George-Chira / twitter-sentiment-analysis Public

Code Issues Pull requests Actions Projects Security Insights

twitter-sentiment-analysis / Final.ipynb

George-Chira final notebook revision d68c65f · 14 minutes ago History

Preview Code Blame 2955 lines (2955 loc) · 369 KB

Raw file content

Download Ctrl shift s

Copy path Ctrl shift .

Copy permalink Ctrl shift ,

View options

- [x] Show code folding buttons
- [ ] Wrap lines
- [ ] Center content
- [x] Open symbols on click

SentimentFlow: Twitter Sentiment for Apple and Google Products

Executive Summary

This project leverages machine learning to analyze Twitter sentiment about Apple and Google products, classifying tweets as positive, negative, or neutral. The data science process involved cleaning over 9,000 tweets, addressing missing values and class imbalance, and transforming text data into numerical features using TF-IDF and CountVectorizer. Preprocessing steps like tokenization and lemmatization enhanced text quality, while SMOTE balanced the dataset for model training.

Several algorithms, including Logistic Regression and Random Forest, were evaluated, with hyperparameter tuning applied to optimize performance. The best models achieved over 83% accuracy, with TF-IDF vectorization providing the strongest feature representation. The results offer valuable insights into public sentiment, aiding companies in making data-driven decisions.

Business Understanding

Overview

SentimentFlow aims to address a real-world problem related to understanding public sentiment towards Apple and Google products on Twitter. The stakeholders include companies, marketing teams, and decision-makers who want to gauge public opinion and make informed strategic decisions based on social media sentiment.

Problem Statement

The problem is to accurately classify the sentiment of tweets related to Apple and Google products. We want to determine whether a tweet expresses a positive, negative, or neutral sentiment. This classification can help companies understand customer satisfaction, identify potential issues, and tailor their responses accordingly.

Stakeholders

Companies (Apple and Google): These organizations are directly impacted by public sentiment. They want to monitor how their products are perceived and identify areas for improvement.

Marketing Teams: Marketing teams can use sentiment analysis to adjust their campaigns, respond to negative feedback, and highlight positive aspects of their products.

Decision-Makers: Executives and managers need insights into public sentiment to make informed decisions about product development, customer support, and brand reputation.

Value Proposition By accurately classifying tweets, our NLP model can provide actionable insights to stakeholders. For example:

Identifying negative sentiment can help companies address issues promptly. Recognizing positive sentiment can guide marketing efforts and

reinforce successful strategies. Understanding neutral sentiment can provide context and balance. Objectives Main Objective

To develop a NLP (Natural Language Processing) multiclass classification model for sentiment analysis, aim to achieve a recall score of 80% and an accuracy of 80%. The model should categorize sentiments into three classes: Positive, Negative, and Neutral.

Specific Objectives

To idenitfy the most common words used in the dataset using Word cloud.

To confirm the most common words that are positively and negatively tagged.

To recognize the products that have been opined by the users.

To spot the distribution of the sentiments.

Conclusion

Our NLP model will contribute valuable insights to the real-world problem of understanding Twitter sentiment about Apple and Google products. Stakeholders can leverage this information to enhance their decision-making processes and improve overall customer satisfaction.

Data Understanding

Data Sources

The dataset originates from CrowdFlower via data.world. Contributors evaluated tweets related to various brands and products. Specifically:

Each tweet was labeled as expressing positive, negative, or no emotion toward a brand or product.

If emotion was expressed, contributors specified which brand or product was the target.

Suitability of Data

Here's why this dataset is suitable for our project:

Relevance: The data directly aligns with our business problem of understanding Twitter sentiment for Apple and Google products.

Real-World Context: The tweets represent actual user opinions, making the problem relevant in practice.

Multiclass Labels: We can build both binary (positive/negative) and multiclass (positive/negative/neutral) classifiers using this data.

Dataset Size

The dataset contains over 9,000 labeled tweets. We'll explore its features to gain insights.

Descriptive Statistics

tweet_text: The content of each tweet.

is_there_an_emotion_directed_at_a_brand_or_product: No emotion toward brand or product, Positive emotion, Negative emotion, I can't tell

emotion_in_tweet_is_directed_at: The brand or product mentioned in the tweet.

Feature Inclusion

Tweet text is the primary feature. The emotion label and target brand/product are essential for classification.

Limitations

Label Noise: Human raters' subjectivity may introduce noise. Imbalanced Classes: We'll address class imbalance during modeling. Contextual Challenges: Tweets are often short and context-dependent. Incomplete & Missing Data: Could affect the overall performance of the models.

Data Loading

Import necessary modules

In [ ]:

```
# Data manipulation
import pandas as pd
import numpy as np

# plotting
import seaborn as sns
#from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from nltk.stem import WordNetLemmatizer


# Download required NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
# sklearn
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import BernoulliNB
from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,cross_val_score

from sklearn.naive_bayes import MultinomialNB

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, recall_score

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# wordCloud
from wordcloud import WordCloud

# pickle
import pickle

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package stopwords to C:\Users\WILSON
[nltk_data]     MACHOKA\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\WILSON
[nltk_data]     MACHOKA\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\WILSON MACHOKA\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Create Class

In [ ]:

```
class DataUnderstanding():
    """Class that gives the data understanding of a dataset"""
    def __init__(self, data='None'):
        """Initialisation"""
        self.df = data

    def load_data(self, path):
        """Loading the data"""
        if self.df == 'None':
```

```
            self.df = pd.read_csv(path, encoding='latin-1')
        return self.df

    def understanding(self):
        # Info
        print("""INFO""")
        print("-"*4)
        self.df.info()

        # Shape
        print("""\n\nSHAPE""")
        print("-"*5)
        print(f"Records in dataset are {self.df.shape[0]} with {self.df.shape[1]} columns.")

        # Columns
        print("\n\nCOLUMNS")
        print("-"*6)
        print(f"Columns in the dataset are:")
        for idx in self.df.columns:
            print(f"- {idx}")

        # Unique Values
        print("\n\nUNIQUE VALUES")
        print("-"*12)
        for col in self.df.columns:
            print(f"Column *{col}* has {self.df[col].nunique()} unique values")
            if self.df[col].nunique() < 12:
                print(f"Top unique values in the *{col}* include:")
                for idx in self.df[col].value_counts().index:
                    print(f"- {idx}")
            print("")

        # Missing or Null Values
        print("""\nMISSING VALUES""")
        print("-"*15)
        for col in self.df.columns:
            print(f"Column *{col}* has {self.df[col].isnull().sum()} missing values.")

        # Duplicate Values
        print("""\n\nDUPLICATE VALUES""")
        print("-"*16)
        print(f"The dataset has {self.df.duplicated().sum()} duplicated records.")
```

Load data

In [ ]:

```
# Load the dataset
data = DataUnderstanding()
df = data.load_data('C:/Users/WILSON MACHOKA/Desktop/phase_4/twitter-sentiment-analysis/data/judge-1377884607_tweet_product_c
df.head()
```

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

Data Understanding

In [ ]:

```python
import pandas as pd

class DataUnderstanding:
    def __init__(self):
        self.df = None  # Initialize df as None

    def load_data(self, path):
        """Loading the data"""
        if self.df is None:  # Check if df is None
            self.df = pd.read_csv(path, encoding='latin1')  # Load DataFrame
        return self.df

    def understanding(self):
        """Display information about the DataFrame"""
        print("INFO")
        print("-" * 4)
        self.df.info()  # Display DataFrame info

        # Shape
        print("Shape:", self.df.shape)

        # First few rows
        print("First few rows:")
        print(self.df.head())

        # Check for missing values
        missing_values = self.df.isnull().sum()
        print("\nMissing Values:")
        print(missing_values[missing_values > 0])  # Only show columns with missing values

        # Check for duplicates
        duplicates = self.df.duplicated().sum()
        print("\nNumber of duplicate rows:", duplicates)

# Load the dataset
data = DataUnderstanding()
df = data.load_data('C:/Users/WILSON MACHOKA/Desktop/phase_4/twitter-sentiment-analysis/data/judge-1377884607_tweet_product_c
```

```
# Call the understanding method
data.understanding()
```

```
INFO
----
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column                                              Non-Null Count  Dtype
---  ------                                              --------------  -----
 0   tweet_text                                          9092 non-null   object
 1   emotion_in_tweet_is_directed_at                     3291 non-null   object
 2   is_there_an_emotion_directed_at_a_brand_or_product  9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
Shape: (9093, 3)
First few rows:
                                          tweet_text  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe...
1  @jessedee Know about @fludapp ? Awesome iPad/i...
2  @swonderlin Can not wait for #iPad 2 also. The...
3  @sxsw I hope this year's festival isn't as cra...
4  @sxtxstate great stuff on Fri #SXSW: Marissa M...

  emotion_in_tweet_is_directed_at  \
0                          iPhone
1              iPad or iPhone App
2                            iPad
3              iPad or iPhone App
4                          Google

  is_there_an_emotion_directed_at_a_brand_or_product
0                                   Negative emotion
1                                   Positive emotion
2                                   Positive emotion
3                                   Negative emotion
4                                   Positive emotion

Missing Values:
tweet_text                            1
emotion_in_tweet_is_directed_at    5802
dtype: int64

Number of duplicate rows: 22
```

COMMENTS

All the columns are in the correct data types.

The columns will need to be renamed.

Features with missing values should be renamed from NaN.

Duplicate records should be dropped.

All records with the target as "I can't tell" should be dropped.

Corrupted records should be removed.

Data Cleaning

Validity

Corrupted Records in the *tweet_text Column*

The data seems to be corrupted in some records. To correct this issue, we create a function that can find these corrupted records, and returns their indexes. We use this index to remove the record from the working dataframe.

In [ ]:
```
# A function targeting corrupted records
def is_corrupted(tweet):
    """This func returns the index of any record that is corrupted"""
    corrupted_cols = []
    for key, text in enumerate(tweet):
        if any(ord(char) > 127 for char in str(text)) == True:
            corrupted_cols.append(key)
    return corrupted_cols
```

In [ ]:
```
# Applying the is_corrupted function to find the indexes of the corrupted records
corrupted_records_idx = is_corrupted(df['tweet_text'])
```

In [ ]:
```
# Test to check if the function worked as intended
df.loc[corrupted_records_idx]['tweet_text'].values[0]
```

```
'@mention  - False Alarm: Google Circles Not Coming Now\x89ÛÒand Probably Not Ever? - {link} #Google #Circles #Social #SXSW'
```

In [ ]:
```
# Drop these records
df.drop(index=corrupted_records_idx, inplace=True)
```

In [ ]:
```
# Test to ensure there are no corrupted records left
is_corrupted(df['tweet_text'])
```

```
[]
```

Drop Records in the *is_there_an_emotion_directed_at_a_brand_or_product column where the value is "I can't tell"*

In [ ]:

```
# Identification of the record
bad_reaction_idx = df[df['is_there_an_emotion_directed_at_a_brand_or_product'] == "I can't tell"].index

# Drop the columns
df.drop(index = bad_reaction_idx, inplace=True)

# Test
df[df['is_there_an_emotion_directed_at_a_brand_or_product'] == "I can't tell"]
```

tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product

Replace Fields in the *is_there_an_emotion_directed_at_a_brand_or_product column where the value is "No emotion toward brand or product" to "Neutral emotion"*

In [ ]:
```
# Identification of the record
neutral_reaction_idx = df[df['is_there_an_emotion_directed_at_a_brand_or_product'] ==\
                          "No emotion toward brand or product"].index

# Replace the values
df.loc[neutral_reaction_idx, 'is_there_an_emotion_directed_at_a_brand_or_product'] = "Neutral emotion"

# Test
df[df['is_there_an_emotion_directed_at_a_brand_or_product'] == "No emotion toward brand or product"]
```

tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product

Completeness

Drop Missing Values in the *tweet_text column*

In [ ]:
```
tweet_missing = df[df['tweet_text'].isnull() == True].index
df.loc[tweet_missing]
```

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product
6	NaN	NaN	Neutral emotion

In [ ]:
```
# Drop the record
df.drop(index=tweet_missing, inplace=True)
```

In [ ]:
```
# Check
df[df['tweet_text'].isnull() == True]
```

tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product

Fill Missing Values in the *emotion_in_tweet_is_directed_at column*

In [ ]:
```
# Find the records with missing values in the 2nd column
df[df['emotion_in_tweet_is_directed_at'].isnull() == True].shape[0]
```

```
5331
```

In [ ]:
```
# List of unique products/ services
products = list(df.emotion_in_tweet_is_directed_at.unique())
products.remove(np.nan) # Removes any np.nan items

def find_product(tweet):
    """This func takes in a tweet and returns the product talked about in the
    tweet; used to fill in the emotion_in_tweet_is_directed_at column"""
    for product in products:
        if str(product) in tweet or str(product).upper() in tweet \
            or  str(product).lower() in tweet or str(product).title() in tweet:
            return product

# Applying the function to find the index of records with missing values in the 2nd column
missing_products_idx = df[df['emotion_in_tweet_is_directed_at'].isnull() == True].index
```

In [ ]:
```
# Replace the field where there are missing values in the emotion_in_tweet_is_directed_at column
df.loc[missing_products_idx, 'emotion_in_tweet_is_directed_at'] = df.loc[missing_products_idx, 'tweet_text']\
                                                                  .apply(lambda x: find_product(x))
```

In [ ]:
```
# In case any field was not captured by our function, we can change it to 'None'
none_index = df[df['emotion_in_tweet_is_directed_at'].isnull()].index
df.loc[none_index, 'emotion_in_tweet_is_directed_at'] = 'None'
# df.loc[none_index]
```

In [ ]:
```
# Check
df['emotion_in_tweet_is_directed_at'].value_counts()
```

```
iPad                               2273
Google                             1984
Apple                              1269
iPhone                             1093
None                                720
iPad or iPhone App                  448
Android                             284
Other Google product or service     278
Android App                          77
Other Apple product or service       33
Name: emotion_in_tweet_is_directed_at, dtype: int64
```

In [ ]:
```
# Number of values in the column are the same as the length of the data
np.sum(df['emotion_in_tweet_is_directed_at'].value_counts().values) == df.shape[0]
```

```
True
```

COMMENTS

Noting that the missing data consists of over 5000 rows, this represent a significant proportion of the data if we dropped this data.

To counter this problem, knowing that there are limited observable products in the 2nd column, we can read each tweet and, find and replace the missing value with the relevant product the tweet talks about.

In the end, we were able to assign all tweets to a product and only 720 were not talking about a product explicitly.

Consistency

Drop the duplicates

In [ ]:
```
# Dropping the duplicates
df.drop_duplicates(inplace=True)
```

In [ ]:
```
# Check if there is any remaining duplicate values
df.duplicated().sum()
```

```
0
```

Uniformity

Rename the columns

In [ ]:
```
# Change the column names
df.rename(columns={'tweet_text': "tweet",
                   'emotion_in_tweet_is_directed_at':"product",
                   'is_there_an_emotion_directed_at_a_brand_or_product': "emotion"},
          inplace=True)
```

In [ ]:
```
# Check
df.columns
```

```
Index(['tweet', 'product', 'emotion'], dtype='object')
```

Reset the Index of the dataframe

In [ ]:
```
# Reset the index
df.reset_index(inplace=True)
# Drop the old index column
df.drop(labels='index', axis=1, inplace=True)
df
```

	tweet	product	emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion
...	...	...	...
8434	@mention Yup, but I don't have a third app yet...	Android	Neutral emotion
8435	Ipad everywhere. #SXSW {link}	iPad	Positive emotion
8436	Wave, buzz... RT @mention We interrupt your re...	Google	Neutral emotion
8437	Google's Zeiger, a physician never reported po...	Google	Neutral emotion
8438	Some Verizon iPhone customers complained their...	iPhone	Neutral emotion

8439 rows × 3 columns

Conclusion

To ensure that the cleaning process worked efficiently, we can access the DataUnderstanding class to perform final checks before proceeding to the next section.

In [ ]:
```
class DataUnderstanding:
    def __init__(self, df):
        # Store the dataframe as an instance variable
        self.df = df

    def understanding(self):
        # Print some basic understanding of the data
        print("Data Overview:")
        print(self.df.head())  # Print first 5 rows

        print("\nData Info:")
        print(self.df.info())  # Print info about data types and null values
```

```
        print("\nData Description:")
        print(self.df.describe())  # Print summary statistics of numerical columns

        print("\nMissing Values:")
        print(self.df.isnull().sum())  # Print count of missing values in each column
```

In [ ]:
```
DataUnderstanding(df).understanding()
```

```
Data Overview:
                                                tweet             product  \
0  .@wesley83 I have a 3G iPhone. After 3 hrs twe...              iPhone
1  @jessedee Know about @fludapp ? Awesome iPad/i...  iPad or iPhone App
2  @swonderlin Can not wait for #iPad 2 also. The...               iPad
3  @sxsw I hope this year's festival isn't as cra...  iPad or iPhone App
4  @sxtxstate great stuff on Fri #SXSW: Marissa M...              Google

            emotion
0  Negative emotion
1  Positive emotion
2  Positive emotion
3  Negative emotion
4  Positive emotion

Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8439 entries, 0 to 8438
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   tweet    8439 non-null   object
 1   product  8439 non-null   object
 2   emotion  8439 non-null   object
dtypes: object(3)
memory usage: 197.9+ KB
None

Data Description:
                                          tweet product          emotion
count                                      8439    8439             8439
unique                                     8434      10                3
top     Win free iPad 2 from webdoc.com #sxsw RT    iPad  Neutral emotion
freq                                          2    2272             5071

Missing Values:
tweet      0
product    0
emotion    0
dtype: int64
```

Data Visualization

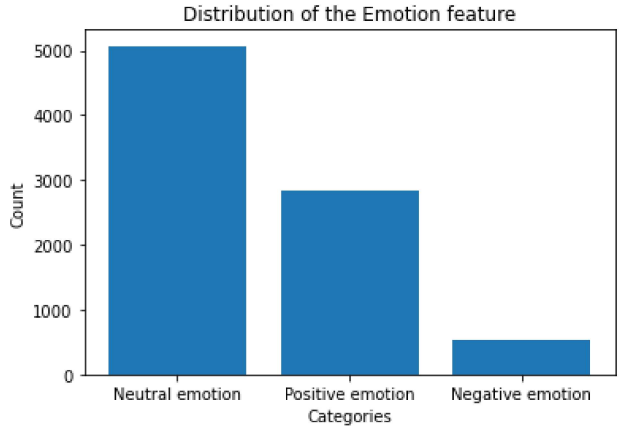
In [ ]:
```
print(df.columns)
```

```
Index(['tweet', 'product', 'emotion'], dtype='object')
```

In [ ]:
```
def plot_bar(feature, plot='bar'):
    """This func returns a bar or a barh plot"""
    if plot == 'bar':
        labels = df[feature].value_counts().index
        values = df[feature].value_counts().values
        plt.bar( x=labels,height=values)
        plt.ylabel("Count")
        plt.xlabel("Categories")
    else:
        labels = df[feature].value_counts(ascending=True).index
        values = df[feature].value_counts(ascending=True).values
        plt.barh(width=values, y=labels)
        plt.xlabel("Count")
        plt.ylabel("Categories")
    plt.title(f"Distribution of the {feature.title()} feature");
```

Distribution of the *emotion feature*

In [ ]:
```
plot_bar('emotion')
```

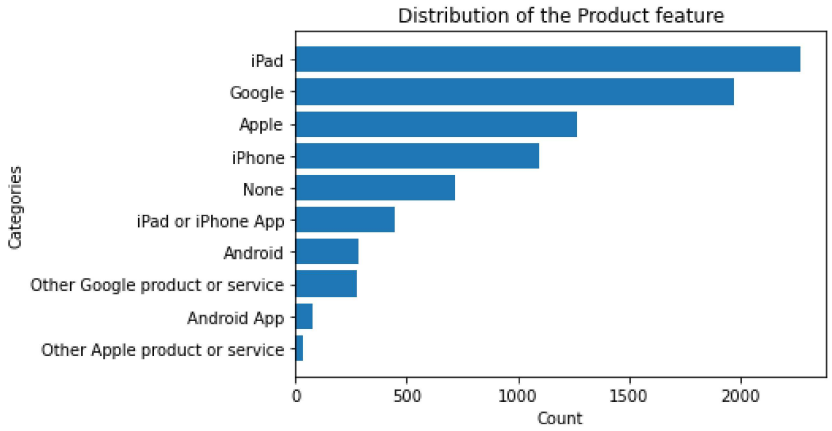


Observation

The distribution shows a huge class imbalance between categories. The Neutral Emotion category holds over 50% of the data.

Distribution of the *product feature*

In [ ]:
```
plot_bar('product', plot='barh')
```



Observation

The data reveals there was a lot of sentiments concerning the iPad in the period, followed closely by the Google and Apple company.

The data has a 'None' category meaning that it did not concern the Apple of Google products originally set out at the start of the project.

Text Preprocessing

Text Processing includes steps like removing punctuation, tokenization (splitting text into words or phrases), converting text to lowercase, removing stop words (common words that add little value), and stemming or lemmatization (reducing words to their base forms)

In [ ]:

```
# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words('english') + ['sxsw', 'sxswi', 'link', 'rt'])
lemmatizer = WordNetLemmatizer()

def lemmatize_tweet(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove mentions
    text = re.sub(r'@\w+', '', text)
    # Remove hashtags (keep the text after the #)
    text = re.sub(r'#', '', text)
    # Remove special characters like """
    text = re.sub(r'&\w+;', '', text)
    # Remove punctuation
    text = "".join([char for char in text if char not in string.punctuation])
    # Tokenize text
    tokens = re.split('\W+', text.lower())
    # Remove stopwords and lemmatize the tokens
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    # Remove extra white spaces
    tokens = [word.strip() for word in tokens if word.strip() != '']
```

```
    # Remove numbers
    tokens = [word for word in tokens if not word.isdigit()]
    # Tag parts of speech
    pos_tags = nltk.pos_tag(tokens)
    # Filter tokens to retain only nouns, adjectives, verbs, and adverbs
    important_pos = {'NN', 'NNS', 'NNP', 'NNPS', 'JJ', 'JJR', 'JJS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ', 'RB', 'RBR', 'R
    tokens = [word for word, tag in pos_tags if tag in important_pos]
    return tokens

# Apply the clean_text function to the 'tweet' column
df['lemmatized_tweet'] = df['tweet'].apply(lambda x: lemmatize_tweet(x))

def join_text(tokens):
    """
    This function takes a list of tokens (words) and returns them as a single string.
    Each token is separated by a space.

    Parameters:
    tokens (list of str): A list of tokens to be joined.

    Returns:
    str: The tokens joined into a single string separated by spaces.
    """
    return " ".join(tokens)

df['clean_tweet'] = df['lemmatized_tweet'].apply(lambda x: join_text(x))

# Print the cleaned tweets
df.head()
```

	tweet	product	emotion	lemmatized_tweet	clean_tweet
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion	[iphone, hr, tweeting, riseaustin, dead, need,...	iphone hr tweeting riseaustin dead need upgrad...
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion	[know, awesome, ipadiphone, app, youll, likely...	know awesome ipadiphone app youll likely appre...
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion	[wait, ipad, also, sale]	wait ipad also sale
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	[hope, year, festival, isnt, crashy, year, iph...	hope year festival isnt crashy year iphone app
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion	[great, stuff, fri, mayer, google, tim, oreill...	great stuff fri mayer google tim oreilly tech ...

Visual of Lematized tweets

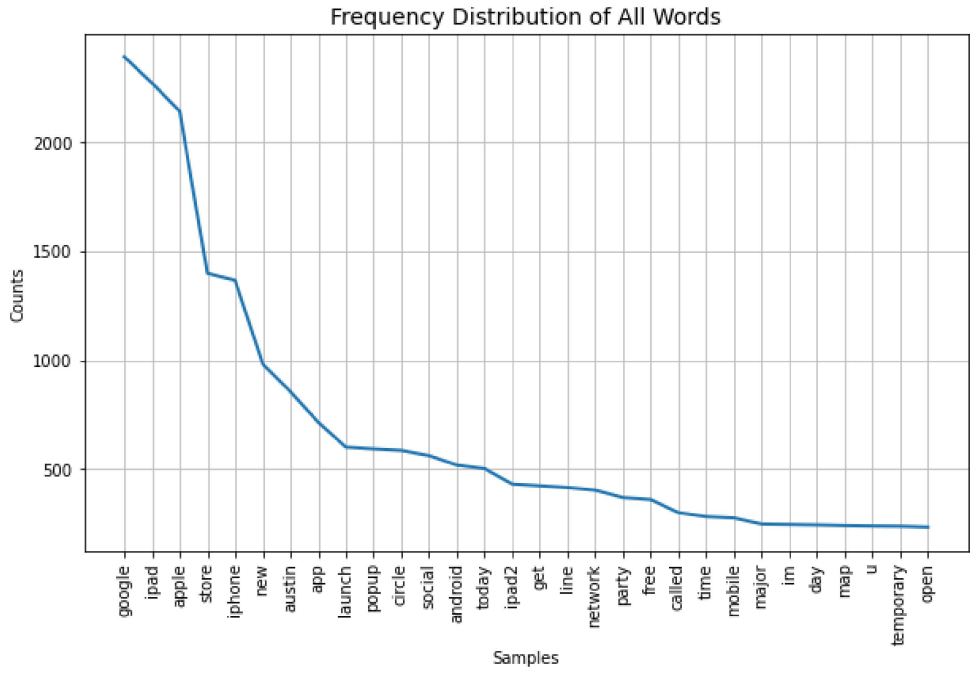
In [ ]:

```
def plot_fdist(sentiment=None, title="Frequency Distribution of All Words", df=df):
    """
```

```
    This func creates a Frequency Distribution plot depending on the sentiment chosen
    """

    if sentiment == None:
        lemmatized_tweet = df['lemmatized_tweet']

        # Flatten the list
        flattened_lemmatized_tweet = [token for sublist in lemmatized_tweet for token in sublist]

    elif sentiment != None:
        lemmatized_tweet = df[df['emotion'] == sentiment]['lemmatized_tweet']

        # Flatten the list
        flattened_lemmatized_tweet = [token for sublist in lemmatized_tweet for token in sublist]

    # Create the frequency distribution
    fdist = FreqDist(flattened_lemmatized_tweet)

    # Plot the frequency distribution
    plt.figure(figsize=(10,6))
    plt.title(title, fontsize=14)
    fdist.plot(30);
```

In [ ]:

```
plot_fdist()
```

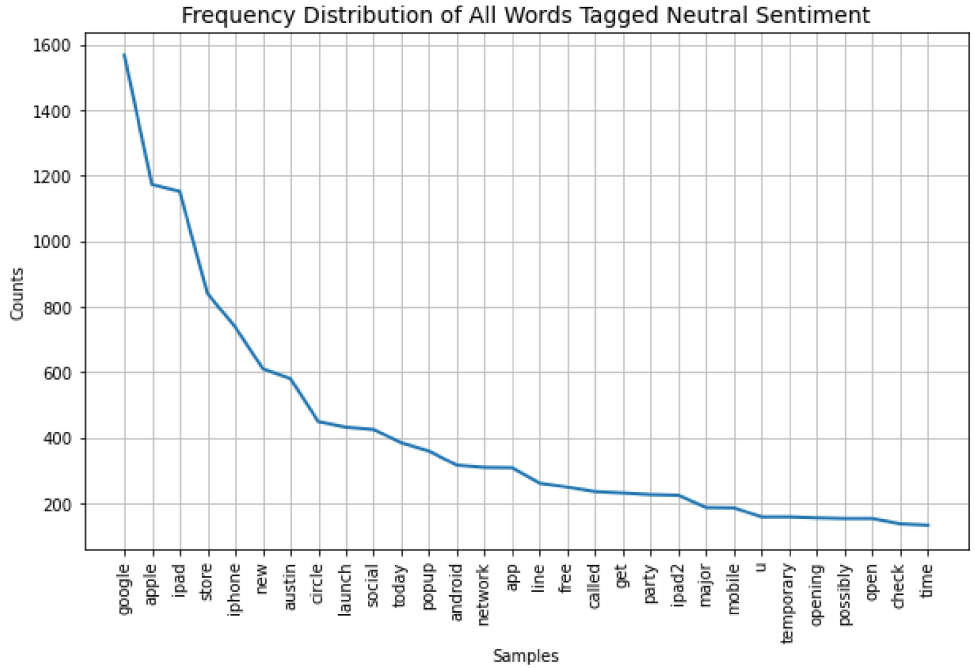


Observation

With respect to all the data, the words 'google', 'apple', 'ipad' and 'store' appeared more frequently than all other words.

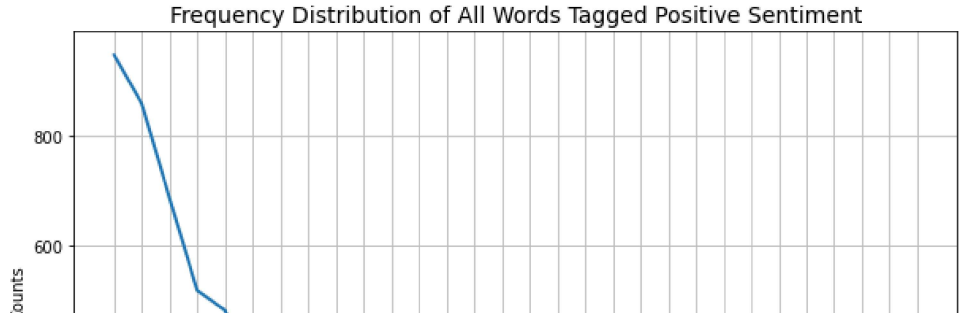
Frequency Distribution of Lemmatized words categorized as Neutral Emotion

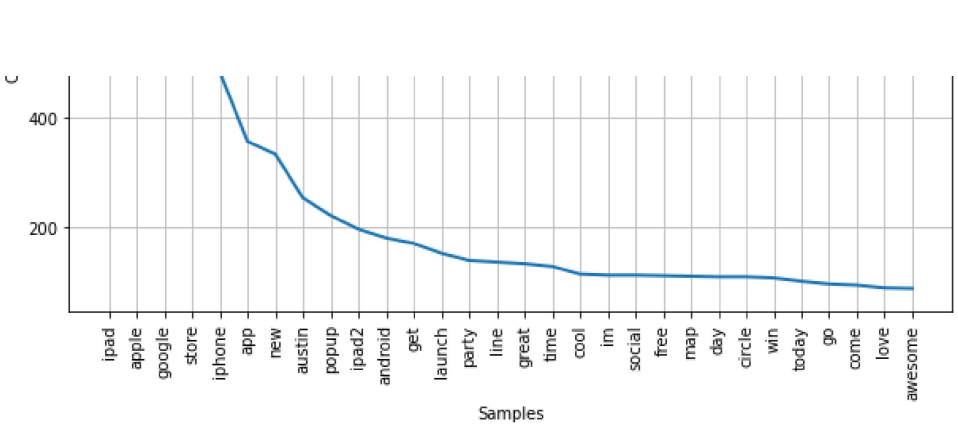
In [ ]:
```
plot_fdist(sentiment="Neutral emotion", title="Frequency Distribution of All Words Tagged Neutral Sentiment")
```



Frequency Distribution of Lemmatized words categorized as Positive Emotion

In [ ]:
```
plot_fdist(sentiment="Positive emotion", title="Frequency Distribution of All Words Tagged Positive Sentiment")
```





Observation

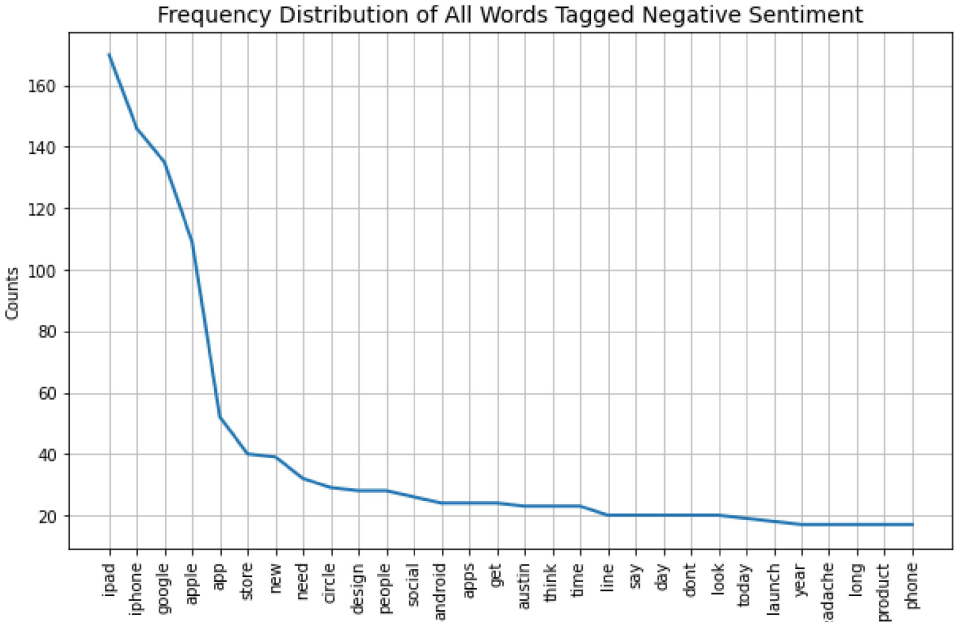
With respect to all the data categorised as 'positive', the words 'ipad', 'apple', 'google' and 'store' appeared more frequently than all other words.

Other key positive words introduced in this section include "awesome", "love", "win", "cool", "great", "party"

But were less than the counts recorded in the Neutral Frequency Distributions.

Frequency Distribution of Lemmatized words categorized as Negative Emotion

In [ ]:
```
plot_fdist(sentiment="Negative emotion", title="Frequency Distribution of All Words Tagged Negative Sentiment")
```



Samples

Observation

With respect to all the data categorised as 'negative', the words 'ipad', 'iphone', 'google' and 'apple' appeared more frequently than all other words. But were less than the counts recorded in the Neutral Frequency Distributions.

Bigrams

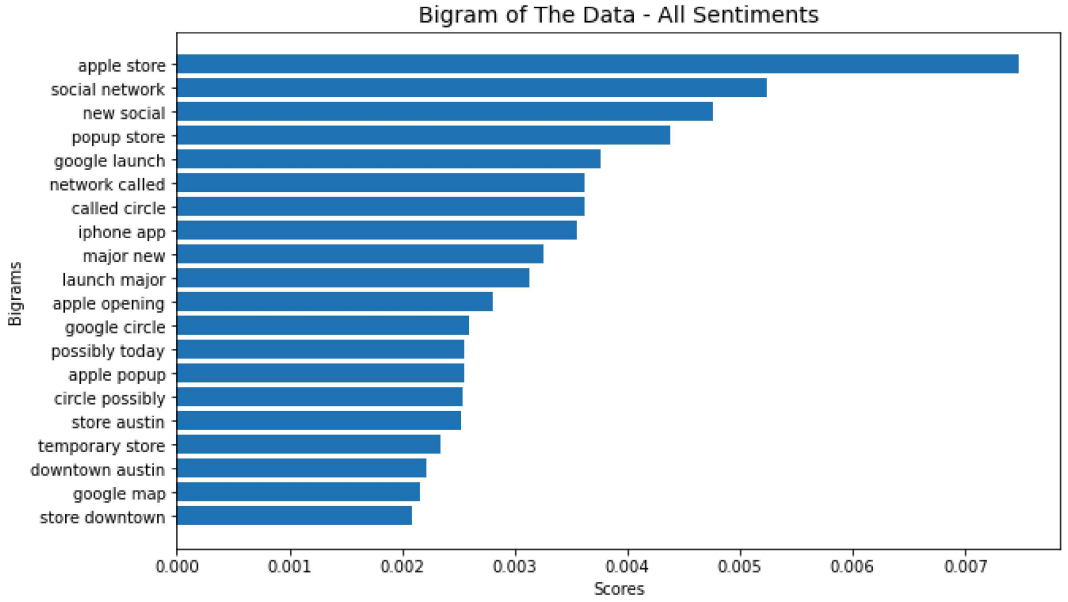
Analyzing bigrams improves context understanding by considering pairs of words together, which helps in understanding phrases that might have specific meanings different from their individual words.

In [ ]:
```
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures

# Alias for BigramAssocMeasures
bigram_measures = BigramAssocMeasures()
```

In [ ]:
```
def bigram_plots(sentiment=None, title="Bigram of The Data - All Sentiments", df=df, items=20):
    """
    This function returns a horizontal plot of the highest scored bigrams in the dataset
    """
    if sentiment == None:
        lemmatized_tweet = df['lemmatized_tweet']

        # Flatten the list
        flattened_lemmatized_tweet = [token for sublist in lemmatized_tweet for token in sublist]

    elif sentiment != None:
        lemmatized_tweet = df[df['emotion'] == sentiment]['lemmatized_tweet']

        # Flatten the list
        flattened_lemmatized_tweet = [token for sublist in lemmatized_tweet for token in sublist]

    # Create BigramCollocationFinder
    finder = BigramCollocationFinder.from_words(flattened_lemmatized_tweet)

    # Score bigrams by raw frequency
    scored = finder.score_ngrams(bigram_measures.raw_freq)

    # Display the 20 most common bigrams
#     for bigram, score in scored[:20]:
#         print(bigram, score)

    # Order the bigrams
    scores = sorted(scored[:items], key=lambda x: x[1])

    # labels and width
    labels, score = [b[0] + " " + b[1] for b, _ in scores], [score for _, score in scores]
```

```
    labels, score = [b[0] + ' ' + b[1] for b, _ in scores], [score for _, score in scores]

    # Plot
    plt.figure(figsize=(10,6))
    plt.title(title, fontsize=14)
    plt.ylabel("Bigrams")
    plt.xlabel("Scores")
    plt.barh(y=labels, width=score);
```

In [ ]:
```
bigram_plots()
```

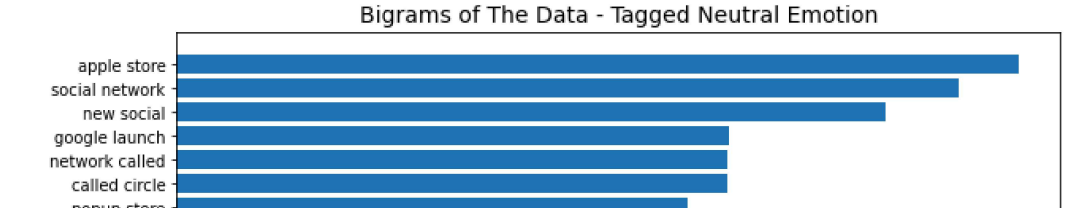


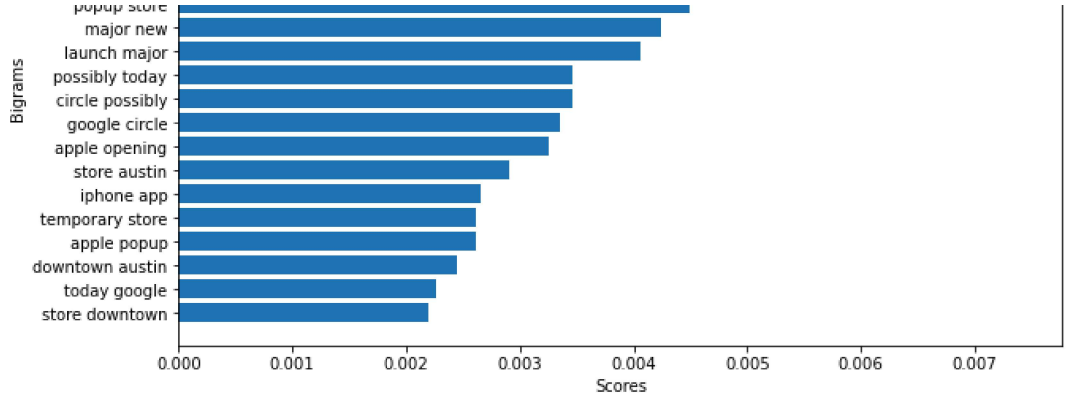
Observation

"apple store", "social network" and "new social" had the highest scores with respect to all the data available.

Neutral Data

In [ ]:
```
bigram_plots(sentiment='Neutral emotion', title='Bigrams of The Data - Tagged Neutral Emotion')
```



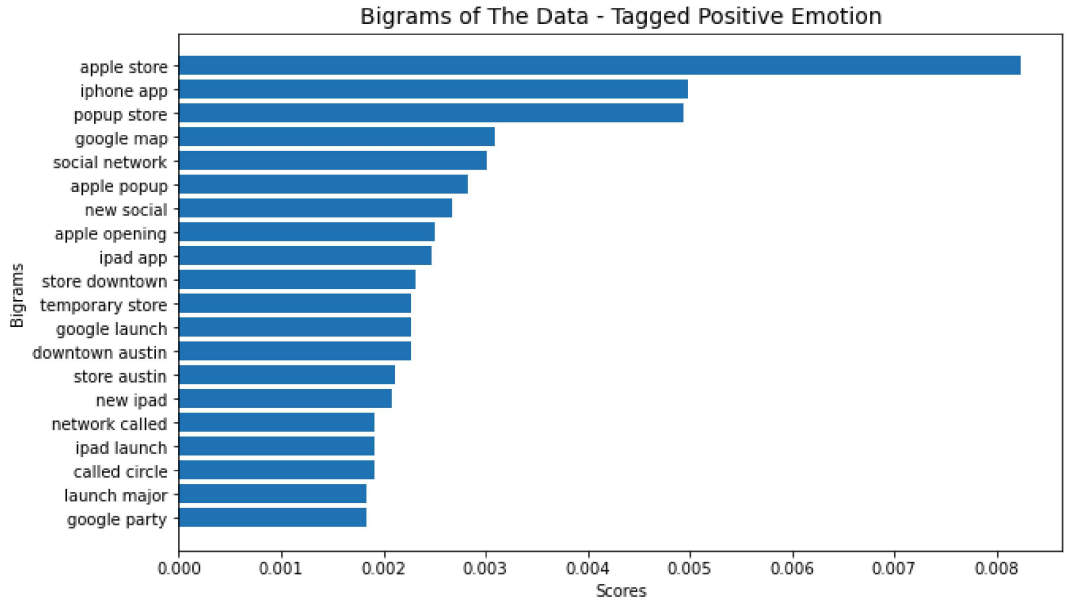


Observation

"apple store", "social network" and "new social" had the highest scores with respect to all the data categorised as neutral.

Bigrams - Positive Emotion

In [ ]:
```
bigram_plots(sentiment='Positive emotion', title='Bigrams of The Data - Tagged Positive Emotion')
```

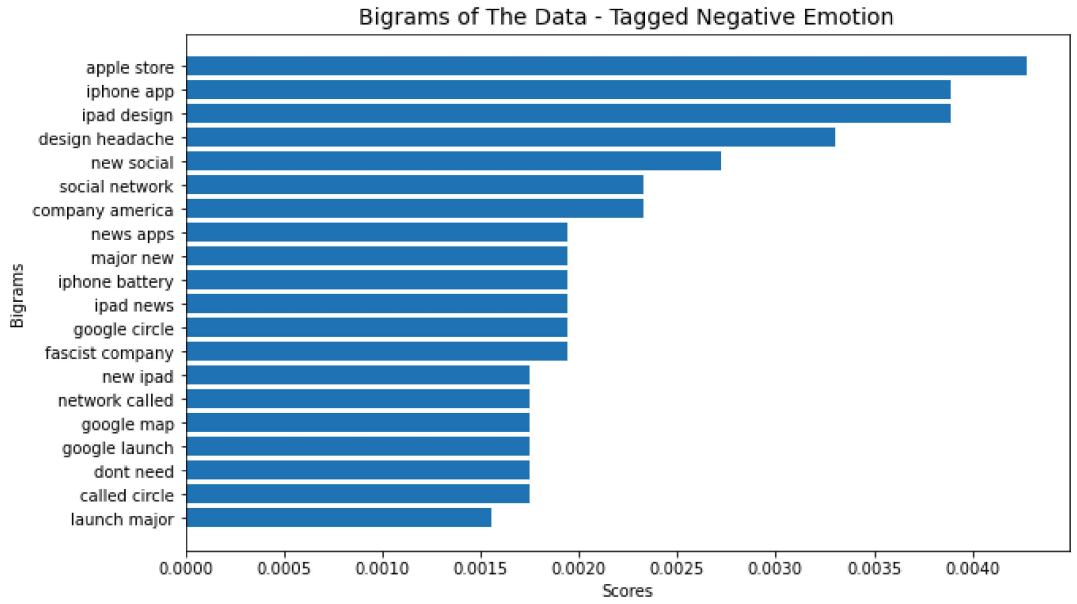


Observation

"apple store", "iphone app" and "popup store" had the highest scores with respect to all the data categorised as positive.

Bigrams - Negative Emotion

In [ ]:
```
bigram_plots(sentiment='Negative emotion', title='Bigrams of The Data - Tagged Negative Emotion')
```



Observation

"apple store", "iphone app" and "ipad design" had the highest scores with respect to all the data categorised as negative.

Modelling

Preprocessing Prepare data for modeling by:

Label Encoding: Converted emotion labels into numerical values. Vectorization: Used TF-IDF and CountVectorizer to transform text data into numerical vectors. SMOTE: Applied SMOTE to handle class imbalance. Train test split: To split the data Benefits:

These steps facilitate machine learning algorithms to process the emotion variable, convert text into a numerical format for better analysis, ensure the model is not biased towards the majority class, and provide clear metrics to evaluate performance on unseen data.

Label Encoding

In [ ]:
```
# Classify the data
y = df['emotion']
```

```
# Label Enconde the target
label_encoder = LabelEncoder()
target = label_encoder.fit_transform(y)
target[:5]
```

```
array([0, 2, 2, 0, 2])
```

In [ ]:
```
# Confirm labels
label_encoder.classes_
```

```
array(['Negative emotion', 'Neutral emotion', 'Positive emotion'],
      dtype=object)
```

Vectorization

CountVectorizer

In [ ]:
```
# Vectorization - CV
cv = CountVectorizer()

X_vec = cv.fit_transform(df['clean_tweet'])
X_vec = pd.DataFrame.sparse.from_spmatrix(X_vec)
X_vec.columns = sorted(cv.vocabulary_)
X_vec.set_index(y.index, inplace=True)
X_vec = X_vec.iloc[:, 160:]
```

In [ ]:
```
import pandas as pd
from imblearn.over_sampling import SMOTE

# Ensure X_vec is a NumPy array
if isinstance(X_vec, pd.DataFrame):
    X_vec = X_vec.values

# Check shapes of X_vec and target
print("Shape of X_vec:", X_vec.shape)  # Should be (n_samples, n_features)
print("Shape of target:", target.shape)  # Should be (n_samples,)

# Proceed with SMOTE if everything looks correct
smote = SMOTE()
X_cv_smote, y_cv_smote = smote.fit_resample(X_vec, target)

print("Shape of X_cv_smote:", X_cv_smote.shape)
print("Shape of y_cv_smote:", y_cv_smote.shape)
```

```
Shape of X_vec: (8439, 8388)
Shape of target: (8439,)
Shape of X_cv_smote: (15213, 8388)
Shape of y_cv_smote: (15213,)
```

Train Test split - CV

In [ ]:
```
# Train Test and split - CV
X_train_cv, X_test_cv, y_train_cv, y_test_cv = train_test_split(X_cv_smote, y_cv_smote, test_size=0.3, random_state=42)
```

In [ ]:
```
# Vectorization - TFIDF
tf = TfidfVectorizer()

X_tf = tf.fit_transform(df['clean_tweet'])
X_tf = pd.DataFrame.sparse.from_spmatrix(X_tf)
X_tf.columns = sorted(tf.vocabulary_)
X_tf.set_index(y.index, inplace=True)
X_tf = X_tf.iloc[:, 160:]
```

In [ ]:
```
# SMOTE - TFIDF

#X_tf_smote, y_tf_smote = smote.fit_resample(X_tf, target)

print(X_tf.shape)
```

```
(8439, 8388)
```

In [ ]:
```
from imblearn.over_sampling import SMOTE

# Check type
print(type(X_tf))  # Ensure this is a DataFrame

# Convert to NumPy array
X_tf_dense = X_tf.values  # or use X_tf.to_numpy()

# Check the shapes
print(X_tf_dense.shape)
print(len(target))

# Apply SMOTE
smote = SMOTE(random_state=42)
X_tf_smote, y_tf_smote = smote.fit_resample(X_tf_dense, target)
```

```
<class 'pandas.core.frame.DataFrame'>
(8439, 8388)
8439
```

In [ ]:
```
# Train Test and Split - TFIDF
X_train_tf, X_test_tf, y_train_tf, y_test_tf = train_test_split(X_tf_smote, y_tf_smote, test_size=0.3, random_state=42)
```

Models

Models

The machine learning algorithims used in this section are:

RandomForest

Naive Bayes(MultinomialNB)

LogisticRegression

DecisionTrees

We will use the split data to predict which model will achieve the highest accuracy and use it for deployment.

In [ ]:

```
def modelling(model, cv=False, tf=False):
    if cv == True:
        # Fit the instantiated model
        model.fit(X_train_cv, y_train_cv)
        # Predict
        y_hat = model.predict(X_test_cv)
        # Results
        print("Count Vectorisation Results\n")
        print("Main Metrics")
        print("-"*12)
        print(f"Accuracy Score {round(accuracy_score(y_test_cv, y_hat), 3)}")
        # Use 'macro' averaging for multiclass classification
        print(f"Recall Score {round(recall_score(y_test_cv, y_hat, average='macro'), 3)}")
        # Classification Report
        print("\nClassification Report")
        print(classification_report(y_test_cv, y_hat))


    elif tf == True:
        # Fit the instantiated model
        model.fit(X_train_tf, y_train_tf)
        # Predict
        y_hat = model.predict(X_test_tf)
        # Results
        print("---------")
        print("TFIDF Vectorisation Results\n")
        print("Main Metrics")
        print("-"*12)
        print(f"Accuracy Score {round(accuracy_score(y_test_tf, y_hat), 3)}")
        # Use 'macro' averaging for multiclass classification
        print(f"Recall Score {round(recall_score(y_test_tf, y_hat, average='macro'), 3)}")
        # Classification Report
        print("\nClassification Report")
        print(classification_report(y_test_tf, y_hat))

def hyper_tuning(model, params, model_name="Random Forest"):
    """This function optimises the base model with the parameters
    passed as params"""
```

```
    # Grid Search Base Model
    grid_search_model = GridSearchCV(model, params, cv=5, scoring='accuracy')
    # Count Vectorisation
    # Perform grid search with 5-fold cross-validation for Count Vectorization
    grid_search_rf_cv = grid_search_model.fit(X_train_cv, y_train_cv)

    # Get the best model from grid search for Count Vectorization
    best_rf_model_cv = grid_search_rf_cv.best_estimator_

    # Predict on the test set using the best model for Count Vectorization
    y_pred_cv = best_rf_model_cv.predict(X_test_cv)

    # Calculate and print the accuracy for Count Vectorization
    accuracy_cv = accuracy_score(y_test_cv, y_pred_cv)

    # Calculate and print the recall for Count Vectorization
    recall_cv = recall_score(y_test_cv, y_pred_cv, average ='macro')

    # Results
    print("Count Vectorisation Results\n")
    print(f"Best {model_name.title()} Model (Count Vectorization):\n", best_rf_model_cv)
    print(f"\nTest Accuracy (Count Vectorization): {accuracy_cv:.3f}")
    print(f"\nTest Recall (Count Vectorization): {recall_cv:.3f}")
    print("---------")

    #TFIDF Vectorisation
    grid_search_rf_tf = grid_search_model.fit(X_train_tf, y_train_tf)

    # Get the best model from grid search for TF-IDF Vectorization
    best_rf_model_tf = grid_search_rf_tf.best_estimator_

    # Predict on the test set using the best model for TF-IDF Vectorization
    y_pred_tf = best_rf_model_tf.predict(X_test_tf)

    # Calculate and print the accuracy for TF-IDF Vectorization
    accuracy_tf = accuracy_score(y_test_tf, y_pred_tf)

    # Calculate and print the reacll for TF-IDF Vectorization
    recall_tf= recall_score(y_test_tf, y_pred_tf, average ='macro')

    # Results
    print("\n\nTFIDF Vectorisation Results\n")
    print(f"Best {model_name.title()} Model (TFIDF Vectorization):\n", best_rf_model_tf)
    print(f"\nTest Accuracy (TFIDF Vectorization): {accuracy_tf:.3f}")
    print(f"\nTest Recall (TFIDF Vectorization): {recall_tf:.3f}")

    # models = [best_rf_model_cv, best_rf_model_tf]

    return best_rf_model_cv, best_rf_model_tf
```

Multinomial Bayes

In [ ]:

```
# Import necessary libraries
from sklearn.naive_bayes import MultinomialNB

# Instantiate the multinomialnb model
mnb = MultinomialNB()

# Optionally, fit the model with your data
# mnb.fit(X_train, y_train)
```

In [ ]:

```
# Classification report of the multinomial using the Count Vectorization
modelling(model=mnb, cv=True)
# Classification report of the multinomial using the TFIDF Vectorization
modelling(model=mnb, tf=True)
```

```
Count Vectorisation Results

Main Metrics
------------
Accuracy Score 0.644
Recall Score 0.643

Classification Report
              precision    recall  f1-score   support

           0       0.73      0.74      0.73      1541
           1       0.70      0.57      0.63      1523
           2       0.53      0.62      0.57      1500

    accuracy                           0.64      4564
   macro avg       0.65      0.64      0.64      4564
weighted avg       0.65      0.64      0.65      4564

---------
TFIDF Vectorisation Results

Main Metrics
------------
Accuracy Score 0.772
Recall Score 0.771

Classification Report
              precision    recall  f1-score   support

           0       0.86      0.97      0.91      1541
           1       0.75      0.59      0.66      1523
           2       0.70      0.76      0.73      1500

    accuracy                           0.77      4564
   macro avg       0.77      0.77      0.76      4564
weighted avg       0.77      0.77      0.77      4564
```

Hyperparameter Tuning the MNB Model

In [ ]:
```
# params
mnb_param_grid = {
    'alpha': [0.01, 0.1]
}

# GridSearchCV for tuning
tuned_mnb_cv_model, tuned_mnb_tf_model = hyper_tuning(model=mnb, params=mnb_param_grid, model_name="MNB")
```

```
Count Vectorisation Results

Best Mnb Model (Count Vectorization):
 MultinomialNB(alpha=0.01)

Test Accuracy (Count Vectorization): 0.672

Test Recall (Count Vectorization): 0.671
---------


TFIDF Vectorisation Results

Best Mnb Model (TFIDF Vectorization):
 MultinomialNB(alpha=0.01)

Test Accuracy (TFIDF Vectorization): 0.798

Test Recall (TFIDF Vectorization): 0.797
```

Observation

The accuracy score is at 80% which is an improvement from 77%. The models improvement is due to tuning

Random Forest

Let's now work with the random forest model and look at its performance.

Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model.

In [ ]:
```
# Instantiate a random forest model
# Set `n_estimators = 1000` , `max_features = 5` and `max_depth = 5`
rf = RandomForestClassifier(n_estimators=1000, max_features=5, max_depth=5)
```

In [ ]:
```
modelling(model=rf, cv=True)
modelling(model=rf, tf=True)
```

```
Count Vectorisation Results

Main Metrics
```

```
------------
Accuracy Score 0.493
Recall Score 0.496

Classification Report
              precision    recall  f1-score   support

           0       0.91      0.18      0.30      1541
           1       0.72      0.44      0.55      1523
           2       0.39      0.87      0.54      1500

    accuracy                           0.49      4564
   macro avg       0.67      0.50      0.46      4564
weighted avg       0.68      0.49      0.46      4564

---------
TFIDF Vectorisation Results

Main Metrics
------------
Accuracy Score 0.703
Recall Score 0.704

Classification Report
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      1541
           1       0.79      0.33      0.47      1523
           2       0.54      0.92      0.68      1500

    accuracy                           0.70      4564
   macro avg       0.77      0.70      0.69      4564
weighted avg       0.77      0.70      0.69      4564
```

Hyperparameter tuning the Random Forest Classifier

In [ ]:

```python
# Define the Random Forest classifier
rf = RandomForestClassifier(random_state= 42)

# Define the parameter grid with the necessary hyperparameters
rf_param_grid = {
    'n_estimators': [100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30]   # Maximum depth of the tree
}

tuned_rf_cv_model, tuned_rf_tf_model = hyper_tuning(model=rf, params=rf_param_grid, model_name="Random Forest")
```

```
Count Vectorisation Results

Best Random Forest Model (Count Vectorization):
 RandomForestClassifier(n_estimators=200, random_state=42)

Test Accuracy (Count Vectorization): 0.709
```

```
Test Recall (Count Vectorization): 0.708
---------


TFIDF Vectorisation Results

Best Random Forest Model (TFIDF Vectorization):
 RandomForestClassifier(n_estimators=200, random_state=42)

Test Accuracy (TFIDF Vectorization): 0.839

Test Recall (TFIDF Vectorization): 0.838
```

Observation

The significant improvement in test accuracy from 0.50 to 0.70 in the model using Count Vectorization.

Note the improvement from 0.71 to 0.837 for the model using TF-IDF Vectorization

We can note an indication that TF-IDF provides a superior feature representation for the Random Forest model.

Logistic Regression

In [ ]:
```
# Instantiate the Logistic Regression Model
lr = LogisticRegression(max_iter=200)
```

In [ ]:
```
modelling(model=lr, cv=True)
modelling(model=lr, tf=True)
```

```
Count Vectorisation Results

Main Metrics
------------
Accuracy Score 0.707
Recall Score 0.706

Classification Report
              precision    recall  f1-score   support

           0       0.74      0.92      0.82      1541
           1       0.72      0.64      0.68      1523
           2       0.65      0.55      0.60      1500

    accuracy                           0.71      4564
   macro avg       0.70      0.71      0.70      4564
weighted avg       0.70      0.71      0.70      4564

---------
TFIDF Vectorisation Results
```

```
Main Metrics
------------
Accuracy Score 0.814
Recall Score 0.813

Classification Report
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      1541
           1       0.74      0.75      0.74      1523
           2       0.78      0.71      0.75      1500

    accuracy                           0.81      4564
   macro avg       0.81      0.81      0.81      4564
weighted avg       0.81      0.81      0.81      4564
```

In [ ]:

```python
# A secondary model manually tuned
lr_tune = LogisticRegression(max_iter=3000, C=100, solver='liblinear')
modelling(lr_tune, cv=True)
modelling(lr_tune, tf=True)
```

```
Count Vectorisation Results

Main Metrics
------------
Accuracy Score 0.712
Recall Score 0.711

Classification Report
              precision    recall  f1-score   support

           0       0.76      0.92      0.83      1541
           1       0.71      0.65      0.68      1523
           2       0.65      0.56      0.60      1500

    accuracy                           0.71      4564
   macro avg       0.71      0.71      0.70      4564
weighted avg       0.71      0.71      0.70      4564

---------
TFIDF Vectorisation Results

Main Metrics
------------
Accuracy Score 0.83
Recall Score 0.829

Classification Report
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      1541
           1       0.78      0.72      0.75      1523
           2       0.77      0.78      0.77      1500

    accuracy                           0.83      4564
```

```
   macro avg       0.83      0.83      0.83      4564
weighted avg       0.83      0.83      0.83      4564
```

Hyperparameter Tuning Logistic Regression Model

In [ ]:

```
# Parameter Tuning
c_space = np.linspace(30, 32, 3)
max_iters = [100, 150, 200]
solvers = ["lbfgs", "liblinear"]
lr_param_grid = { 'C': c_space, 'max_iter':max_iters }
tuned_lr_cv_model, tuned_lr_tf_model = hyper_tuning(model=lr, params=lr_param_grid, model_name="Logistic Regression")
```

```
Count Vectorisation Results

Best Logistic Regression Model (Count Vectorization):
 LogisticRegression(C=30.0)

Test Accuracy (Count Vectorization): 0.711

Test Recall (Count Vectorization): 0.710
---------


TFIDF Vectorisation Results

Best Logistic Regression Model (TFIDF Vectorization):
 LogisticRegression(C=30.0, max_iter=200)

Test Accuracy (TFIDF Vectorization): 0.829

Test Recall (TFIDF Vectorization): 0.828
```

Observation

We found the best model to be Random Forest Model and Logistic Regression both with the highest accuracy scores of 83%

Key Findings

We explored the effectiveness of various machine learning models to predict the sentiment of tweets about Apple and Google products. Our data was preprocessed by label encoding the emotion labels, applying SMOTE to address class imbalance, vectorizing the text data using both CountVectorizer and TF-IDF. We then evaluated the performance of the models including RandomForest, Naive Bayes (MultinomialNB), Logistic Regression and conducted hyperparameter tuning to optimize their performance.

The best model was found to be Random Forest Classifier and Logistic Regression

Vectorization:

TF-IDF Vectorization consistently outperformed CountVectorizer in all models. It has demonstrated its superior capability in feature

representation for sentiment analysis.

Model Performance:

Tuned Random Forest and Tuned Logistic Regression models achieved the highest accuracy and recall scores with TF-IDF vectorization, both scoring approximately 83.7% in accuracy and 83.6% in recall. MultinomialNB also performed well, especially after tuning, reaching an accuracy and recall score of 80%.

Hyperparameter Tuning:

Hyperparameter tuning significantly improved model performance, as seen in the Random Forest and Logistic Regression models where accuracy and recall improved by more than 10% in some cases.

Class Imbalance:

Applying SMOTE was effective in handling class imbalance, ensuring that the models did not bias towards the majority class and provided balanced performance across all emotion categories.

Recommendations

Based on the findings, here are some recommendations for future work and practical application:

Monitoring Negative Sentiments:

Implement real-time monitoring and alert systems to flag negative sentiments as they arise. This allows for prompt intervention and resolution of customer issues.

Scalability:

Assess the scalability of the models for handling large-scale data in a production environment. Optimize the models for performance and efficiency to ensure they can process a high volume of tweets quickly and accurately.

Real-Time Processing:

Explore real-time processing capabilities to provide up-to-date sentiment analysis, which is crucial for timely decision-making and responding to emerging trends.

Continuous Model Monitoring:

Implement continuous monitoring of the deployed models to detect any performance degradation over time. Retrain the models if necessary with new data to maintain accuracy and relevance.

Integration with Social Media Platforms:

Integrate the sentiment analysis models with social media platforms' APIs for seamless data collection and analysis, enabling continuous monitoring and real-time insights.

Conclusion

Summary of Findings:

The project successfully evaluated various machine learning models for classifying tweets into emotion categories. TF-IDF Vectorization demonstrated superior performance compared to CountVectorizer, and the Tuned Logistic Regression and Tuned Random Forest models achieved high accuracy and recall. Hyperparameter tuning significantly enhanced model performance, and the SMOTE technique effectively addressed class imbalance.

Project Impact:

The project's results offer valuable insights into effective sentiment analysis for social media data. The models developed provide robust tools for classifying tweets and understanding public sentiment, contributing to advancements in sentiment analysis and machine learning applications.

Final Thoughts:

The project highlights the importance of effective data preprocessing, model evaluation, and hyperparameter tuning in achieving high-performance sentiment analysis. The findings emphasize the potential of machine learning models in practical applications and pave the way for future research and enhancements in the field.

In [ ]:
```
print ("THE END")
```

```
THE END
```