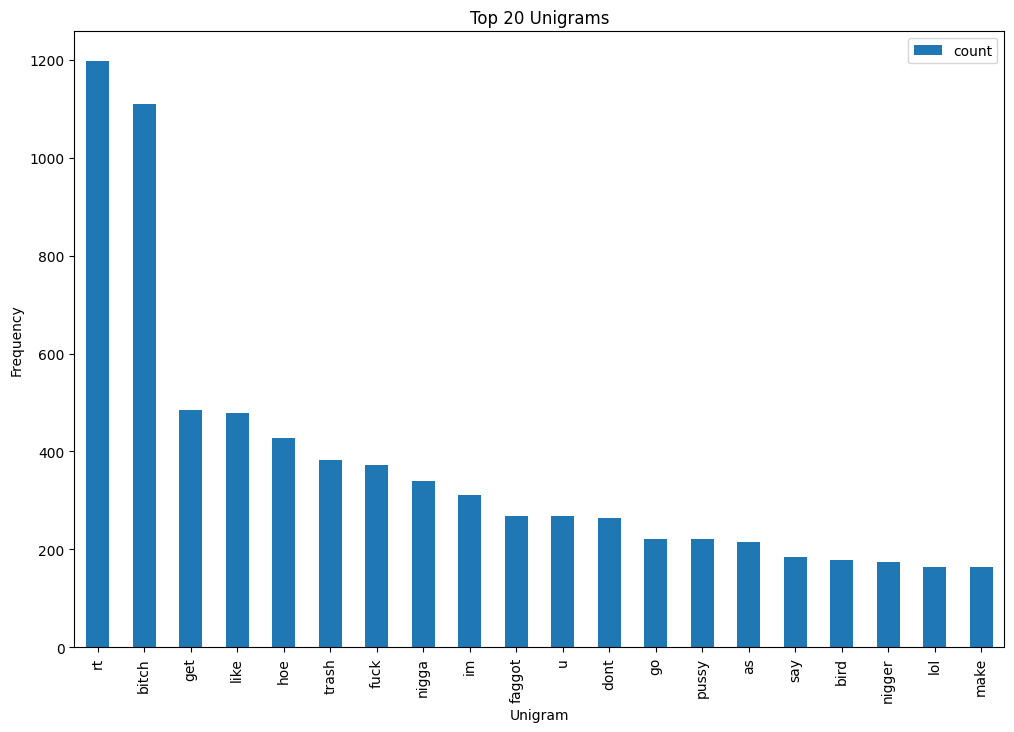
The study was conducted on a highly unbalanced dataset, where the class distribution was identified as 19190 instances for class 1, 4163 instances for class 2, and 1430 instances for class 0. It was observed that due to the class imbalance, even a simplistic model that predicts every instance as class 1 can achieve more than 90% accuracy. However, the study aimed to develop an unbiased model that could perform well on all classes. To address the issue of class imbalance, two basic options were considered, namely minority up sampling and majority down sampling. However, as the dataset consisted of textual data, it was observed that minority up sampling was prone to overfitting. Hence, the majority down sampling approach was implemented. This approach involves randomly selecting instances from the majority class and removing them from the dataset until the distribution of classes becomes more balanced. The downside of this approach is the reduction of data size.

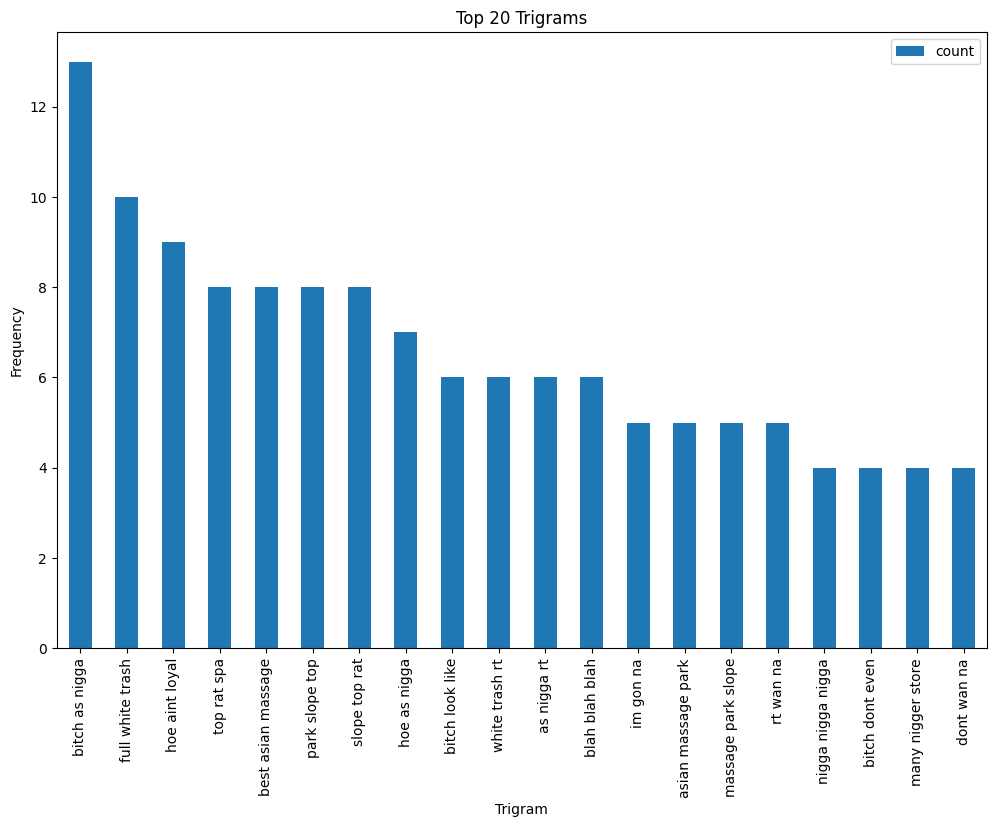
After balancing the dataset, the data was preprocessed using basic Natural Language Processing (NLP) techniques. This involved removing punctuation, converting all letters to lowercase, and removing stop words.

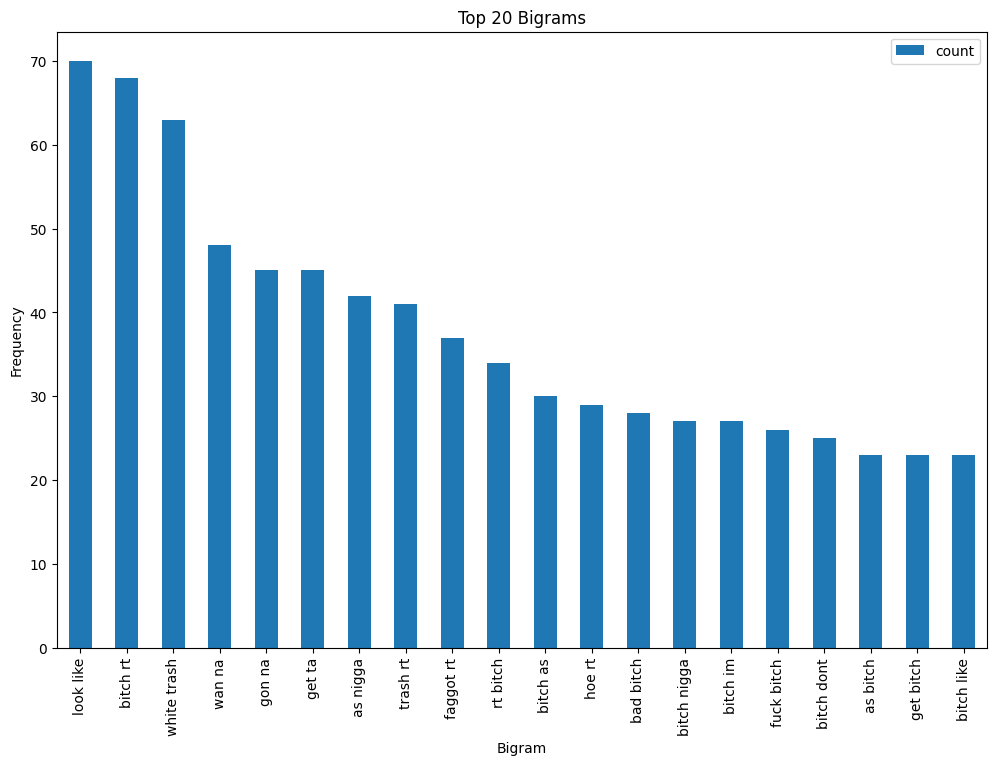
To gain insights into the most common words in the dataset, word clouds and n-grams (unigram, bigram, and trigram) were plotted for both the raw and preprocessed data. The visuals for the raw data revealed that most of the common words were stop words, while for the preprocessed data, the common words were more meaningful.

After text preprocessing



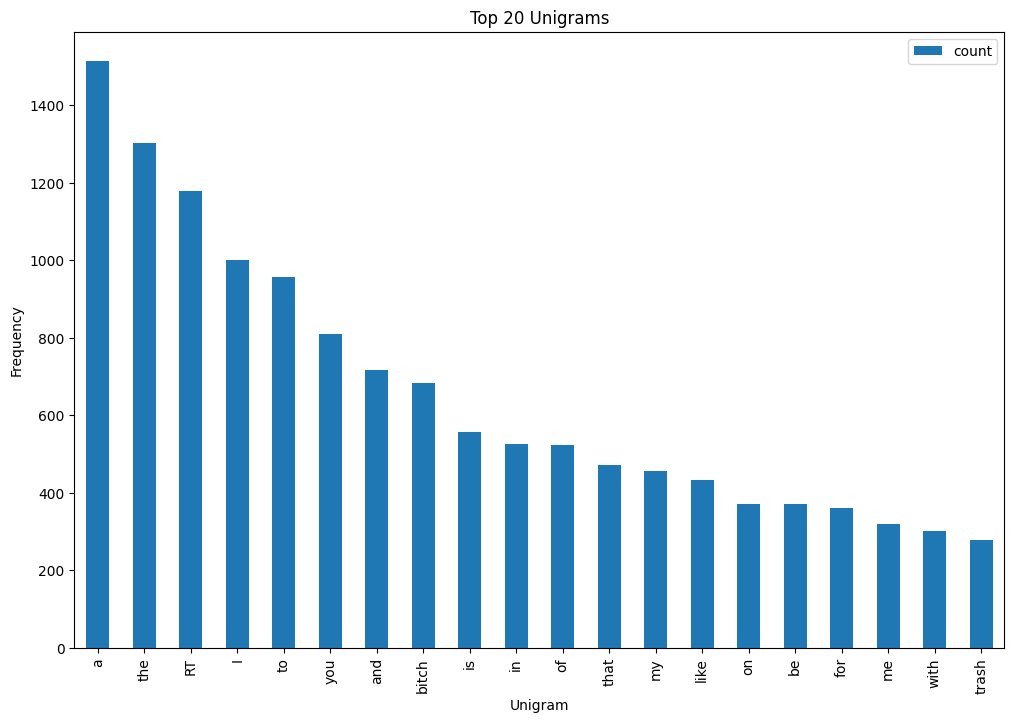


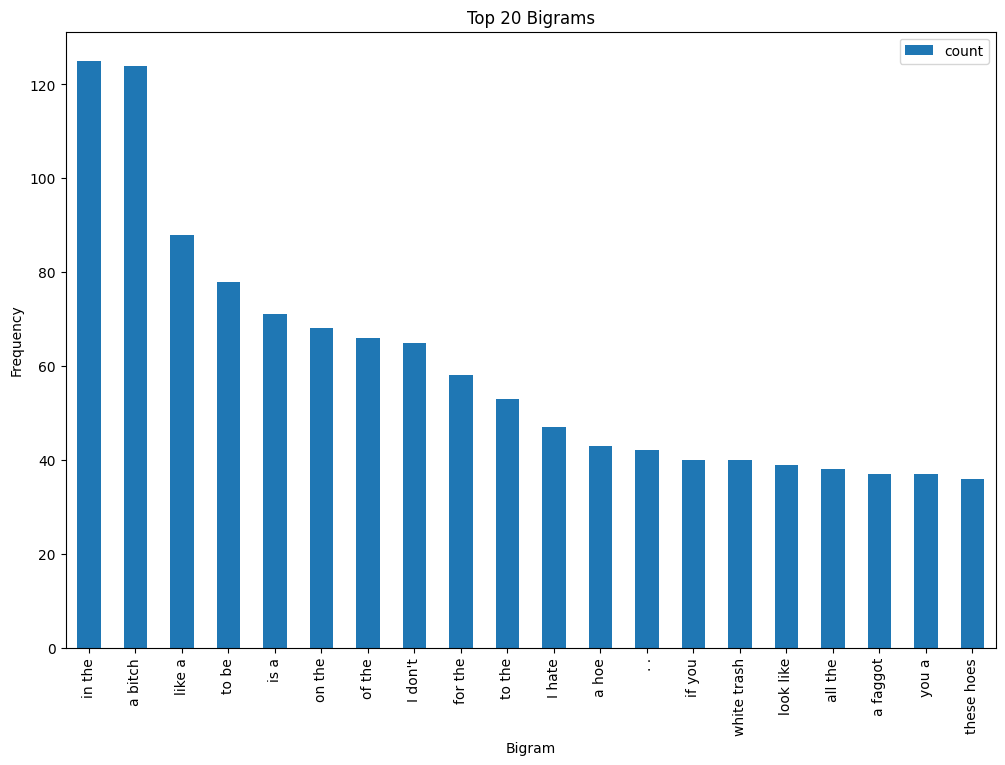


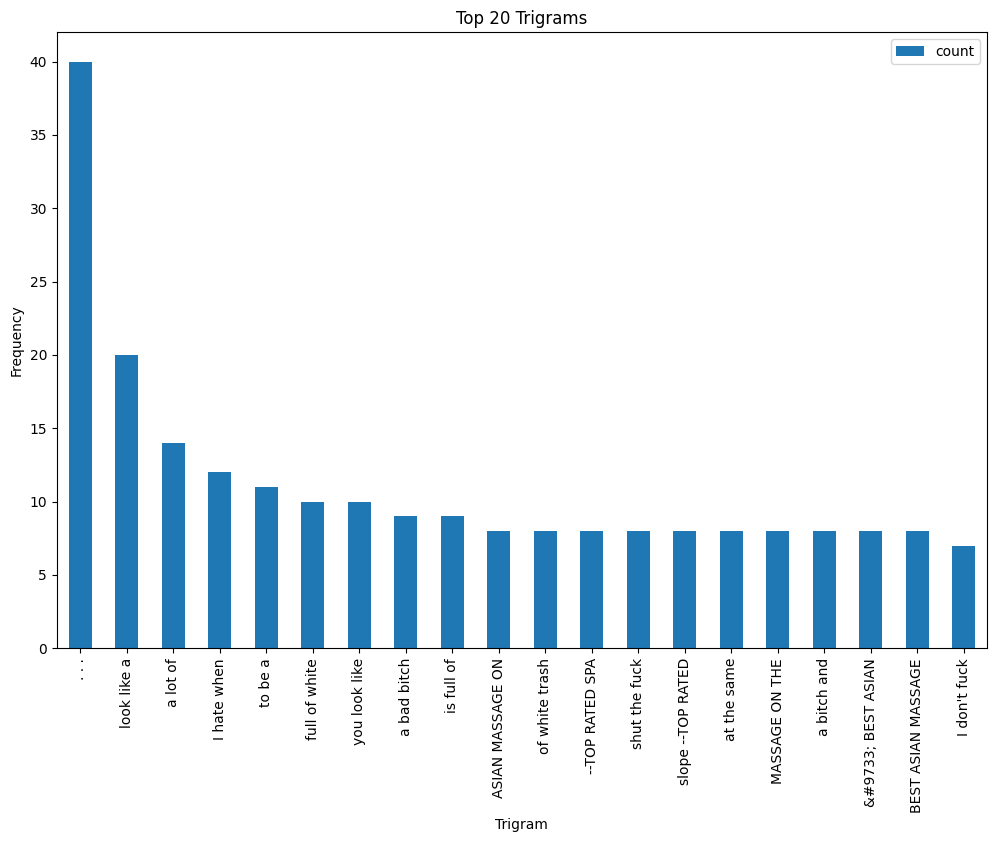


Before Text Preprocessing









# LSTM

The text classification task was first approached using an LSTM-based model. The data was preprocessed by tokenizing the text and padding the sequences to a fixed length. The LSTM architecture consisted of an embedding layer, a LSTM layer with a dropout rate of 0.2, and a fully connected output layer with softmax activation. The model was trained using the Adam optimizer and categorical cross-entropy loss function. The hyper parameters selected for the model were an embedding size of 100, a maximum number of words to keep based on word frequency of 10,000, a maximum sequence length of 100, and a batch size of 32. The model was trained for 20 epochs.

The results of the study showed that the model achieved an accuracy of 0.7097902097902098 on the test set of processed tweets. The precision was 0.7165401741481754, recall was 0.7097902097902098, and F1-score was 0.712048559335627. The confusion matrix showed that the model performed well in predicting the correct class for the majority of the tweets in the validation set, with only a relatively small number of misclassifications.

# Bert

The BERT model is loaded along with the tokenizer from the transformers library. The text data and labels are defined and tokenized using the tokenizer, and the labels are one-hot encoded. The maximum sequence length is set to 128, and the texts are truncated or padded to fit this length. The data is split into training and testing sets. A BERT model is built by defining the input layers, which take the input\_ids and attention\_masks, followed by the BERT model itself. The pooled\_output of the BERT model is passed through a dropout layer and a fully connected output layer with a softmax activation function. The BERT model layers are frozen, and the model is compiled using the Adam optimizer with a learning rate of 2e-5 and a categorical cross-entropy loss function. The model is trained for 20 epochs with a batch size of 32, and the training and validation data are provided to the model along with their respective masks. The model's performance is evaluated on the test set using the evaluate method, which returns the loss and accuracy.

The resulting accuracy, precision, recall, F1 score, and confusion matrix show that the BERT model achieved an accuracy of 0.636, indicating that it performed reasonably well on the task of sentiment analysis. However, these metrics do not provide any information about the comparative performance of BERT with other models.

# Comparison

The LSTM model yielded an accuracy of 0.7097, while the BERT model resulted in an accuracy of 0.6364. This indicates that the LSTM model performed better than the BERT model in predicting the sentiment of tweets. Looking at the precision, recall, and F1 scores, the LSTM model also outperformed the BERT model, with higher values in all three metrics. The confusion matrices also show that the LSTM model had more accurate predictions across all three sentiment classes.

However, it's important to note that the two models have different architectures. The LSTM model is a recurrent neural network that uses sequential information to make predictions, while the BERT model is a transformer-based model that is pre-trained on a large corpus of text data. In addition, the BERT model was trained on the 'bert-base-uncased' model, which is a smaller version of the BERT architecture, while the LSTM model was trained with a custom-built architecture.

The main reason that LSTM outperformed BERT is that the size of the data became small after majority down sample. BERT being an encoder model requires huge amount of data for optimal training which not the case is.

While we also used other models namely Decision Tree, Random Forest and Naïve Bayes, but none of them were able to surpass 35% accuracy.