News/Tweets Sentiment Analysis - Deep Learning problem

The mission of this project is to use deep learning techniques to develop a machine learning model that assesses the sentiment of textual content related to a business. This sentiment analysis of news and tweets could potentially indicate future stock price trends for the business.

Gather data, determine the method of data collection and provenance of the data (1 point)

The datasets used in the machine learning are full coming from Kaggle. I will be importing all the datasets from the Kaggle input directory.

Installing and importing dependencies

```
!pip install beautifulsoup4
!pip install tweepy
```

```
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.12.3-py3-none-any.whl (147 kB)
ent already satisfied: requests-oauthlib<2,>=1.2.0 in
/opt/conda/lib/python3.7/site-packages (from tweepy) (1.3.0)
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.3.2-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (136 kB)
ent already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests<3,>=2.27.0-
>tweepy) (1.26.3)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.7/site-packages (from requests<3,>=2.27.0-
>tweepy) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests<3,>=2.27.0-
>tweepy) (2020.12.5)
Installing collected packages: charset-normalizer, requests, oauthlib,
tweepy
  Attempting uninstall: requests
    Found existing installation: requests 2.25.1
    Uninstalling requests-2.25.1:
      Successfully uninstalled requests-2.25.1
  Attempting uninstall: oauthlib
    Found existing installation: oauthlib 3.0.1
    Uninstalling oauthlib-3.0.1:
      Successfully uninstalled oauthlib-3.0.1
```

```
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
conda 4.9.2 requires ruamel_yaml>=0.11.14, which is not installed.
jupyterlab-git 0.11.0 requires nbdime<2.0.0,>=1.1.0, but you have
nbdime 2.1.0 which is incompatible.
Successfully installed charset-normalizer-3.3.2 oauthlib-3.2.2
requests-2.31.0 tweepy-4.14.0
```

```
import numpy as np # linear algebra
import pandas as pd # data processing
import os
import tweepy as tw #for accessing Twitter API
import pickle

#For Preprocessing
import re    # RegEx for removing non-letter characters
import nltk  #natural language processing
nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.stem.porter import *

# For Building the model
from sklearn.model_selection import train_test_split
import tensorflow as tf
import seaborn as sns

#For data visualization
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import plotly.express as px
%matplotlib inline

pd.options.plotting.backend = "plotly"
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Identify a Deep Learning Problem

This problem of assessing sentiment from text is well-suited to deep learning techniques. Given that the problem involves processing text sequences, a recurrent neural network (RNN) architecture is appropriate. I will experiment with different approaches, including simpleRNN and long short-term memory (LSTM), to train the machine learning model. Given the relatively short nature of the text input in the dataset, simpleRNN might be sufficient. However, I will also explore the more advanced LSTM approach, which can remember longer sequences, to see how these different methods compare in generating better results.

Inspect, Visualize and Clean the Data

Determine if any data needs to be transformed. For example, if you're planning on using an SVM method for prediction, you may need to normalize or scale the data if there is a considerable difference in the range of the data.

Using your hypothesis, indicate if it's likely that you should transform data, such as using a log transform or other transformation of the dataset.

You should determine if your data has outliers or needs to be cleaned in any way. Are there missing data values for specific factors? How will you handle the data cleaning? Will you discard, interpolate or otherwise substitute data values?

If you believe that specific factors will be more important than others in your analysis, you should mention which and why. You will use this to confirm your intuitions in your final write-up.

Show a few visualizations like histograms. Describe any data cleaning procedures. Based on your EDA, what is your plan of analysis?

Datasets

The following section provides a detailed description of the data to be used for machine learning.

The original dataset, sourced from the Financial Phrase Bank, contains sentiment annotations for financial news headlines from the perspective of a retail investor. One reason why statistical techniques in financial sentiment analysis have low utilization is the lack of high-quality training data. The Financial Phrase Bank provides a rare collection of annotated financial news. This collection, consisting of approximately 5,000 news headlines, was annotated from an investor's viewpoint to determine whether the information might have a positive, negative, or neutral influence on stock prices.

To determine sentiment from other perspectives that could potentially impact businesses and, consequently, their stock performance, additional textual data with sentiment labels are aggregated with the Financial Phrase Bank's data. For example, datasets of tweets related to Apple and its products and services, as well as tweets about various airlines, are included. These aggregated datasets provide a broader view of sentiment on different aspects of the business, which could influence its stock price.

Describe the factors or components that make up the dataset

This release of the financial phrase bank covers a collection of 4840 sentences. The selected collection of phrases was annotated by 16 people with adequate background knowledge on financial markets and the sentiment given is either "positive, neutral or negative".

Besides, Financial Phrase Bank's dataset, the following four different datasets will be aggragated to become a more diverse dataset with sentiment data.

- appletwittersentimenttexts
- twitter-airline-sentiment

- twitter-and-reddit-sentimental-analysis-dataset
- finalSentimentdata2

Correlations between different factors of the dataset

All five datasets used in this project have various features (columns) providing different types of textual data. However, they all share a common structure in their CSV files, each containing two essential columns: one for the text data and one for sentiment assessment. Below is more detailed information about each dataset:

- Financial Phrase Bank News: This dataset collects financial news headlines and their sentiment assessments. It contains a CSV file with two columns: "headlines" and "sentiment."
- Apple Twitter Sentiment: This dataset contains tweets related to the company Apple, covering aspects such as its products, services, and support experiences. It contains a CSV file with two columns: "tweets" and "sentiment."
- Airline Twitter Sentiment: This dataset contains tweets about various airline companies, such as Virgin America. Its CSV file contains 15 columns, including "tweets," "sentiment," "airline name," "negative reason," "retweet count," and more.
- Reddit and Twitter Sentiment: This dataset includes tweets and discussions from Reddit and Twitter, covering a wide range of topics. There are two CSV files, each with two columns: "text" and "sentiment," storing data from Reddit and Twitter respectively.
- Twitter Data: This dataset contains general tweets, primarily from users in regions close to India. Its CSV file has two columns: "text" and "sentiment."

Data Transform

The structure of these datasets is relatively simple. The major transformation needed is to extract the "text" and "sentiment" columns from each dataset and aggregate them into a combined dataset.

Since the sentiment column is encoded slightly differently across datasets—some using numeric representations like -1, 0, and 1, and others using textual representations like "positive," "neutral," and "negative"—one essential transformation process is to standardize the sentiment labels to a textual format with categories: "positive," "neutral," and "negative."

Important factors for the analysis

The combined dataset comprises 187,174 sentences (one per row) along with their corresponding sentiments (positive, neutral, and negative).

These five datasets, though not directly related by a common topic, collectively showcase different aspects of user sentiment related to a business:

- Financial Phrase Bank News: Provides a direct assessment of potential immediate impacts on a business's stock price.
- Apple and Airline Twitter Sentiment: Reflects customer feelings about a company's products or services.
- Reddit and Twitter Sentiment: Offers a general sentiment assessment across a broad range of topics.

- Twitter Data from India: Helps uncover sentiment assessments from a specific cultural perspective.

By combining these datasets, we aim to capture a comprehensive view of sentiment that could potentially influence a business's stock price.

Loading and prepping dataset

The datasets will be loaded from the Kaggle directory and aggregated into a single pandas DataFrame.

```
# Load finanical news dataset
df1 = pd.read_csv('../input/sentiment-analysis-for-financial-news/all-data.csv',delimiter=',',encoding='latin-1')
df1 = df1.rename(columns={'neutral':'category','According to Gran , the company has no plans to move all production to Russia , although that is where the company is growing .':'clean_text'})
df1['category'] = df1['category'].map({'negative': -1.0, 'neutral': 0.0, 'positive':1.0})
df1.head()
```

```
   category                                         clean_text
0       0.0  Technopolis plans to develop in stages an area...
1      -1.0  The international electronic industry company ...
2       1.0  With the new production plant the company woul...
3       1.0  According to the company 's updated strategy f...
4       1.0  FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...
```

```
# Load Tweet dataset
df2 = pd.read_csv('../input/appletwittersentimenttexts/apple-twitter-sentiment-texts.csv')
df2 = df2.rename(columns={'text': 'clean_text', 'sentiment':'category'})
df2['category'] = df2['category'].map({-1: -1.0, 0: 0.0, 1:1.0})
df2.head()
```

```
                                          clean_text  category
0  Wow. Yall needa step it up @Apple RT @heynyla:...      -1.0
1  What Happened To Apple Inc?   http://t.co/FJEX...       0.0
2  Thank u @apple I can now compile all of the pi...       1.0
3  The oddly uplifting story of the Apple co-foun...       0.0
4  @apple can i exchange my iphone for a differen...       0.0
```

```
# Load Tweet dataset
df3 = pd.read_csv('../input/twitterdata/finalSentimentdata2.csv')
df3 = df3.rename(columns={'text': 'clean_text', 'sentiment':'category'})
df3['category'] = df3['category'].map({'sad': -1.0, 'anger': -1.0, 'fear': -1.0, 'joy':1.0})
```

```
df3 = df3.drop(['Unnamed: 0'], axis=1)
df3.head()
```

```
   category                                         clean_text
0      -1.0  agree the poor in india are treated badly thei...
1       1.0  if only i could have spent the with this cutie...
2       1.0  will nature conservation remain a priority in ...
3      -1.0  coronavirus disappearing in italy show this to...
4      -1.0  uk records lowest daily virus death toll since...
```

```
# Load Tweet dataset
df4 = pd.read_csv('../input/twitter-airline-sentiment/Tweets.csv')
df4 = df4.rename(columns={'text': 'clean_text',
'airline_sentiment':'category'})
df4['category'] = df4['category'].map({'negative': -1.0, 'neutral':
0.0, 'positive':1.0})
df4 = df4[['category','clean_text']]
# Output first five rows
df4.head()
```

```
   category                                         clean_text
0       0.0                @VirginAmerica What @dhepburn said.
1       1.0  @VirginAmerica plus you've added commercials t...
2       0.0  @VirginAmerica I didn't today... Must mean I n...
3      -1.0  @VirginAmerica it's really aggressive to blast...
4      -1.0  @VirginAmerica and it's a really big bad thing...
```

```
# Load Tweet dataset
df5 = pd.read_csv('../input/twitter-and-reddit-sentimental-analysis-
dataset/Twitter_Data.csv')
# Output first five rows
df5.head()
```

```
                                          clean_text  category
0  when modi promised “minimum government maximum...      -1.0
1  talk all the nonsense and continue all the dra...       0.0
2  what did just say vote for modi  welcome bjp t...       1.0
3  asking his supporters prefix chowkidar their n...       1.0
4  answer who among these the most powerful world...       1.0
```

```
# aggregating all the datasets
df = pd.concat([df1, df2, df3, df4, df5], ignore_index=True)  #added
df5
```

Cleaning and adjusting data

To cleanse the "text" column as described, here's how you can process it in Python using pandas and basic string operations:

- Drop NaN and None records: Remove rows where the "text" column is NaN or None.

- Remove HTML tags and syntax: Clean the text by removing HTML tags and any unnecessary syntax.
- Convert text to string: Ensure all entries in the "text" column are converted to strings.
- Encode sentiment value: Convert the sentiment labels to numeric encoding if needed.

```
from bs4 import BeautifulSoup
def cleanText(text):
    if text is None or pd.isna(text):
        text = ''
    else:
        text = str(text)  # Convert to string if not already
    text = BeautifulSoup(text, "lxml").text
    text = re.sub(r'\|\|\|', r' ', text)
    text = re.sub(r'http\S+', r'<URL>', text)
    #text = text.lower()
    #text = text.replace('x', '')
    #text = text.fillna('').astype(str)
    return text

df.isnull().sum()
df.dropna(axis=0, inplace=True)
df.fillna('', inplace=True)
df['clean_text'] = df['clean_text'].apply(cleanText)#.astype(str)
# Map tweet categories
df['category'] = df['category'].map({-1.0:'Negative', 0.0:'Neutral',
1.0:'Positive'})
# dimensionality of the data
print(df.shape)

#########################
# SAVE df to disk
#########################
df.to_csv('my_sentimentdata.csv', index=False)
df.head()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:7:
MarkupResemblesLocatorWarning: The input looks more like a filename
than markup. You may want to open this file and pass the filehandle
into Beautiful Soup.
  import sys
```

```
(187174, 2)
```

	category	clean_text
0	Neutral	Technopolis plans to develop in stages an area...
1	Negative	The international electronic industry company ...
2	Positive	With the new production plant the company woul...
3	Positive	According to the company 's updated strategy f...
4	Positive	FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...

```
########################
# LOAD df from disk
########################
#df.to_csv('my_sentimentdata.csv', index=False)
#df = pd.read_csv('my_sentimentdata.csv')
df = pd.read_csv('my_sentimentdata.csv', dtype={'clean_text': str})
df['clean_text'] = df['clean_text'].astype(str)  # this works
```

Exploratory Data Analysis

The chart below illustrates the distribution of sentiment labels in the dataset. It shows a balanced distribution across sentiments: approximately 76,000 tweets with positive sentiment, 62,000 with neutral sentiment, and 48,000 with negative sentiment. This breaks down to about 41% positive, 33% neutral, and 26% negative sentiments respectively.

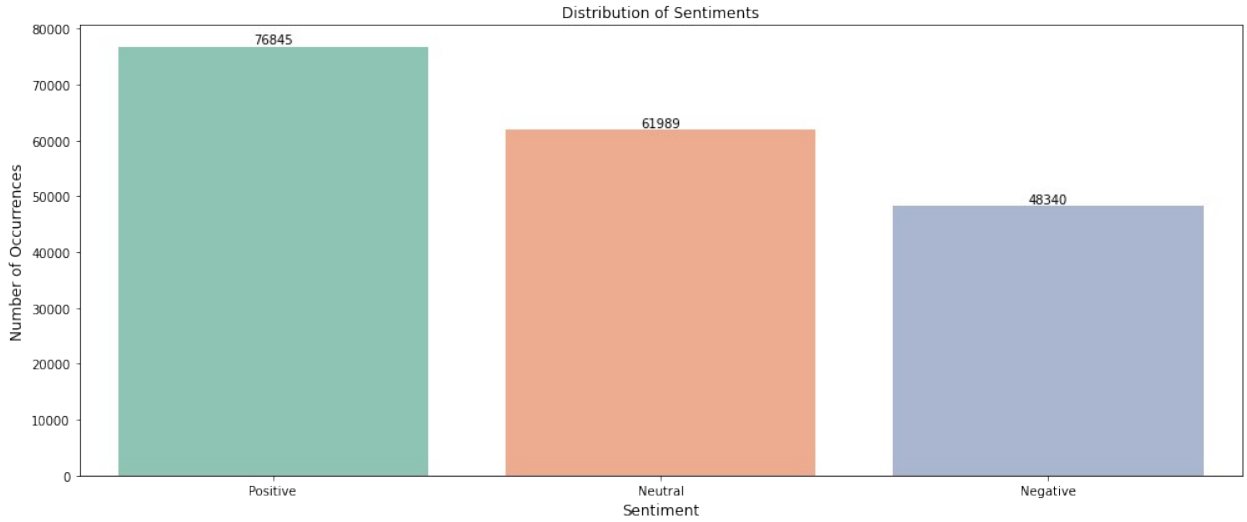
In total, the dataset contains 187,174 rows of text data.

```
# total number of tweets
print(f"Total number of tweet is {df.shape[0]}")

# Calculate sentiment breakdown
sentimentbreakdown = df['category'].value_counts()

# Set color palette for consistency
colors = sns.color_palette('Set2')

# Create a figure with subplots for bar plot and pie chart
fig, ax1 = plt.subplots(1, 1, figsize=(14, 6))

# Bar plot
sns.barplot(sentimentbreakdown.index, sentimentbreakdown.values,
alpha=0.8, ax=ax1, palette=colors)
ax1.set_ylabel('Number of Occurrences', fontsize=12)
ax1.set_xlabel('Sentiment', fontsize=12)
ax1.set_title('Distribution of Sentiments')

# Add count values on each bar
for index, value in enumerate(sentimentbreakdown.values):
    ax1.text(index, value, str(value), ha='center', va='bottom')

# Pie chart
pie_data = df['category'].value_counts(normalize=True).reset_index()
pie_data.columns = ['category', 'percentage']
fig2 = px.pie(pie_data, values='percentage', names='category',
title='Pie Chart of Sentiment Distribution',
              color_discrete_sequence=px.colors.qualitative.Set2)

# Adjust pie chart properties
```

```
fig2.update_traces(textposition='inside', textinfo='percent+label')

# Show plots
plt.tight_layout()
plt.show()
fig2.show()
```

```
Total number of tweet is 187174
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y.
From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  FutureWarning
```

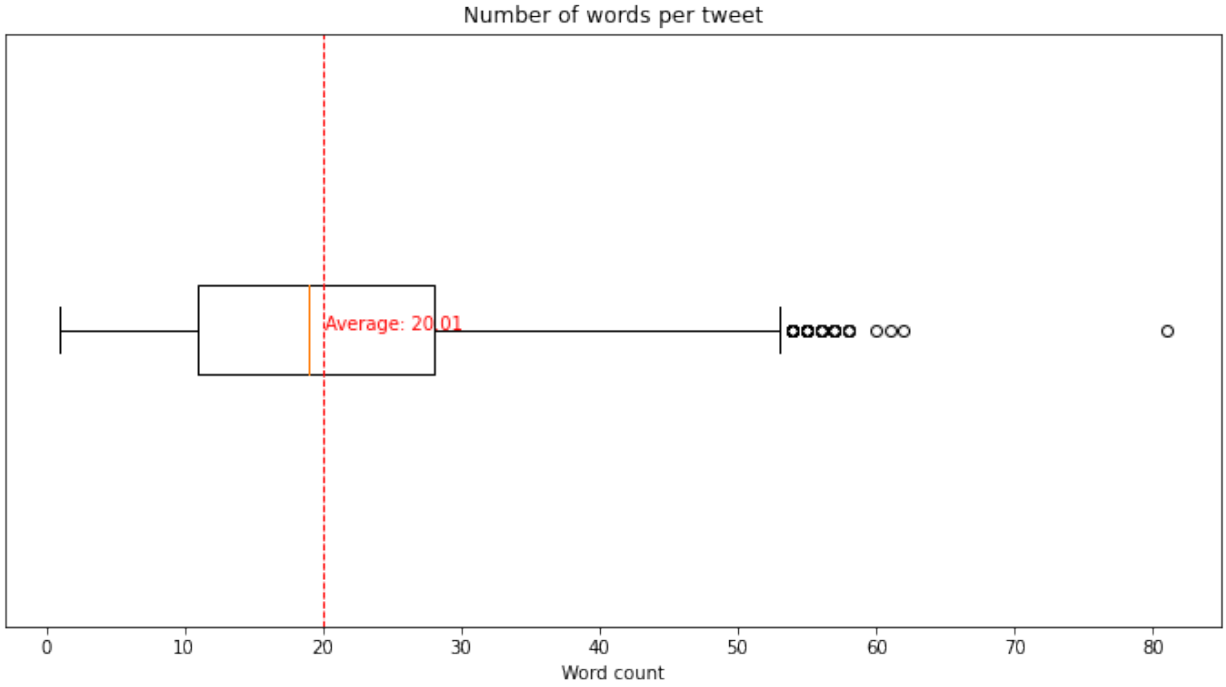


The box plot below displays the distribution of the number of words in the content (tweets). On average, tweets contain around 20 words. Instances with more than approximately 50 words are considered outliers. Setting a maximum input length of 50 words for the training data may be appropriate.

```
# Check the data type of the 'clean_text' column
print(df['clean_text'].dtypes)

# If the column contains text data, proceed with the calculation
if df['clean_text'].dtype == 'object':
    # Calculate tweet lengths
    tweet_len = pd.Series([len(str(tweet).split()) for tweet in
df['clean_text']])

    # Creating the box plot
    plt.figure(figsize=(12, 6))
    plt.boxplot(tweet_len, vert=False)
```

```
    avg_word_count = tweet_len.mean()
    plt.axvline(x=avg_word_count, color='r', linestyle='--',
linewidth=1)
    plt.text(avg_word_count + 0.1, 1, f'Average:
{avg_word_count:.2f}', color='r')
    plt.title('Number of words per tweet')
    plt.xlabel('Word count')
    plt.yticks([])  # Disable y ticks
    plt.show()
else:
    print("The 'clean_text' column does not contain text data.")
```

```
object
```



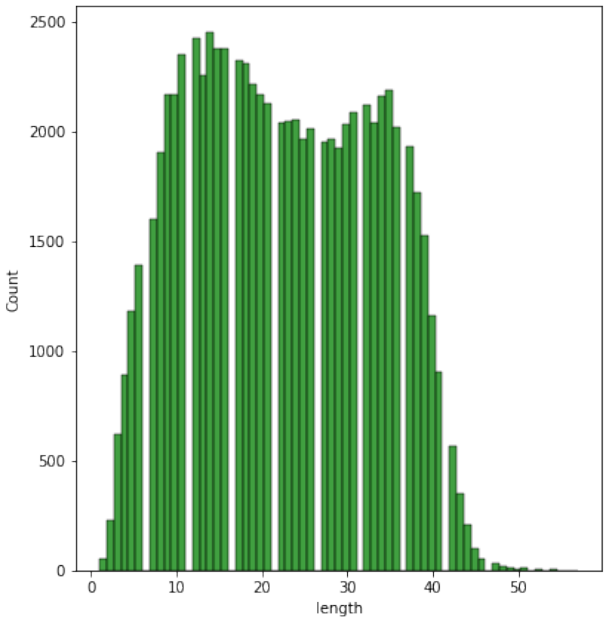
The plot below shows the distribution of text lengths for tweets with positive sentiment. The distribution appears to be bimodal, indicating two distinct peaks. This suggests that there are two main clusters or groups within the positive sentiment tweets. One cluster centers around shorter tweets, while the other is characterized by longer tweets. Exploring these two different behaviors further could be valuable to assess their impact on the machine learning model.

```
fig = plt.figure(figsize=(14,7))
#df['length'] = df.clean_text.str.split().apply(len)
df['length'] = df['clean_text'].astype(str).str.split().apply(len)
ax1 = fig.add_subplot(122)  # 1 row, 2 col, second chart
sns.histplot(df[df['category']=='Positive']['length'],
ax=ax1,color='green')
describe =
df.length[df.category=='Positive'].describe().to_frame().round(2)
```

```
ax2 = fig.add_subplot(121)
ax2.axis('off')
font_size = 14
bbox = [0, 0, 1, 1]
table = ax2.table(cellText = describe.values, rowLabels =
describe.index, bbox=bbox, colLabels=describe.columns)
table.set_fontsize(font_size)
fig.suptitle('Distribution of text length for positive sentiment
tweets.', fontsize=16)

plt.show()
```

Distribution of text length for positive sentiment tweets.

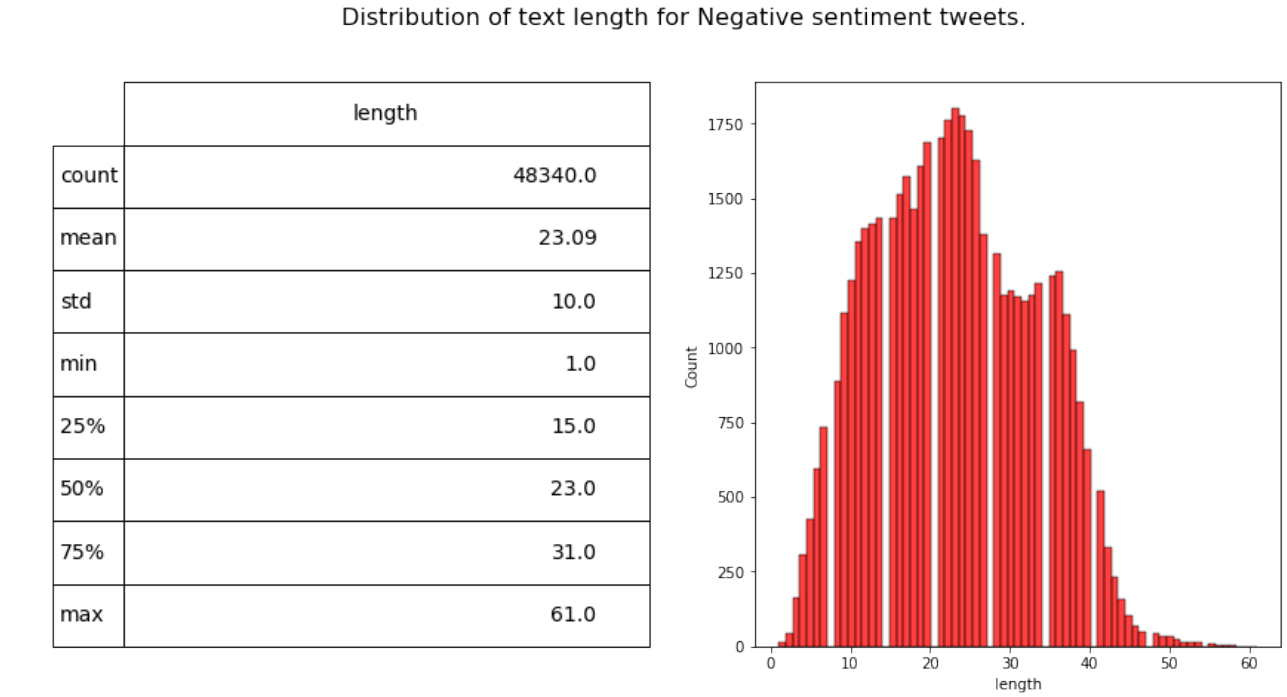
	length
count	76845.0
mean	22.4
std	10.63
min	1.0
25%	13.0
50%	22.0
75%	31.0
max	57.0



The plot below illustrates the distribution of text lengths for tweets with negative sentiment. While not as pronounced as in positive sentiment tweets, there appears to be a bimodal pattern. This suggests the presence of two distinct groups or clusters within the negative sentiment tweets. Further investigation into these behavioral patterns could provide insights into their potential impact on the machine learning model.

```
fig = plt.figure(figsize=(14,7))
#df['length'] = df.clean_text.str.split().apply(len)
df['length'] = df['clean_text'].astype(str).str.split().apply(len)

ax1 = fig.add_subplot(122)
sns.histplot(df[df['category']=='Negative']['length'],
ax=ax1,color='red')
describe =
df.length[df.category=='Negative'].describe().to_frame().round(2)
```

```
ax2 = fig.add_subplot(121)
ax2.axis('off')
font_size = 14
bbox = [0, 0, 1, 1]
table = ax2.table(cellText = describe.values, rowLabels =
describe.index, bbox=bbox, colLabels=describe.columns)
table.set_fontsize(font_size)
fig.suptitle('Distribution of text length for Negative sentiment
tweets.', fontsize=16)

plt.show()
```




```
# cleanup
df.drop(['length'], axis=1, inplace=True)
```

```
<bound method NDFrame.head of         category
clean_text
0        Neutral  Technopolis plans to develop in stages an area...
1       Negative  The international electronic industry company ...
2       Positive  With the new production plant the company woul...
3       Positive  According to the company 's updated strategy f...
4       Positive  FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...
...          ...                                                ...
187169  Negative  why these 456 crores paid neerav modi not reco...
187170  Negative  dear rss terrorist payal gawar what about modi...
187171   Neutral  did you cover her interaction forum where she ...
187172   Neutral  there big project came into india modi dream p...
```

```
187173  Positive  have you ever listen about like gurukul where ...

[187174 rows x 2 columns]>
```

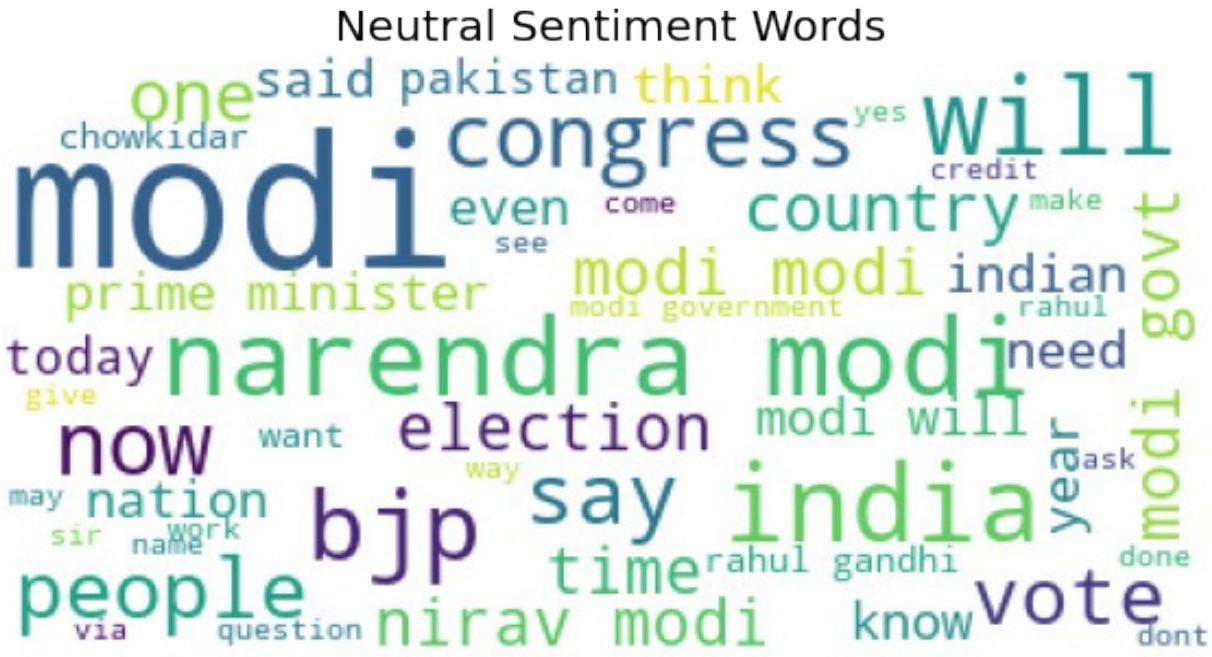
The word cloud below displays the relative frequency of the most common words in the tweet corpus. Notably, words like "Modi" and "India" appear frequently across all three sentiment categories. This suggests that some datasets may originate from sources with a significant number of users from India. This raises a concern that the sentiment captured in the data may be influenced by the cultural perspective of these users. If this specific cultural context does not reflect the sentiment perceived by the average person, it could introduce bias into the model trained on this data.

Further analysis is recommended to assess the dominance of tweets originating from India compared to other regions. Adjustments to balance the dataset may be necessary to mitigate potential cultural biases in the model.

```
#### Visualizing data into wordclouds

from wordcloud import WordCloud, STOPWORDS

def wordcount_gen(df, category):

    # Combine all tweets
    combined_tweets = " ".join([tweet for tweet in
df[df.category==category]['clean_text']])

    # Initialize wordcloud object
    wc = WordCloud(background_color='white',
                   max_words=50,
                   stopwords = STOPWORDS)

    # Generate and plot wordcloud
    plt.figure(figsize=(10,10))
    plt.imshow(wc.generate(combined_tweets))
    plt.title('{} Sentiment Words'.format(category), fontsize=20)
    plt.axis('off')
    plt.show()

# Positive tweet words
wordcount_gen(df, 'Positive')

# Negative tweet words
wordcount_gen(df, 'Negative')

# Neutral tweet words
wordcount_gen(df, 'Neutral')
```

Positive Sentiment Words

Negative Sentiment Words



Data Preprocessing

The following process converts tweets into a sequence of words represented as a list.

```
# Convert tweet text into a sequence of words
def tweet_to_words(tweet):
    text = tweet.lower()
    # remove non letters
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    # tokenize
    words = text.split()
    # remove stopwords
    words = [w for w in words if w not in stopwords.words("english")]
    # apply stemming
    words = [PorterStemmer().stem(w) for w in words]
    # return list
    return words

print("\nOriginal tweet ->", df['clean_text'][0])
print("\nProcessed tweet ->", tweet_to_words(df['clean_text'][0]))
# Apply data processing to each tweet and
X = list(map(tweet_to_words, df['clean_text']))

####################
# Save X to a file
with open('processed_tweets.pkl', 'wb') as file:
    pickle.dump(X, file)
```

```
Original tweet -> Technopolis plans to develop in stages an area of no
less than 100,000 square meters in order to host companies working in
computer technologies and telecommunications , the statement said .

Processed tweet -> ['technopoli', 'plan', 'develop', 'stage', 'area',
'less', '100', '000', 'squar', 'meter', 'order', 'host', 'compani',
'work', 'comput', 'technolog', 'telecommun', 'statement', 'said']

---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-20-ab3839431e52> in <module>
     17
     18 # Apply data processing to each tweet and
---> 19 X = list(map(tweet_to_words, df['clean_text']))
     20
     21 ###################

<ipython-input-20-ab3839431e52> in tweet_to_words(tweet)
      7     words = text.split()
      8     # remove stopwords
----> 9     words = [w for w in words if w not in
stopwords.words("english")]
     10     # apply stemming
     11     words = [PorterStemmer().stem(w) for w in words]

<ipython-input-20-ab3839431e52> in <listcomp>(.0)
      7     words = text.split()
      8     # remove stopwords
----> 9     words = [w for w in words if w not in
stopwords.words("english")]
     10     # apply stemming
     11     words = [PorterStemmer().stem(w) for w in words]

/opt/conda/lib/python3.7/site-packages/nltk/corpus/reader/wordlist.py
in words(self, fileids, ignore_lines_startswith)
     20     """
     21     def words(self, fileids=None, ignore_lines_startswith='\
n'):
---> 22         return [line for line in
line_tokenize(self.raw(fileids))
     23                 if not
line.startswith(ignore_lines_startswith)]
     24

/opt/conda/lib/python3.7/site-packages/nltk/corpus/reader/wordlist.py
in <listcomp>(.0)
     20     """
```

```
     21     def words(self, fileids=None, ignore_lines_startswith='\
n'):
---> 22         return [line for line in
line_tokenize(self.raw(fileids))
     23                 if not
line.startswith(ignore_lines_startswith)]
     24

KeyboardInterrupt:
```

```
#####################
# To load df and X back
#df.to_csv('my_sentimentdata.csv', index=False)

# load dataset as a df
df = pd.read_csv('my_sentimentdata.csv')
df['clean_text'] = df['clean_text'].astype(str)  # this works

# load tweet word as a list
with open('processed_tweets.pkl', 'rb') as file:
    X = pickle.load(file)
```

To prepare the data for training, the following code will encode the labels from text to numeric representation:

- Negative is encoded as 0
- Neutral is encoded as 1
- Positive is encoded as 2

```
from sklearn.preprocessing import LabelEncoder

# Encode target labels
le = LabelEncoder()
Y = le.fit_transform(df['category'])

# Print the numerical labels assigned to each original category
for label, category in zip(le.transform(le.classes_), le.classes_):
    print(f"{category} is encoded as {label}")

print(f"Total number of tweet is {len(Y)}")
```

```
Negative is encoded as 0
Neutral is encoded as 1
Positive is encoded as 2
Total number of tweet is 187174
```

Tokenizing & Padding

Tokenization and padding are crucial steps in preparing text data for NLP and RNN modeling. Tokenization converts text into numerical integers, essential for machine learning models.

Padding ensures that all tokenized sequences are of the same length, which is necessary for consistent input during model training.

Initially, the maximum number of words in the token vocabulary is set to 5000. A higher maximum could potentially improve model performance, but it would also increase the computational requirements. This setting will be reviewed to determine if a higher number of words is necessary.

Additionally, the maximum length of input text is set to 50. This length is sufficient because very few tweets in the dataset exceed 50 words, and those that do are considered outliers.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

max_words = 5000
max_len=50

def tokenize_pad_sequences(text):
    '''
    This function tokenize the input text into sequnences of intergers
and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=max_len)
    # return sequences
    return X, tokenizer

print('Before Tokenization & Padding \n', df['clean_text'][0])
X, tokenizer = tokenize_pad_sequences(df['clean_text'])
print('After Tokenization & Padding \n', X[0])

##############
# saving
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
Before Tokenization & Padding
 Technopolis plans to develop in stages an area of no less than
100,000 square meters in order to host companies working in computer
technologies and telecommunications , the statement said .
After Tokenization & Padding
 [1399   26 1539   63  335 1402   58  277  594   77  445 1555 4103
63
  717   26 1405  355   63 3004    3    2  582   81    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]

####################
# loading tokenized data
with open('tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
```

Spliting data into training, validation and test dataset

From the total of 187174 rows of tweets, they will be split into three different sets. Initially, 80% of the tweets will be randomly selected as the training set for modeling, and the remaining 20% will be allocated to the test set. From the training set, an additional 25% of the tweets will be randomly chosen to form the validation set. Below is the breakdown of the sizes for each dataset:

- Train Set -> 112304 tweets
- Validation Set -> 37435 tweets
- Test Set -> 37435 tweets

```
y = pd.get_dummies(df['category'])
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=1)
print('Train Set ->', len(X_train), len(y_train))
print('Validation Set ->', len(X_val), len(y_val))
print('Test Set ->', len(X_test), len(y_test))
```

```
Train Set -> 112304 112304
Validation Set -> 37435 37435
Test Set -> 37435 37435
```

Model Architecture

I will employ deep learning and Recurrent Neural Network (RNN) architecture to construct a machine learning model for this problem. RNNs are particularly effective for modeling sequence data like tweet text.

I'll explore a range of architectures, starting from simpler models and progressively iterating to improve performance. The RNN model will incorporate various layers including Embedding, Convolution1D, Maxpooling, LSTM, Dropout, and Dense layers. Different combinations of these layers along with tuning hyperparameters will be explored to optimize the model and achieve the best performance for sentiment prediction.

Building version 1 basic simple RNN

The first version of the RNN model will be a basic configuration aimed at establishing a workflow and setting a performance baseline for more advanced models.

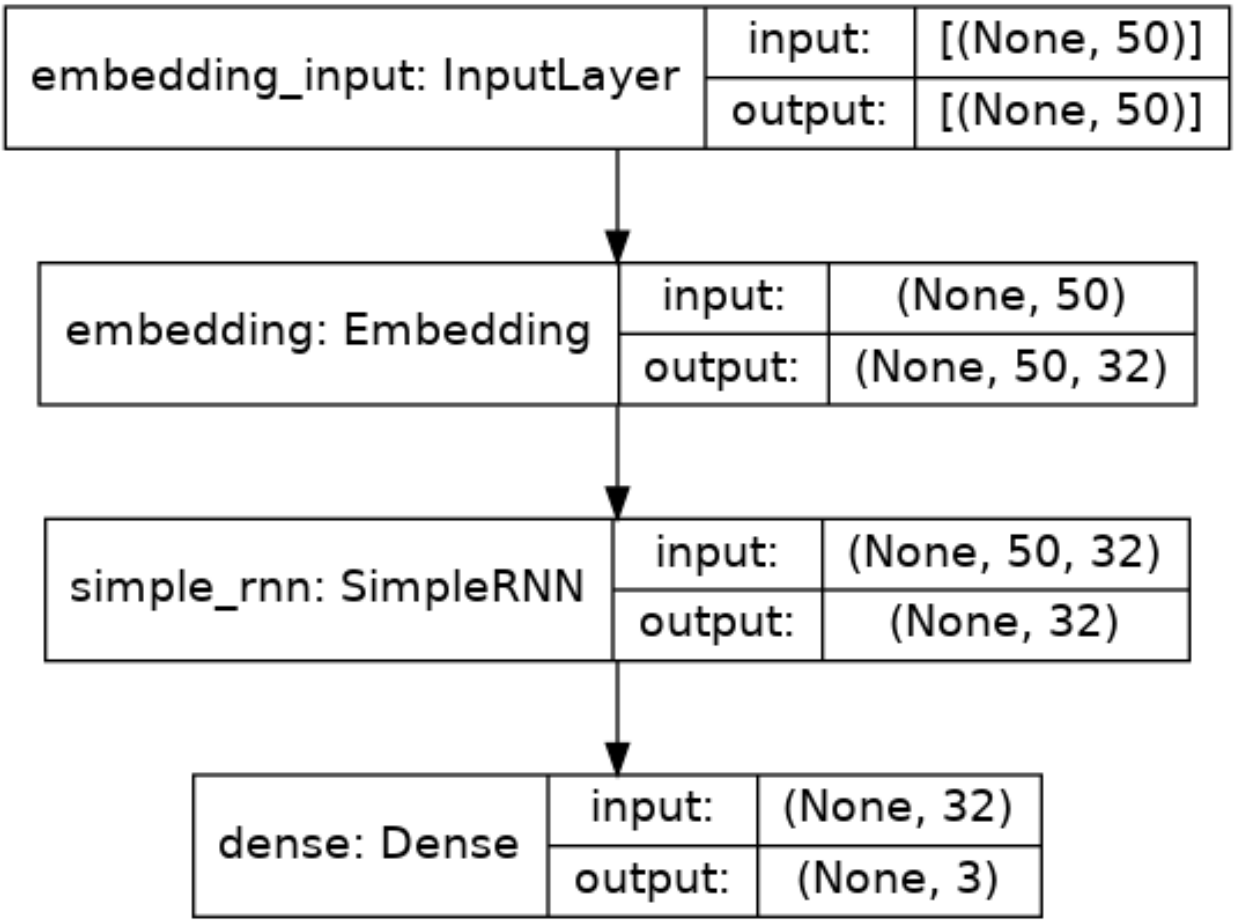
This initial RNN will include:

- An Embedding layer, which transforms categorical data into a continuous vector space.
- A SimpleRNN layer, which processes sequential data by passing a hidden state along with the input at each time step.
- A Dense layer, which is fully connected and commonly used in feedforward neural networks.

The model will be trained for 10 epochs to evaluate its performance with this architecture.

```
from keras.models import Sequential
from keras.models import Model
from keras.layers import Input, Embedding, SimpleRNN, Conv1D,
MaxPooling1D, Bidirectional, LSTM, Dense, Dropout
from keras.metrics import Precision, Recall
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras import datasets

from keras.callbacks import LearningRateScheduler
from keras.callbacks import History
from keras import losses
import tensorflow as tf

vocab_size = 5000
embedding_size = 32
max_len = 50  # majority of tweets have less than 50 words
epochs = 10
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8

sgd = SGD(learning_rate=learning_rate, momentum=momentum,
decay=decay_rate, nesterov=False)

# Build model
model = Sequential()
# Embedding layer
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
#model.add(MaxPooling1D(pool_size=2))  ############## this makes a
difference!!!!!!!!!!!!
model.add(SimpleRNN(32, return_sequences=False))
# Dense layer
model.add(Dense(3, activation='softmax'))
```

```
# Compile the model
model.compile(optimizer=sgd, loss='categorical_crossentropy',
metrics=['accuracy', Precision(), Recall()])

# Show model
tf.keras.utils.plot_model(model, show_shapes=True)
```




```
print(model.summary())
# Compile model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy', Precision(), Recall()])
# Train model
batch_size = 64
history1 = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
embedding (Embedding)        (None, 50, 32)            160000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                2080
_________________________________________________________________
dense (Dense)                (None, 3)                 99
=================================================================
Total params: 162,179
Trainable params: 162,179
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
1755/1755 [==============================] - 55s 30ms/step - loss:
1.0867 - accuracy: 0.4016 - precision_1: 0.3791 - recall_1: 0.0070 -
val_loss: 1.0638 - val_accuracy: 0.4340 - val_precision_1: 0.5707 -
val_recall_1: 0.0421
Epoch 2/10
1755/1755 [==============================] - 54s 31ms/step - loss:
1.0811 - accuracy: 0.4126 - precision_1: 0.4403 - recall_1: 7.2102e-04
- val_loss: 1.0804 - val_accuracy: 0.4122 - val_precision_1:
0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 3/10
1755/1755 [==============================] - 54s 31ms/step - loss:
1.0824 - accuracy: 0.4089 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0804 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 4/10
1755/1755 [==============================] - 55s 31ms/step - loss:
1.0815 - accuracy: 0.4090 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0802 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 5/10
1755/1755 [==============================] - 54s 31ms/step - loss:
1.0813 - accuracy: 0.4098 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0803 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 6/10
1755/1755 [==============================] - 52s 30ms/step - loss:
1.0819 - accuracy: 0.4073 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0804 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 7/10
1755/1755 [==============================] - 53s 30ms/step - loss:
1.0814 - accuracy: 0.4099 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0802 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 8/10
1755/1755 [==============================] - 53s 30ms/step - loss:
1.0816 - accuracy: 0.4091 - precision_1: 0.0000e+00 - recall_1:
```

```
0.0000e+00 - val_loss: 1.0802 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 9/10
1755/1755 [==============================] - 52s 30ms/step - loss:
1.0812 - accuracy: 0.4102 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0803 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
Epoch 10/10
1755/1755 [==============================] - 52s 30ms/step - loss:
1.0808 - accuracy: 0.4117 - precision_1: 0.0000e+00 - recall_1:
0.0000e+00 - val_loss: 1.0803 - val_accuracy: 0.4122 -
val_precision_1: 0.0000e+00 - val_recall_1: 0.0000e+00
```

Results and Analysis

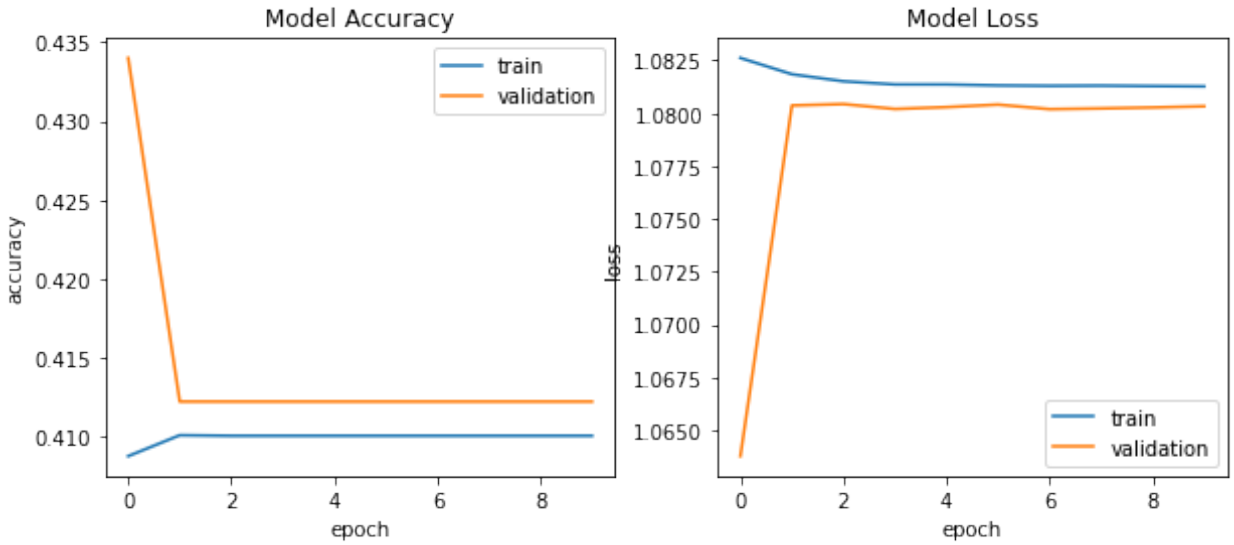
The primary layer in this initial version is the SimpleRNN layer, typically effective for basic sequence data. Despite its limitation in recalling distant data points, our tweet dataset consists mostly of short sequences (limited words), suggesting SimpleRNN could perform adequately.

Unexpectedly, the results were disappointing. The model's first version exhibited poor performance: the training dataset accuracy showed minimal improvement throughout all epochs, plateauing at approximately 40%. Validation accuracy also stagnated around 40%, showing no signs of improvement. Both training and validation losses remained high and exhibited instability.

This could indicate that the SimpleRNN layer struggles to effectively process the text in our datasets. SimpleRNN often faces challenges with long-term dependencies due to the vanishing gradient problem, which might contribute to the lack of improvement in accuracy and loss. Additionally, the RNN architecture may be lacking regularization layers, and the learning rate could possibly be too high.

```
### Function to plot history for accuracy and loss
def plot_training_hist(history):
    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    # first plot
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('Model Accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['train', 'validation'], loc='best')
    # second plot
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('Model Loss')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('loss')
    ax[1].legend(['train', 'validation'], loc='best')
```

```
plot_training_hist(history1)
```

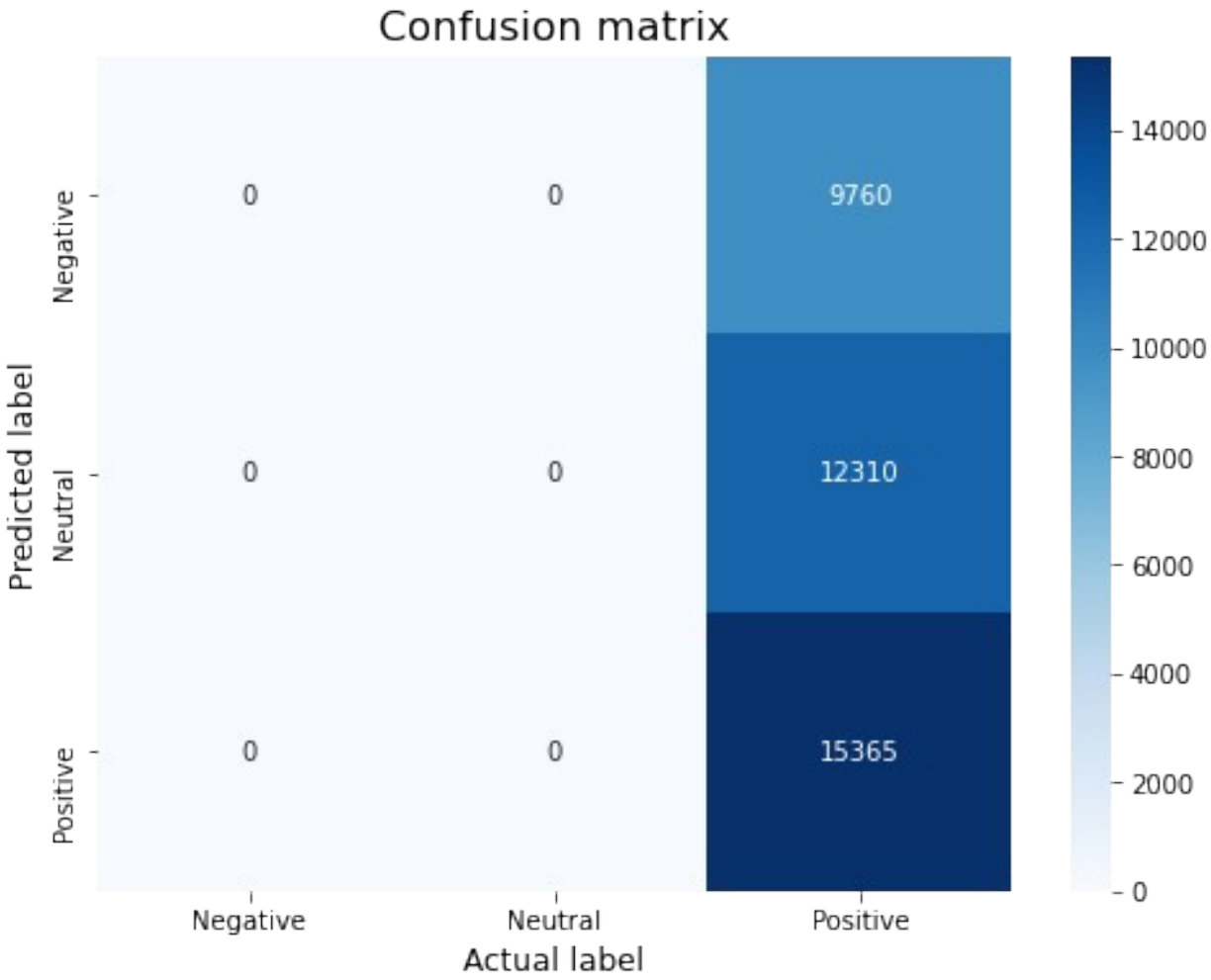



```
#Function to calculate f1 score
import keras.backend as K

def f1_score(precision, recall):
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)
# Print metrics
print('Performance of version 1 model')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

```
Performance of version 1 model
Accuracy  : 0.4104
Precision : 0.0000
Recall    : 0.0000
F1 Score  : 0.0000
```

```
from sklearn.metrics import confusion_matrix
## Function to plot confusion matrix for the passed model and the data
def plot_confusion_matrix(model, X_test, y_test):

    sentiment_classes = ['Negative', 'Neutral', 'Positive']
    y_pred = model.predict(X_test)
```

```
    cm = confusion_matrix(np.argmax(np.array(y_test),axis=1),
np.argmax(y_pred, axis=1))
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, cmap=plt.cm.Blues, annot=True, fmt='d',
xticklabels=sentiment_classes, yticklabels=sentiment_classes)
    plt.title('Confusion matrix', fontsize=16)
    plt.xlabel('Actual label', fontsize=12)
    plt.ylabel('Predicted label', fontsize=12)

plot_confusion_matrix(model, X_test, y_test)
```




```
# Save the model architecture & the weights
#model.save('best_model0.h5') #original
model.save('version1.h5') #my
print('Version 1 saved')
```

```
Version 1 saved
```

Building version 2

The main addition to model verion 2 is to add maxpooling layers after the embedding layers. Maxpooling could potential help extract important features from the data so that the RNN layer can focus on the most important critical aspects of the input sequences, potentially improving their ability to learn and generalize.

Max pooling does not directly address the gradients' size during backpropagation. However, by reducing the spatial size of the input and emphasizing important features, it can help simplify the learning process and make the optimization landscape smoother. This can indirectly make the gradients more stable.

I have doubled the number of epochs to 20 to give the model more time to learn from the data.

```
vocab_size = 5000
embedding_size = 32
max_len = 50  # majority of tweets have less than 50 words
epochs = 20
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8

sgd = SGD(learning_rate=learning_rate, momentum=momentum,
decay=decay_rate, nesterov=False)
# Build model
model = Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(MaxPooling1D(pool_size=2))  # version 2 addition
model.add(SimpleRNN(32, return_sequences=False))
model.add(Dense(3, activation='softmax'))
# Compile the model
model.compile(optimizer=sgd, loss='categorical_crossentropy',
metrics=['accuracy', Precision(), Recall()])
# Show model
tf.keras.utils.plot_model(model, show_shapes=True)
print(model.summary())
# Compile model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy', Precision(), Recall()])
# Train model
batch_size = 64
history2 = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 32)            160000
```

```
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 25, 32)            0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 162,179
Trainable params: 162,179
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
1755/1755 [==============================] - 34s 18ms/step - loss:
1.0708 - accuracy: 0.4236 - precision_3: 0.4834 - recall_3: 0.1060 -
val_loss: 1.0117 - val_accuracy: 0.4932 - val_precision_3: 0.6014 -
val_recall_3: 0.1296
Epoch 2/20
1755/1755 [==============================] - 31s 18ms/step - loss:
1.0240 - accuracy: 0.4764 - precision_3: 0.5491 - recall_3: 0.1909 -
val_loss: 1.0072 - val_accuracy: 0.4973 - val_precision_3: 0.5521 -
val_recall_3: 0.2924
Epoch 3/20
1755/1755 [==============================] - 32s 18ms/step - loss:
1.0011 - accuracy: 0.4963 - precision_3: 0.5790 - recall_3: 0.2356 -
val_loss: 0.9967 - val_accuracy: 0.5023 - val_precision_3: 0.5691 -
val_recall_3: 0.2643
Epoch 4/20
1755/1755 [==============================] - 32s 18ms/step - loss:
0.9913 - accuracy: 0.5031 - precision_3: 0.5871 - recall_3: 0.2334 -
val_loss: 0.9838 - val_accuracy: 0.5113 - val_precision_3: 0.6081 -
val_recall_3: 0.1864
Epoch 5/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.9745 - accuracy: 0.5140 - precision_3: 0.5901 - recall_3: 0.2537 -
val_loss: 0.9607 - val_accuracy: 0.5378 - val_precision_3: 0.5855 -
val_recall_3: 0.3815
Epoch 6/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.9519 - accuracy: 0.5421 - precision_3: 0.5959 - recall_3: 0.3782 -
val_loss: 0.9477 - val_accuracy: 0.5464 - val_precision_3: 0.6003 -
val_recall_3: 0.3918
Epoch 7/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.9352 - accuracy: 0.5531 - precision_3: 0.6077 - recall_3: 0.3911 -
val_loss: 0.9313 - val_accuracy: 0.5615 - val_precision_3: 0.6224 -
val_recall_3: 0.3941
Epoch 8/20
1755/1755 [==============================] - 32s 18ms/step - loss:
```

```
0.9160 - accuracy: 0.5695 - precision_3: 0.6196 - recall_3: 0.4280 -
val_loss: 0.9159 - val_accuracy: 0.5738 - val_precision_3: 0.6297 -
val_recall_3: 0.4405
Epoch 9/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8870 - accuracy: 0.5940 - precision_3: 0.6407 - recall_3: 0.4773 -
val_loss: 0.9033 - val_accuracy: 0.5854 - val_precision_3: 0.6319 -
val_recall_3: 0.4867
Epoch 10/20
1755/1755 [==============================] - 32s 18ms/step - loss:
0.8651 - accuracy: 0.6052 - precision_3: 0.6581 - recall_3: 0.4954 -
val_loss: 0.8767 - val_accuracy: 0.5980 - val_precision_3: 0.6546 -
val_recall_3: 0.4829
Epoch 11/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8506 - accuracy: 0.6121 - precision_3: 0.6681 - recall_3: 0.4983 -
val_loss: 0.8731 - val_accuracy: 0.5989 - val_precision_3: 0.6546 -
val_recall_3: 0.4946
Epoch 12/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8367 - accuracy: 0.6184 - precision_3: 0.6784 - recall_3: 0.5083 -
val_loss: 0.8692 - val_accuracy: 0.6026 - val_precision_3: 0.6569 -
val_recall_3: 0.5021
Epoch 13/20
1755/1755 [==============================] - 32s 18ms/step - loss:
0.8312 - accuracy: 0.6200 - precision_3: 0.6774 - recall_3: 0.5112 -
val_loss: 0.8674 - val_accuracy: 0.6029 - val_precision_3: 0.6543 -
val_recall_3: 0.4997
Epoch 14/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8234 - accuracy: 0.6237 - precision_3: 0.6824 - recall_3: 0.5128 -
val_loss: 0.8772 - val_accuracy: 0.5972 - val_precision_3: 0.6452 -
val_recall_3: 0.4988
Epoch 15/20
1755/1755 [==============================] - 30s 17ms/step - loss:
0.8116 - accuracy: 0.6305 - precision_3: 0.6892 - recall_3: 0.5254 -
val_loss: 0.8600 - val_accuracy: 0.6050 - val_precision_3: 0.6631 -
val_recall_3: 0.5004
Epoch 16/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8103 - accuracy: 0.6292 - precision_3: 0.6866 - recall_3: 0.5236 -
val_loss: 0.8692 - val_accuracy: 0.6038 - val_precision_3: 0.6567 -
val_recall_3: 0.5050
Epoch 17/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.8047 - accuracy: 0.6334 - precision_3: 0.6888 - recall_3: 0.5303 -
val_loss: 0.8703 - val_accuracy: 0.6038 - val_precision_3: 0.6606 -
val_recall_3: 0.4934
Epoch 18/20
```

```
1755/1755 [==============================] - 31s 18ms/step - loss:
0.7953 - accuracy: 0.6364 - precision_3: 0.6929 - recall_3: 0.5343 -
val_loss: 0.8612 - val_accuracy: 0.6068 - val_precision_3: 0.6700 -
val_recall_3: 0.4926
Epoch 19/20
1755/1755 [==============================] - 30s 17ms/step - loss:
0.7922 - accuracy: 0.6379 - precision_3: 0.6966 - recall_3: 0.5351 -
val_loss: 0.8562 - val_accuracy: 0.6089 - val_precision_3: 0.6597 -
val_recall_3: 0.5137
Epoch 20/20
1755/1755 [==============================] - 31s 18ms/step - loss:
0.7873 - accuracy: 0.6415 - precision_3: 0.6956 - recall_3: 0.5447 -
val_loss: 0.8590 - val_accuracy: 0.6084 - val_precision_3: 0.6678 -
val_recall_3: 0.4980
```

Results and Analysis

The verion 2 model has dramatic improvement over version 1 after 20 epochs. Although both train and validation's accruaracy rate are not particular good, they are indeed improving as they get more epochs. The validation accruarcy reaches 0.61 while training accruracy is close to 0.65. Train accruracy seems to continue to improve, but validation seems to be flat.

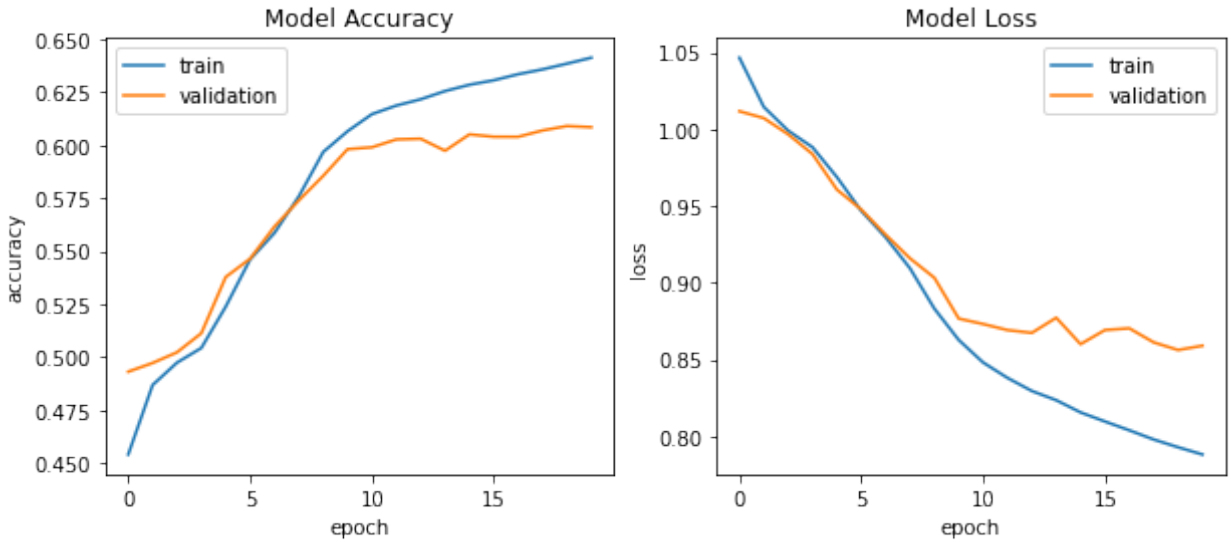
It seems that I won't get much more validation improvement even if I increase more epoch. The model probably needs more complex acrchitecture to learn from the data. Therefore, for the next version, I will replace the SimpleRNN with LSTM layers to see if I can improve the model performance.

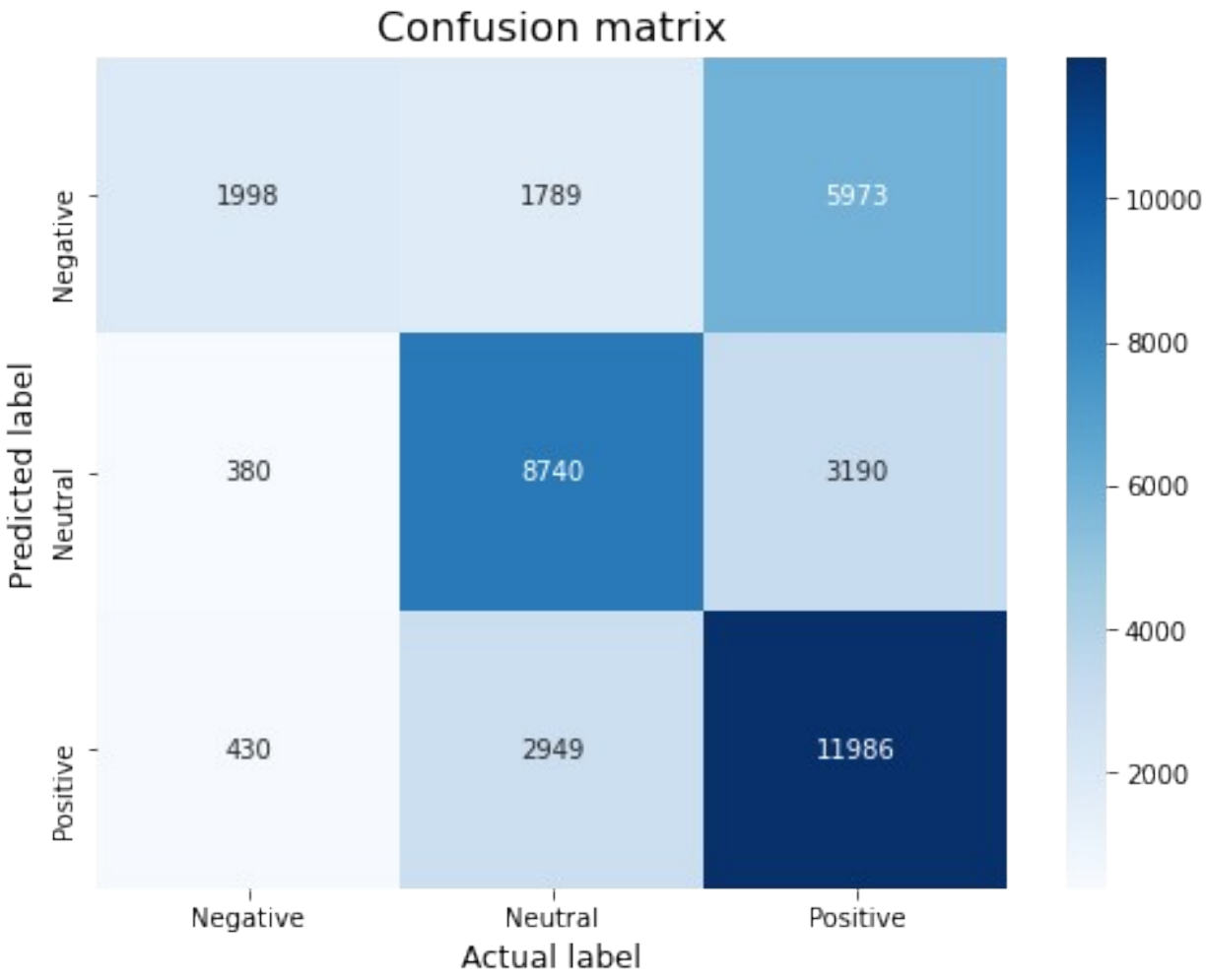
Upon examining the confusion matrix, it is evident that the model struggles to correctly predict negative sentiment content. This difficulty may stem from the relatively small amount of negative sentiment data available for training.

```
#plot training performance over epochs
plot_training_hist(history2)

# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)
# Print metrics
print('Performance of version 1 model')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))

#plot confusion matrix
plot_confusion_matrix(model, X_test, y_test)

# Save the model architecture & the weights
#model.save('best_model0.h5') #original
```

```
model.save('version2.h5') #my
print('Version 2 saved')
```

```
Performance of version 1 model
Accuracy  : 0.6070
Precision : 0.6631
Recall    : 0.4957
F1 Score  : 0.5673
Version 2 saved
```





Build version 3:

The highlight of the version 3 model is the inclusion of an LSTM layer in the RNN architecture. Based on the performance of the previous version, it appears that the SimpleRNN layer has reached its limit with this dataset. Therefore, I simply replaced the SimpleRNN layer with a non-bidirectional LSTM layer. This allows us to directly observe the impact of LSTM on the model, as no other changes were made beyond this replacement.

LSTMs are designed to capture long-term dependencies in the data due to their special gating mechanisms (input gate, forget gate, and output gate). These gates help LSTMs maintain information over long sequences and selectively forget irrelevant information. Additionally, LSTMs have more trainable parameters compared to SimpleRNN, which should enhance the model's ability to make better sentiment predictions.

```
vocab_size = 5000
embedding_size = 32
epochs=20
learning_rate = 0.1
decay_rate = learning_rate / epochs
```

```
momentum = 0.8

sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate,
nesterov=False)
# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(LSTM(32,return_sequences=False))
model.add(Dense(3,activation="softmax"))
model.compile(optimizer=sgd, loss='categorical_crossentropy',
metrics=['accuracy', Precision(), Recall()])
# Show model
tf.keras.utils.plot_model(model, show_shapes=True)
print(model.summary())
# Compile model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy', Precision(), Recall()])
# Train model
batch_size = 64
history3 = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     batch_size=batch_size, epochs=epochs, verbose=1)

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 50, 32)            160000
_________________________________________________________________
lstm (LSTM)                  (None, 32)                8320
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99
=================================================================
Total params: 168,419
Trainable params: 168,419
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
1755/1755 [==============================] - 15s 8ms/step - loss:
1.0647 - accuracy: 0.4265 - precision_5: 0.3618 - recall_5: 0.0706 -
val_loss: 1.0098 - val_accuracy: 0.4957 - val_precision_5: 0.5508 -
val_recall_5: 0.2935
Epoch 2/20
1755/1755 [==============================] - 12s 7ms/step - loss:
1.0041 - accuracy: 0.4964 - precision_5: 0.5732 - recall_5: 0.2575 -
val_loss: 0.9994 - val_accuracy: 0.5004 - val_precision_5: 0.5796 -
val_recall_5: 0.2570
Epoch 3/20
1755/1755 [==============================] - 13s 7ms/step - loss:
```

```
0.9944 - accuracy: 0.5024 - precision_5: 0.5791 - recall_5: 0.2581 -
val_loss: 0.9775 - val_accuracy: 0.5174 - val_precision_5: 0.6088 -
val_recall_5: 0.2477
Epoch 4/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.9571 - accuracy: 0.5294 - precision_5: 0.6057 - recall_5: 0.2946 -
val_loss: 0.8755 - val_accuracy: 0.5892 - val_precision_5: 0.6638 -
val_recall_5: 0.4319
Epoch 5/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.8745 - accuracy: 0.5861 - precision_5: 0.6612 - recall_5: 0.4439 -
val_loss: 0.8115 - val_accuracy: 0.6209 - val_precision_5: 0.6825 -
val_recall_5: 0.5436
Epoch 6/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.8271 - accuracy: 0.6135 - precision_5: 0.6918 - recall_5: 0.4917 -
val_loss: 0.7417 - val_accuracy: 0.6712 - val_precision_5: 0.7429 -
val_recall_5: 0.5671
Epoch 7/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.7056 - accuracy: 0.6875 - precision_5: 0.7537 - recall_5: 0.5833 -
val_loss: 0.6520 - val_accuracy: 0.7342 - val_precision_5: 0.7535 -
val_recall_5: 0.6981
Epoch 8/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.6499 - accuracy: 0.7288 - precision_5: 0.7558 - recall_5: 0.6880 -
val_loss: 0.6396 - val_accuracy: 0.7330 - val_precision_5: 0.7503 -
val_recall_5: 0.7076
Epoch 9/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.6020 - accuracy: 0.7584 - precision_5: 0.7750 - recall_5: 0.7341 -
val_loss: 0.6314 - val_accuracy: 0.7374 - val_precision_5: 0.7543 -
val_recall_5: 0.7104
Epoch 10/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.5704 - accuracy: 0.7782 - precision_5: 0.7902 - recall_5: 0.7598 -
val_loss: 0.6141 - val_accuracy: 0.7663 - val_precision_5: 0.7737 -
val_recall_5: 0.7522
Epoch 11/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.5344 - accuracy: 0.7983 - precision_5: 0.8080 - recall_5: 0.7847 -
val_loss: 0.5243 - val_accuracy: 0.8091 - val_precision_5: 0.8184 -
val_recall_5: 0.7960
Epoch 12/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.5129 - accuracy: 0.8106 - precision_5: 0.8191 - recall_5: 0.7979 -
val_loss: 0.5104 - val_accuracy: 0.8162 - val_precision_5: 0.8263 -
val_recall_5: 0.8032
Epoch 13/20
```

```
1755/1755 [==============================] - 12s 7ms/step - loss:
0.4990 - accuracy: 0.8174 - precision_5: 0.8268 - recall_5: 0.8035 -
val_loss: 0.5005 - val_accuracy: 0.8217 - val_precision_5: 0.8313 -
val_recall_5: 0.8083
Epoch 14/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.4873 - accuracy: 0.8240 - precision_5: 0.8348 - recall_5: 0.8118 -
val_loss: 0.5164 - val_accuracy: 0.8126 - val_precision_5: 0.8236 -
val_recall_5: 0.7967
Epoch 15/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.4786 - accuracy: 0.8284 - precision_5: 0.8378 - recall_5: 0.8148 -
val_loss: 0.4934 - val_accuracy: 0.8236 - val_precision_5: 0.8337 -
val_recall_5: 0.8108
Epoch 16/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.4692 - accuracy: 0.8335 - precision_5: 0.8427 - recall_5: 0.8220 -
val_loss: 0.4760 - val_accuracy: 0.8336 - val_precision_5: 0.8418 -
val_recall_5: 0.8232
Epoch 17/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.4612 - accuracy: 0.8352 - precision_5: 0.8450 - recall_5: 0.8230 -
val_loss: 0.4840 - val_accuracy: 0.8297 - val_precision_5: 0.8387 -
val_recall_5: 0.8177
Epoch 18/20
1755/1755 [==============================] - 12s 7ms/step - loss:
0.4452 - accuracy: 0.8443 - precision_5: 0.8524 - recall_5: 0.8332 -
val_loss: 0.4692 - val_accuracy: 0.8364 - val_precision_5: 0.8462 -
val_recall_5: 0.8240
Epoch 19/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.4430 - accuracy: 0.8460 - precision_5: 0.8541 - recall_5: 0.8365 -
val_loss: 0.4657 - val_accuracy: 0.8373 - val_precision_5: 0.8479 -
val_recall_5: 0.8252
Epoch 20/20
1755/1755 [==============================] - 13s 7ms/step - loss:
0.4396 - accuracy: 0.8479 - precision_5: 0.8560 - recall_5: 0.8376 -
val_loss: 0.4635 - val_accuracy: 0.8419 - val_precision_5: 0.8495 -
val_recall_5: 0.8315
```

Results and Analysis

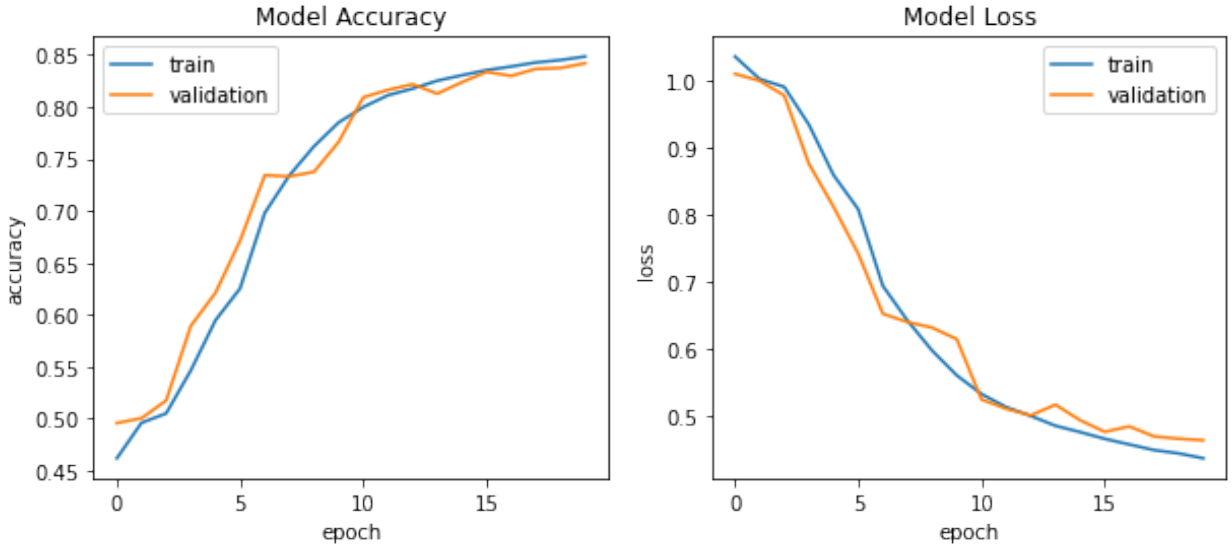
As expected, model version 3 shows significant improvement over the previous version. The single non-bidirectional LSTM layer enhances both training and validation accuracies, reaching up to 83.77% by the 20th epoch. Both training and validation accuracies improve consistently, with minimal discrepancy between them. This indicates that the model's predictions generalize well to unseen data, and there is no evident overfitting.

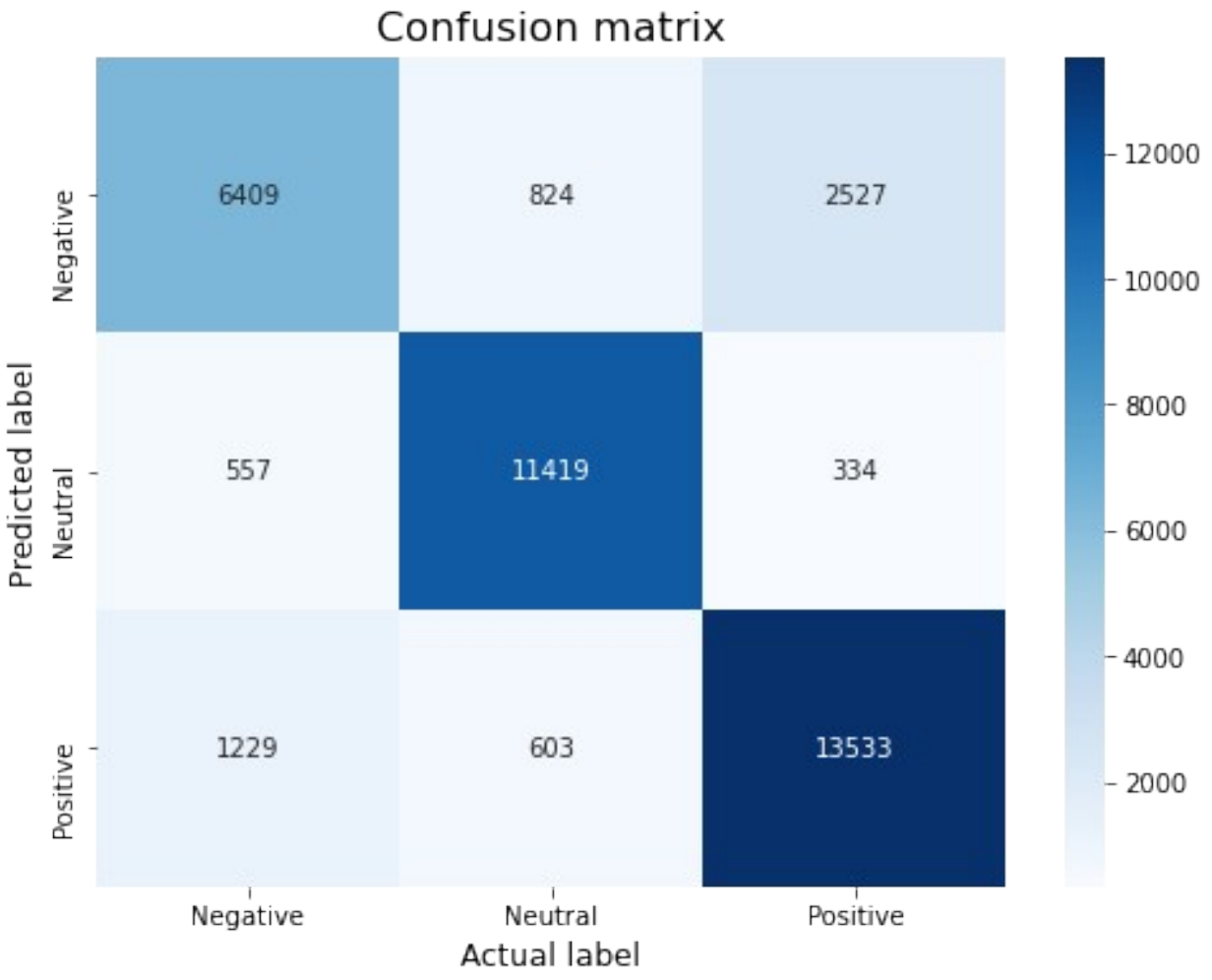
Moreover, the confusion matrix also demonstrates improvement, particularly in the accuracy of negative sentiment predictions compared to model version 2.

For version 4, the plan is to increase the complexity of the architecture by incorporating more advanced layers. While non-bidirectional LSTMs have produced significant results by allowing the model to capture long-term information flow in one direction, the logical next step is to explore bidirectional LSTMs in version 4. Bidirectional LSTMs enable the model to learn from information flowing in both directions, potentially capturing richer contextual dependencies. Additionally, layers such as Conv1D and Dropout will be added to further enrich the architecture.

```
#plot training performance over epochs
plot_training_hist(history3)

# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)
# Print metrics
print('Performance of version 1 model')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))

#plot confusion matrix
plot_confusion_matrix(model, X_test, y_test)

# Save the model architecture & the weights
#model.save('best_model0.h5') #original
model.save('version3.h5') #my
print('Version 3 saved')
```

```
Performance of version 1 model
Accuracy  : 0.8377
Precision : 0.8459
Recall    : 0.8279
F1 Score  : 0.8368
Version 3 saved
```


Model Accuracy
train
validation
accuracy
epoch
Model Loss
train
validation
loss
epoch


Confusion matrix
Predicted label
Negative
Neutral
Positive
6409
824
2527
557
11419
334
1229
603
13533
Negative
Neutral
Positive
Actual label
12000
10000
8000
6000
4000
2000

Building version 4:

My final attempt involves trying a bidirectional LSTM along with other layers to increase regularization in the model. The bidirectional LSTM helps the model learn information both before and after the current input data, providing a more comprehensive understanding of the sequence.

In addition to the bidirectional LSTM layer, I will add a one-dimensional convolution layer (Conv1D) and a max pooling layer before it. The Conv1D layer, effective for sequential data, captures local patterns and reduces noise in the input data. Together with max pooling, the Conv1D layer also reduces feature dimensionality and enhances the representation of the input sequence. To further minimize overfitting, a dropout layer will be added after the bidirectional LSTM.

Here is a summary of the layers used in this version:

- Conv1D layer to capture local patterns and reduce noise.
- MaxPooling layer to reduce feature dimensionality.
- Bidirectional LSTM layer to capture information from both directions.
- Dropout layer to reduce overfitting.

By incorporating these layers, we aim to create a more robust model that can better predict sentiment from the text data.

```
vocab_size = 5000
embedding_size = 32
epochs=20
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8

sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate,
nesterov=False)
# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(32)))
model.add(Dropout(0.4))
model.add(Dense(3, activation='softmax'))
# Compile the model
model.compile(optimizer=sgd, loss='categorical_crossentropy',
metrics=['accuracy', Precision(), Recall()])
# Show model
tf.keras.utils.plot_model(model, show_shapes=True)
print(model.summary())
# Compile model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
```

```
                metrics=['accuracy', Precision(), Recall()])
# Train model
batch_size = 64
history4 = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 50, 32)            160000
_________________________________________________________________
conv1d (Conv1D)              (None, 50, 32)            3104
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 25, 32)            0
_________________________________________________________________
bidirectional (Bidirectional (None, 64)                16640
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 195
=================================================================
Total params: 179,939
Trainable params: 179,939
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
1755/1755 [==============================] - 19s 9ms/step - loss:
1.0093 - accuracy: 0.4894 - precision_7: 0.5619 - recall_7: 0.2003 -
val_loss: 0.8981 - val_accuracy: 0.5735 - val_precision_7: 0.6466 -
val_recall_7: 0.3811
Epoch 2/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.8554 - accuracy: 0.5943 - precision_7: 0.6515 - recall_7: 0.4461 -
val_loss: 0.7259 - val_accuracy: 0.6744 - val_precision_7: 0.7074 -
val_recall_7: 0.6154
Epoch 3/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.6955 - accuracy: 0.6905 - precision_7: 0.7245 - recall_7: 0.6365 -
val_loss: 0.6116 - val_accuracy: 0.7566 - val_precision_7: 0.7878 -
val_recall_7: 0.7077
Epoch 4/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.5745 - accuracy: 0.7697 - precision_7: 0.7989 - recall_7: 0.7332 -
val_loss: 0.4921 - val_accuracy: 0.8247 - val_precision_7: 0.8467 -
val_recall_7: 0.7989
Epoch 5/20
1755/1755 [==============================] - 15s 8ms/step - loss:
```

```
0.4782 - accuracy: 0.8355 - precision_7: 0.8568 - recall_7: 0.8099 -
val_loss: 0.4339 - val_accuracy: 0.8582 - val_precision_7: 0.8714 -
val_recall_7: 0.8390
Epoch 6/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.4252 - accuracy: 0.8643 - precision_7: 0.8785 - recall_7: 0.8456 -
val_loss: 0.3970 - val_accuracy: 0.8728 - val_precision_7: 0.8833 -
val_recall_7: 0.8600
Epoch 7/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3892 - accuracy: 0.8814 - precision_7: 0.8920 - recall_7: 0.8674 -
val_loss: 0.3873 - val_accuracy: 0.8763 - val_precision_7: 0.8833 -
val_recall_7: 0.8643
Epoch 8/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3725 - accuracy: 0.8875 - precision_7: 0.8974 - recall_7: 0.8757 -
val_loss: 0.3631 - val_accuracy: 0.8886 - val_precision_7: 0.8966 -
val_recall_7: 0.8794
Epoch 9/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3504 - accuracy: 0.8960 - precision_7: 0.9050 - recall_7: 0.8848 -
val_loss: 0.3581 - val_accuracy: 0.8918 - val_precision_7: 0.8993 -
val_recall_7: 0.8821
Epoch 10/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3452 - accuracy: 0.8988 - precision_7: 0.9075 - recall_7: 0.8882 -
val_loss: 0.3492 - val_accuracy: 0.8947 - val_precision_7: 0.9015 -
val_recall_7: 0.8870
Epoch 11/20
1755/1755 [==============================] - 15s 8ms/step - loss:
0.3350 - accuracy: 0.9017 - precision_7: 0.9094 - recall_7: 0.8923 -
val_loss: 0.3447 - val_accuracy: 0.8949 - val_precision_7: 0.9012 -
val_recall_7: 0.8876
Epoch 12/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3320 - accuracy: 0.9030 - precision_7: 0.9104 - recall_7: 0.8943 -
val_loss: 0.3422 - val_accuracy: 0.8956 - val_precision_7: 0.9032 -
val_recall_7: 0.8883
Epoch 13/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3245 - accuracy: 0.9063 - precision_7: 0.9132 - recall_7: 0.8979 -
val_loss: 0.3401 - val_accuracy: 0.8977 - val_precision_7: 0.9038 -
val_recall_7: 0.8913
Epoch 14/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3211 - accuracy: 0.9075 - precision_7: 0.9152 - recall_7: 0.8999 -
val_loss: 0.3362 - val_accuracy: 0.8979 - val_precision_7: 0.9039 -
val_recall_7: 0.8909
Epoch 15/20
```

```
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3173 - accuracy: 0.9089 - precision_7: 0.9161 - recall_7: 0.9007 -
val_loss: 0.3341 - val_accuracy: 0.8989 - val_precision_7: 0.9051 -
val_recall_7: 0.8919
Epoch 16/20
1755/1755 [==============================] - 15s 8ms/step - loss:
0.3117 - accuracy: 0.9108 - precision_7: 0.9182 - recall_7: 0.9021 -
val_loss: 0.3309 - val_accuracy: 0.9005 - val_precision_7: 0.9063 -
val_recall_7: 0.8944
Epoch 17/20
1755/1755 [==============================] - 15s 8ms/step - loss:
0.3079 - accuracy: 0.9125 - precision_7: 0.9192 - recall_7: 0.9047 -
val_loss: 0.3296 - val_accuracy: 0.9011 - val_precision_7: 0.9073 -
val_recall_7: 0.8940
Epoch 18/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3038 - accuracy: 0.9132 - precision_7: 0.9201 - recall_7: 0.9053 -
val_loss: 0.3283 - val_accuracy: 0.9017 - val_precision_7: 0.9075 -
val_recall_7: 0.8956
Epoch 19/20
1755/1755 [==============================] - 15s 9ms/step - loss:
0.3009 - accuracy: 0.9152 - precision_7: 0.9215 - recall_7: 0.9081 -
val_loss: 0.3274 - val_accuracy: 0.9016 - val_precision_7: 0.9072 -
val_recall_7: 0.8956
Epoch 20/20
1755/1755 [==============================] - 15s 8ms/step - loss:
0.2969 - accuracy: 0.9149 - precision_7: 0.9214 - recall_7: 0.9077 -
val_loss: 0.3270 - val_accuracy: 0.9018 - val_precision_7: 0.9078 -
val_recall_7: 0.8954
```

Results and Analysis

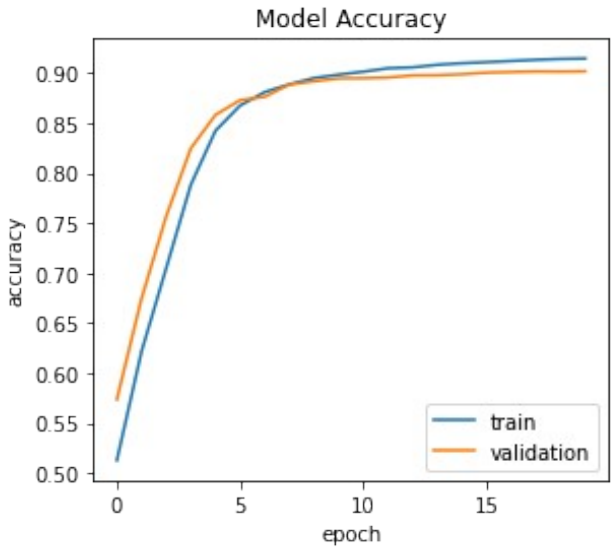
The final results of the version 4 model are very promising. The overall validation accuracy reaches 90%. Both the training and validation learning curves appear smooth with minimal gap between them, indicating that overfitting is not an issue. Additionally, the learning curves flatten out after about 10 epochs, making the choice to stop training at 20 epochs reasonable.

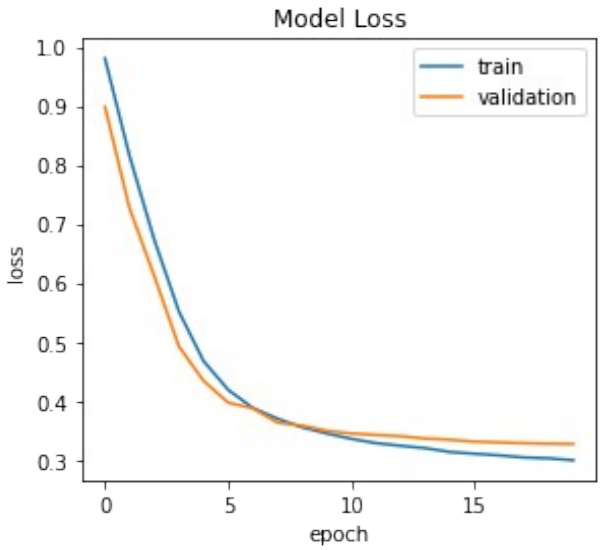
The confusion matrix shows that the model effectively makes true positive predictions for all three labels: positive, neutral, and negative. This indicates a well-balanced model capable of accurately predicting sentiment across different classes.

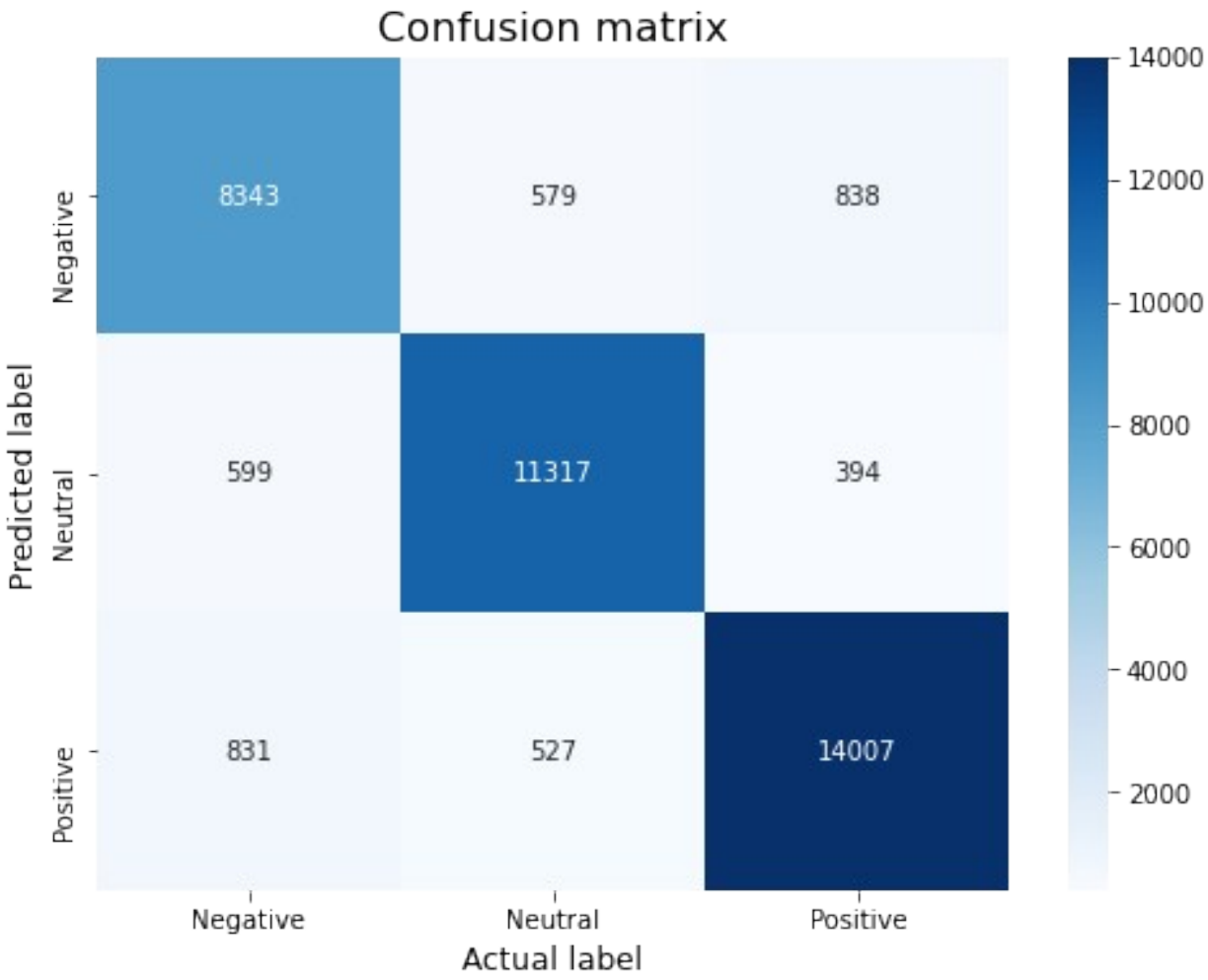
```
#plot training performance over epochs
plot_training_hist(history4)

# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)
```

```python
# Print metrics
print('Performance of version 1 model')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))

#plot confusion matrix
plot_confusion_matrix(model, X_test, y_test)

# Save the model architecture & the weights
#model.save('best_model0.h5') #original
model.save('version4.h5') #my
print('Version 4 saved')
```

```
Performance of version 1 model
Accuracy  : 0.8993
Precision : 0.9054
Recall    : 0.8933
F1 Score  : 0.8993
Version 4 saved
```








```
from keras.models import load_model

# Load model
model = load_model('version4.h5')

def predict_class(text):
    sentiment_classes = ['Negative', 'Neutral', 'Positive']
    max_len=50
    # Transforms text to a sequence of integers using a tokenizer
object
    xt = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    # Do the prediction using the loaded model
    yt = model.predict(xt).argmax(axis=1)
    # Print the predicted sentiments
    print('The predicted sentiment is', sentiment_classes[yt[0]])

predict_class(['"I hate when I have to call and wake people up'])
```

```
The predicted sentiment is Negative
```

```
predict_class(['The food was meh'])
```

```
The predicted sentiment is Neutral
```

```
predict_class(['He is a best minister india ever had seen'])
```

```
The predicted sentiment is Positive
```

Conclusion

The following chart summarizes the results of the four models I built. The last model, version 4, clearly outperforms the others. All key metrics—accuracy, precision, recall, and F1 score—reach around 90%. Here are the major learnings from this problem:

- SimpleRNN without much regularization produces a very poor model: The initial model, which used SimpleRNN, performed poorly due to a lack of capacity to capture long-term dependencies and insufficient regularization.
- LSTM elevates the model performance significantly: Introducing LSTM layers made a substantial difference in processing text data like tweets, which typically have a length within 50 words.
- Bi-directional LSTM enhances the model further: Incorporating bi-directional LSTM, along with additional layers for dimensionality reduction and regularization, further improved model performance.

If I were to continue the model development, I would add multiple rounds of bi-directional LSTM layers wrapped with Conv1D and dropout layers. A more complex RNN architecture might push the model accuracy rate further into the 90+ range.

```
from keras.models import load_model
import matplotlib.pyplot as plt
import numpy as np

def evaluate_model(model, X_test, y_test):
    # Evaluate model on the test set
    loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)

    # Calculate F1 Score
    if precision + recall == 0:
        f1 = 0.0
    else:
        f1 = 2 * (precision * recall) / (precision + recall)
    #f1 = 2 * (precision * recall) / (precision + recall)

    # Print metrics
    #print('')
```

```
    #print('Accuracy  : {:.4f}'.format(accuracy))
    #print('Precision : {:.4f}'.format(precision))
    #print('Recall    : {:.4f}'.format(recall))
    #print('F1 Score  : {:.4f}'.format(f1))

    return accuracy, precision, recall, f1

# Function to load models
def load_models(start, end, base_name='version'):
    models = []
    for i in range(start, end + 1):
        model = load_model(f'{base_name}{i}.h5')
        #print(f'load model {i}')
        models.append(model)
    return models

# Load models
firstmodel = 1
lastmodel = 4
models = load_models(firstmodel, lastmodel)  # Change the range as
needed
model_names = [f'Model {i}' for i in range(1, lastmodel+1)]

# Store results
results = {
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': []
}

# Evaluate each model and store results
for model in models:
    accuracy, precision, recall, f1 = evaluate_model(model, X_test,
y_test)
    results['Accuracy'].append(accuracy)
    results['Precision'].append(precision)
    results['Recall'].append(recall)
    results['F1 Score'].append(f1)

# Plotting
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
num_metrics = len(metrics)
bar_width = 0.2
index = np.arange(len(models))

fig, ax = plt.subplots(figsize=(20, 6))  # Adjust the width to make it
twice as wide

# Create bars for each metric
```

```
for i, metric in enumerate(metrics):
    bar_positions = index + i * bar_width
    bars = ax.bar(bar_positions, results[metric], bar_width,
label=metric)

    # Add y-values on each bar
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2.0, yval,
f'{yval:.4f}', ha='center', va='bottom')

# Set the labels and title
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_title('Model Evaluation Metrics')
ax.set_xticks(index + bar_width * (num_metrics / 2 - 0.5))
ax.set_xticklabels(model_names)
ax.set_ylim(0, 1)  # Set y-axis scale from 0.6 to 1
ax.legend(loc='upper left', bbox_to_anchor=(1, 1))  # Place legend
outside the plot area

# Show the plot
plt.tight_layout()
plt.show()
```