TWITTER SENTIMENT ANALYSIS

REPORT

Aarya Yogesh Pakhale

2nd Year Undergraduate Student
IIT Kharagpur
Data Analyst Intern: Nexus

INDEX

PROJECT GOAL AND CONTEXT

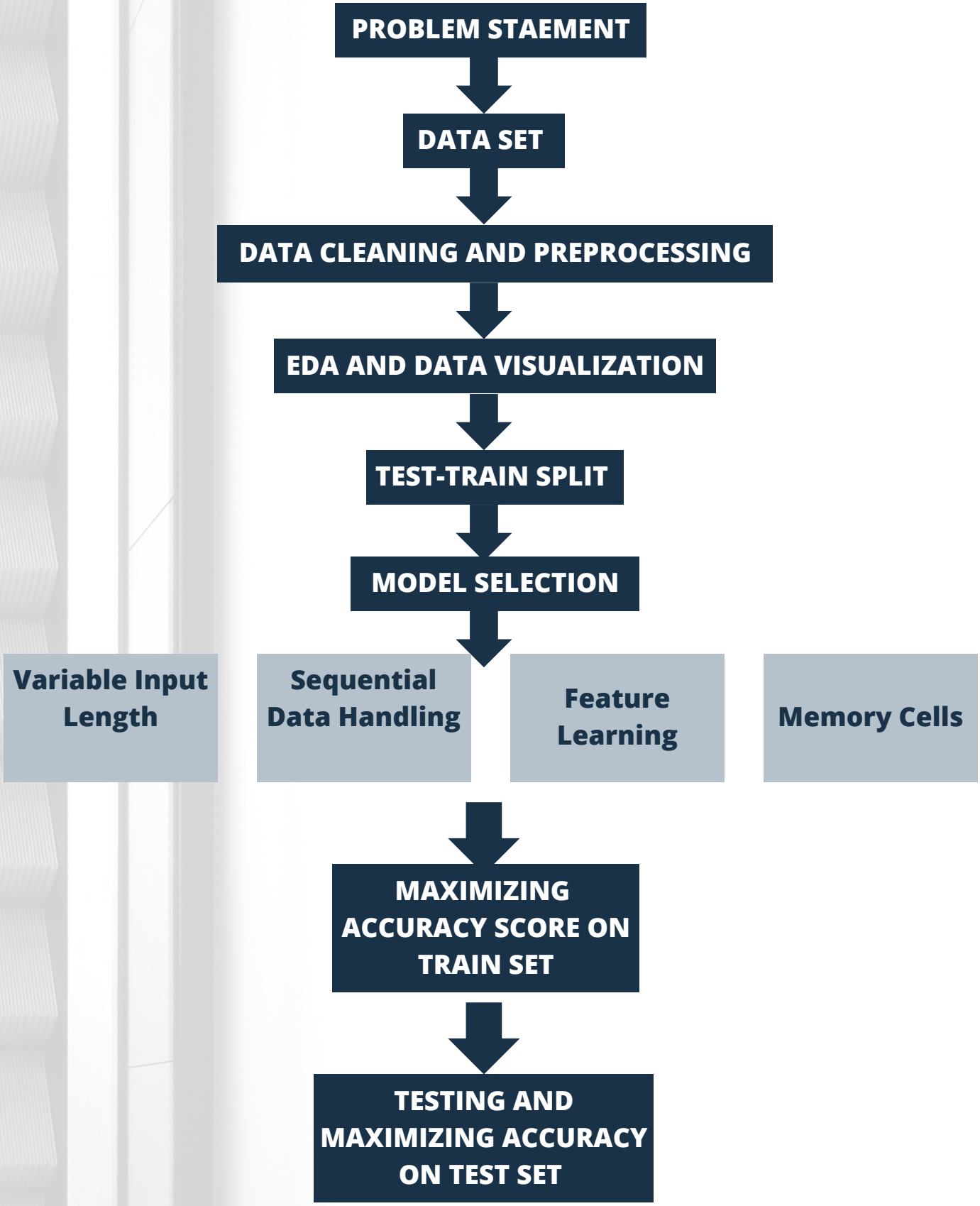
Project Title:

Twitter Sentiment Analysis

Project Overview:

Twitter Sentiment Analysis is a data analytics project that involves analyzing a dataset of tweets to determine the sentiment expressed in each tweet—whether it is positive, negative, or neutral. This project aims to gain insights into public opinions, trends, and sentiments shared on Twitter, utilizing data analytics techniques.

Approach used:



DATA SOURCES USED

(https://www.kaggle.com/datasets/kazanova/sentiment140)

The dataset provided is the Sentiment140 Dataset which consists of 1,600,000 tweets that have been extracted using the Twitter API. The various columns present in this Twitter data are:

- target: the polarity of the tweet (positive or negative)
- ids: Unique id of the tweet
- date: the date of the tweet
- flag: It refers to the query. If no such query exists, then it is NO QUERY.
- user: It refers to the name of the user that tweeted
- text: It refers to the text of the tweet

DATA PREPROCESSING

Initially, I indexed the dataset and proceeded to eliminate duplicate entries. Subsequently, I employed a curated stop-word list sourced from the internet to remove irrelevant terms from the text column. I systematically cleaned the data by eliminating repeated characters, URLs, numbers, and punctuations using dedicated functions.

Additionally, I standardized the text by converting all uppercase letters to lowercase.
Following the data cleaning phase, I performed tokenization, a critical step in preparing text for sentiment analysis. Tokenization is essential for feature extraction, managing special characters, accommodating text variations, and creating a structured input for subsequent analysis.

Upon completing EDA, I encoded the text into numerical representations. This encoding step is vital for efficient training, allowing machine learning models to process the data effectively.

DATA PREPROCESSING

Code snippets:

```
#cleaning the text
import string
import re

stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s', 'same', 'she', "shes", 'should', "shouldve",'so', 'some', 'such',
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
             'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',
             'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',
             'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",
             "youve", 'your', 'yours', 'yourself', 'yourselves']

STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
df['text'] = df['text'].apply(lambda text: cleaning_stopwords(text))
print(df['text'].head())

def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
df['text'] = df['text'].apply(lambda x: cleaning_repeating_char(x))
print(df['text'].tail())

def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
df['text'] = df['text'].apply(lambda x: cleaning_URLs(x))
print(df['text'].tail())
```

```
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
df['text'] = df['text'].apply(lambda x: cleaning_numbers(x))
print(df['text'].tail())

def cleaning_uppercase(text):
  return text.lower()
df['text'] = df['text'].apply(lambda x: cleaning_uppercase(x))
print(df['text'].tail())

def cleaning_punctuations(text):
    punctuations = string.punctuation + "¡" + "¿"
    no_punc_sentence = text.translate(str.maketrans('', '', punctuations))
    return no_punc_sentence
df['text'] = df['text'].apply(lambda x: cleaning_punctuations(x))
print(df['text'].tail())
```

Code for cleaning data

DATA PREPROCESSING

Code snippets:

```python
from keras.preprocessing import sequence
from keras.datasets.imdb import get_word_index

# tokenizing text
word_index=get_word_index()
MAXLEN= 250


def tokenize_text(text):
    tokens=tf.keras.preprocessing.text.text_to_word_sequence(text)
    tokens=[word_index[word] if word in word_index else 0 for word in tokens]
    return sequence.pad_sequences([tokens], MAXLEN)[0]

```

Tokenizing text

```python
#test_train split
_, df = train_test_split(df, test_size=0.05, random_state=42)
train_data, test_data = train_test_split(df, test_size=0.5, random_state=42)
```

```python
#encoding data from the dataset
from tqdm import tqdm
for x in tqdm(range(len(train_data))):
  train_data.text.iloc[x]=tokenize_text(train_data.text.iloc[x])
for x in tqdm(range(len(test_data.text))):
  test_data.text.iloc[x]=tokenize_text(test_data.text.iloc[x])

train_labels = train_data.target
train_text=sequence.pad_sequences(train_data.text, MAXLEN)

test_labels = test_data.target
test_text=sequence.pad_sequences(test_data.text, MAXLEN)
```

Encoding text

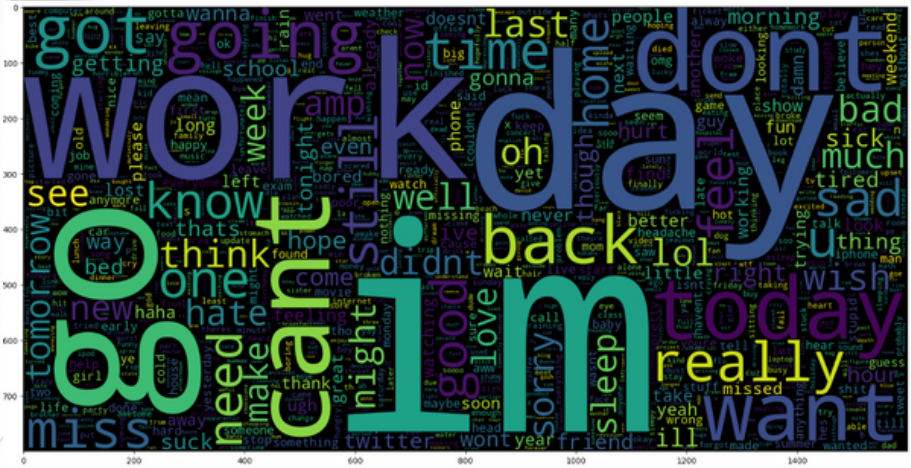
EXPLORATORY DATA ANALYSIS

During the exploratory data analysis (EDA) phase, I generated distinct word clouds for tweets associated with negative and positive sentiments. This visual exploration provided valuable insights into the most frequently used words within each sentiment category. Words like work, day, don’t , can’t show a majority in negative sentiment tweets while good, day, well are majority in the positive sentiment tweets. Additionally, I created count plots for positive and negative sentiment tweets, revealing an evenly distributed dataset.

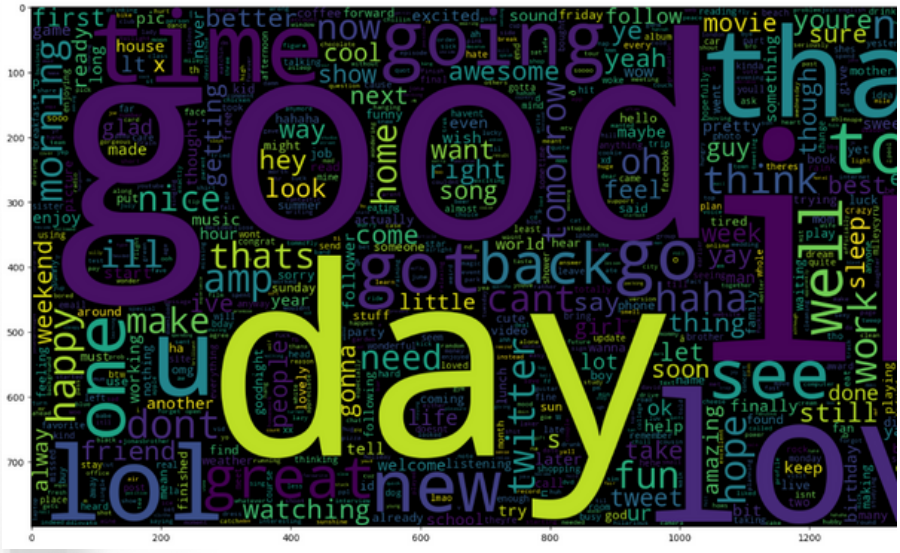
To further understand the dataset's characteristics, I plotted a histogram depicting the distribution of text lengths in the tweets. Notably, the histogram peaked at approx. 40 words and displayed a right-skewed distribution, shedding light on the varying lengths of tweets within the dataset. This information serves as a foundation for understanding the composition and structure of the text data.

EXPLORATORY DATA ANALYSIS

The following are the beautiful plots, that were plotted:

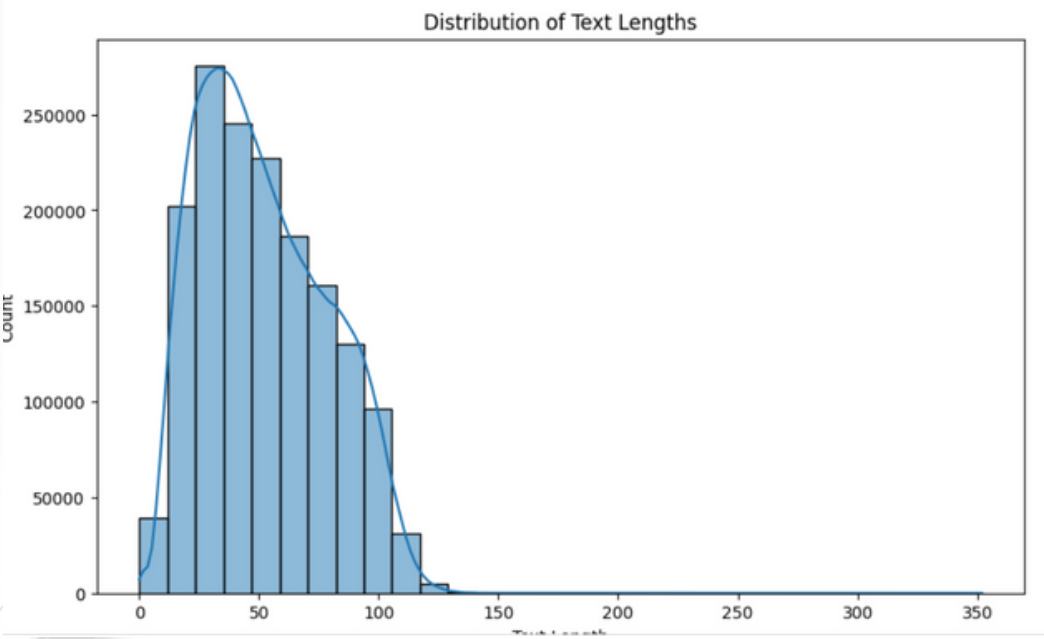


Word cloud for negative sentiment

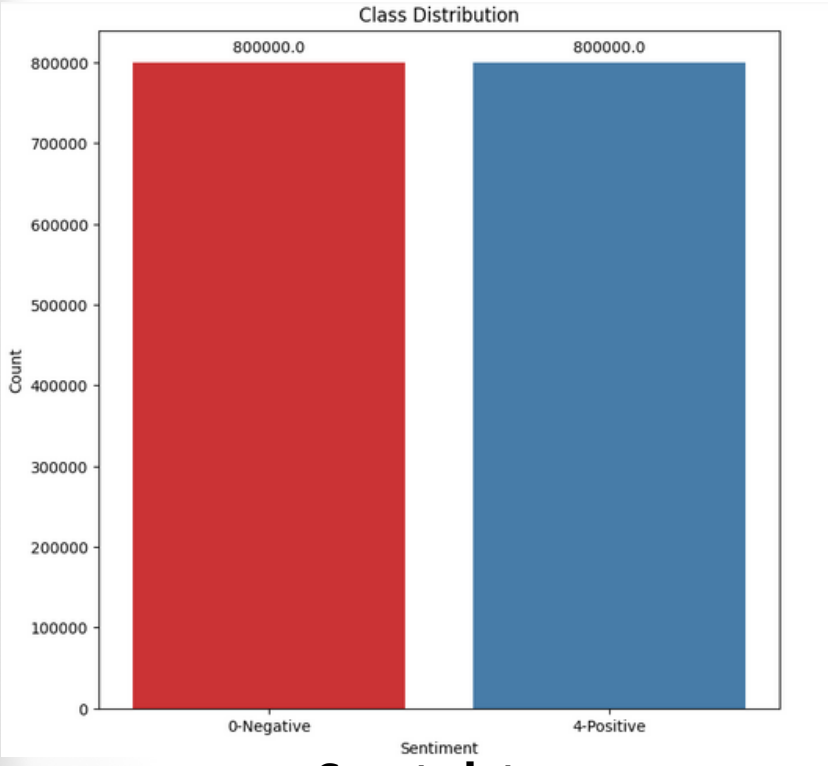


Word cloud for positive sentiment

EXPLORATORY DATA ANALYSIS

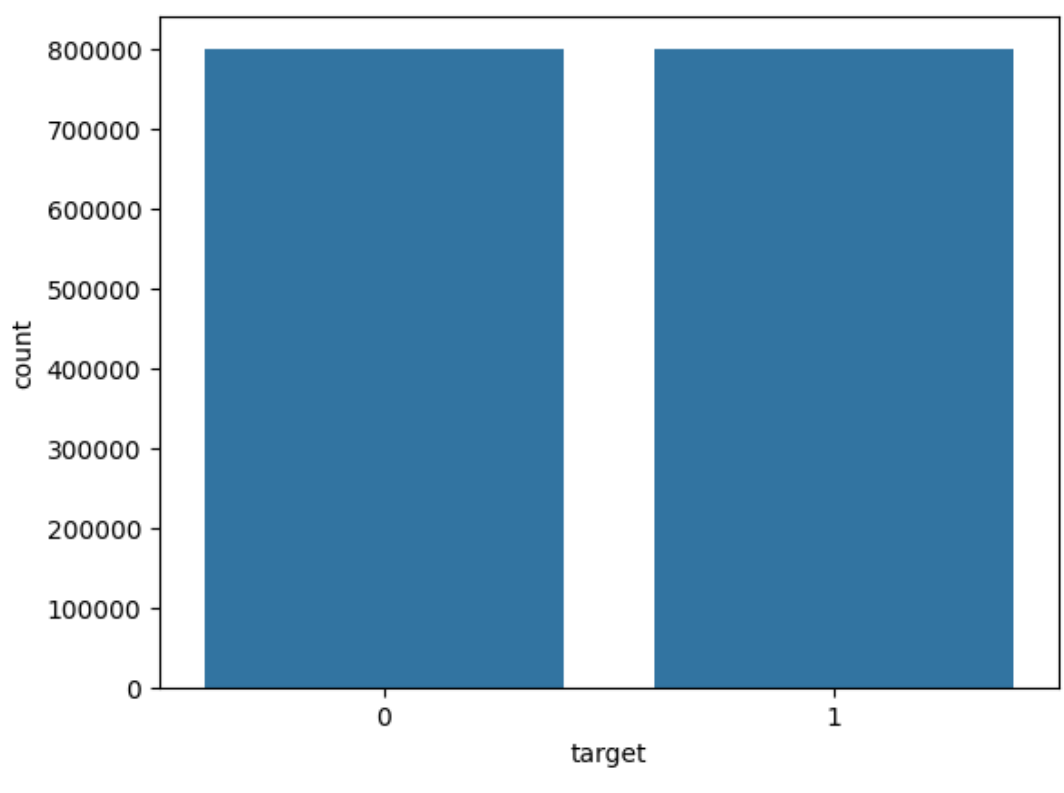


Histogram of word lengths



Count plot

EXPLORATORY DATA ANALYSIS



Count plot

MODEL IMPLEMENTATION

Model Selection: LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture. Unlike standard feedforward neural networks, LSTM has feedback connections. A standard LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

Why LSTM?

- **Handling sequential data:** Twitter data consists of sentences, which are sequences of words. LSTMs are specifically designed to excel in tasks involving sequential data, unlike traditional models that struggle to capture long-term dependencies.
- **Context awareness:** LSTMs can remember information from earlier words in a sentence, allowing them to consider the context when analyzing sentiment. This is crucial for understanding sarcasm, slang, and other nuances in tweets.
- **Parameter Sharing:** LSTMs share parameters across different parts of the sequence, making them more parameter-efficient compared to some other models. This can be important when dealing with large datasets, as it helps prevent overfitting and reduces the risk of model complexity.

MODEL IMPLEMENTATION

ACCURACY ACHIECHED:

71.88% On test Data

98.06% On train Data

Code Snippets:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout

# Define parameters
sequence_length = 250  # Maximum tweet length
embedding_dim = 100  # Dimension of word embeddings
vocab_size = 50000  # Number of unique words
lstm_units = 128  # Number of LSTM units

# Build the model
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=sequence_length))
model.add(LSTM(lstm_units))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

TRAINING AND EVALUATION

Code Snippets:

```
# Compile and train the model
from keras.callbacks import EarlyStopping

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
model.fit(train_text, train_labels, epochs=20, batch_size=32,verbose=1, callbacks=[early_stopping])
```

Compile and train the model

```
model.evaluate(test_text, test_labels)

```

```
1250/1250 [==============================] - 8s 6ms/step - loss: 2.2610 - accuracy: 0.7187
[2.2610273361206055, 0.7187250256538391]
```

Evaluation

RESULTS AND INSIGHTS

- Cleaning and preprocessing the text data increased the accuracy significantly. This is because, the model learns better from a uniform and simple dataset rather than a complex one.

- Effective tokenization helps capture relevant features and nuances within the text that are crucial for accurately discerning sentiment. Raw text is large and complex for machines to handle directly. Tokenizing breaks it down into smaller, more manageable units, making it easier for models like LSTMs to process and analyze the information.

- Models that take variable inputs and have a good pre and post accessing memory work well with sentiment analysis.

- Using regularization techniques like Early stopping and Patience improved the model's process significantly while evaluation. During training, the model's performance on a validation set is monitored. If there is no improvement in the validation performance for a certain number of consecutive epochs (patience), the training is stopped early.

- The model has been trained using a subset of the available data, specifically 42,000 tweets out of the total 16 million provided. However, expanding the training set to encompass a larger portion of the data is currently impractical due to technical constraints such as limited memory and RAM on my PC. Executing the training on a more extensive dataset is estimated to take approximately 72 hours. It is anticipated that training on a larger dataset could potentially lead to improved accuracy, given the additional diversity and richness of information available in the expanded data.

- **FINAL ACCURACY ACHIEVED: 71.87%**

RESOURCES USED

- **ChatGPT,**
- **BARD,**
- **https://www.analytixlabs.co.in/blog/twitter-sentiment-analysis/**
- **https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/**