Sentiment Analysis of Tweets on Apple and Google Products

Business Understanding

Business overview

This project focuses on leveraging sentiment analysis, a key natural language processing (NLP) technique, to classify tweets about Apple and Google products and services into positive, negative or neutral categories. Sentiment analysis helps uncover public opinions, emotions and reactions from textual data, offering valuable insights for businesses to improve customer experiences, predict behavior and enhance decision making.

Apple, a global leader in consumer electronics, and Google, a provider of diverse hardware, software and cloud services, both benefit from understanding customer sentiment to refine their products and maintain market dominance. By analyzing tweets, this project aims to identify trends and patterns in public discussions about these brands. Using machine learning, the goal is to develop a model that accurately categorizes tweets based on sentiment, providing actionable insights for stakeholders to address customer concerns and improve brand performance.

Problem Statement

In the digital era, customer feedback is crucial for grinding business strategies. However the vast amount of unstructured textual data from reviews, surveys and social media makes manual analysis inefficient and unreliable.

This project aims to develop a sentiment analysis model to automatically classify customer feedback as positive or negative or neutral. By doing so, businesses can:

- Pinpoint areas needing improvement
- Customize marketing efforts based on customer sentiment
- Track and manage brand reputation effectively.

Project Objectives

1. Primary Objective:

Develop an unsupervised machine learning model to classify tweets as positive or negative toward a brand or product.

2. Secondary Objectives:

- Identify whether a tweet contains an emotion directed at a specific brand or product.
- Preprocess and clean the tweet text to remove noise (e.g., hashtags, mentions, and URLs).
- Identify the most positive words directed at apple and google company's products/services
- Identify the most negative words directed at apple and google company's products/ services
- Provide actionable insights to help businesses improve customer satisfaction and marketing strategies.

Approach Methodology

The methodology involves a systematic approach to preprocessing, feature extraction, model selection and evaluation. By iteratively testing and refining different techniques, the goal is to identify the most effective combination of features and models for accurately classifying tweet sentiment. This will provide actionable insights into customer opinions and help improve decision making for Apple and Google.

To address the key questions outlined in this project, the following approach methodology was employed.

- Text normalization: to convert text, remove punctuation and handle contractions.
- Tokenization: to split text into individual words or tokens
- Stop word removal: to eliminate common words that do not contribute to sentiment.
- Lemmatization: reduce words to their base or root forms to standardize variations.
- noise removal: filter out irrelevant data like urls, hashtags and mentions
- Baseline models, like logistic regression or Naive Bayes.
- advanced models, experiment with ensemble methods like random forest and gradient boosting for improved performance
- deep learning models,
- model evaluation
- Performance benchmarking: compare the performance of different models and feature combinations to identify the best approach, fine tune hyper-parameters to optimize model performance and report the achieved levels of accuracy, precision and recall for the final sentiment classification.

Metrics of Success.

To assess the effectiveness of the sentiment analysis model, the following evaluation metrics will be used:

1. Accuracy: Measures the percentage of correctly classified sentiment instances out of the total.

Target: 85% or higher.

2. Precision: Indicates the percentage of correctly predicted positive sentiments, reflecting how often the model is correct when predicting a specific sentiment.

Target: 80% or higher for each sentiment class (positive, negative or neutral).

3. Recall: evaluates the model's ability to correctly identify all relevant instances of a specific sentiment, balancing true positives and false negatives.

Target: 75% or higher for each class.

4. F1-Score: provides a balanced measure by combining precision and recall into a single metric.

Target: 80% or higher overall.

5. Confusion matrix: visualizes the model's performance by showing true positives, true negatives, false positives and false negatives.

6. Business impact:

- Enhances customer satisfaction by identifying and addressing negative sentiments.
- Informs better marketing strategies by analyzing trends in positive feedback

These metrics will ensure the model's reliability and its ability to deliver actionable insights for improving customer experiences and business strategies.

Data Understanding

Obtaining the data

The dataset used to build the NLP model for analyzing Twitter sentiment about Apple and Google products was sourced from Crowd Flower via https://data.world/crowdflower/brands-and-product-emotions and was read using the Panda's library in CSV format. The key steps in data inspection included checking the dataset's shape (9093 rows, three columns), extracting metadata with .info(), describing the data using .describe(), identifying duplicates with drop duplicates(), and detecting missing values using .isna(). Additionally, unique categories and their distribution in categorical columns such as "emotions_in_tweet_is_directed_at" and "is there an emotion directed at a brand or product" were analyzed. The dataset consists of three main columns: tweet_text, which contains the original Twitter messages; "Emotions_in_tweet_is_directed_at," which specifies the product (e.g., Apple or Google) being discussed; and "Is_there_an_emotion_directed_at_a_brand_or_product," which captures the emotion (e.g., positive or negative) associated with the tweet.

Data Preparation

1. Handling missing values:

- Rows with missing tweet_text were dropped using dropna().
- The Emotions_in_tweet_is_directed_at

Data cleaning

We performed series of steps to clean and preprocess the dataset to ensure accurate analysis of Twitter sentiment. First, the missing values in the 'tweet_text' and 'emotion_in_tweet_is_directed_at' columns were handled. In the 'tweet_text' column, any rows with missing values were removed using the dropna() method. For the 'emotion_in_tweet_is_directed_at' column, which had 63% of its values missing, it was decided to drop the column due to the high percentage of missing data. Instead, a new column was created to capture the required information, which will be discussed later. Next, the dataset was checked for duplicates, and 22 duplicate rows were identified and removed to improve data accuracy.

The text was then standardized by addressing case inconsistencies, eliminating special characters such as question marks, commas, and periods, and converting all text to lowercase for uniformity. Tokenization was performed to break the text into individual words or tokens, making it easier to process. A new processed text column was created by joining the cleaned tokens. Further cleaning included removing stop words, common, low-information words like "the," "is," and "in." Specific elements of the text, such as mentions (e.g., @username) and hashtags (e.g., #product), were extracted and treated separately to avoid introducing noise into the analysis. Two additional columns were created to capture mentions and hashtags separately, allowing these components to be analyzed independently. Finally, additional words and numbers that were not relevant to the analysis were removed to further clean and transform the data. These preprocessing steps ensured that the dataset was ready for modeling and analysis, with text cleaned and structured for effective sentiment analysis.

Handling duplicates is an essential step to ensure data integrity and avoid skewed results in a data science project. In this case, we identified 22 duplicate rows in the dataset, which could otherwise over represent certain observations and affect model accuracy. To resolve this, we used the drop_duplicates() method with the keep='first' parameter, which keeps the first instance of each duplicate while removing the rest. This ensures that unique observations are retained, preventing bias and ensuring that our analysis is based on clean, distinct data. By doing so, we improve the quality of the dataset and ensure more reliable insights from subsequent analysis or model training.

To handle missing values, we start by removing rows with missing values in the 'tweet_text' column using pandas' dropna() method, as missing tweet texts cannot be used in the analysis. For the 'emotion_in_tweet_is_directed_at' column, we found that over 50% of the values are missing. Instead of dropping the column, we evaluate the trade-off and decide to impute the missing values. We loop through the 'tweet_text' column, cleaning the text by removing special characters and converting it to lowercase. Then, we apply keyword extraction or sentiment analysis to infer the missing emotional values from the

text. These inferred values are stored in a new column, preserving the dataset's integrity. This approach helps to retain useful data while minimizing the impact of missing values.

```
# Drop the missing value in tweet_text column. Only one was identified above
df = df.dropna(subset = ['tweet_text'])
```

```
# Check if missing value is removed in tweet , also check remaining missing values in other columns
Pf.check_for_missing_values(df)
```

```
tweet_text                                            0
emotion_in_tweet_is_directed_at                    5788
is_there_an_emotion_directed_at_a_brand_or_product    0
dtype: int64
```

```
# check for the percentage distribution of missing values
missing_percentage = df['emotion_in_tweet_is_directed_at'].isna().value_counts(normalize = True)* 100
# Format the percentages to include the '%' symbol
missing_percentage = missing_percentage.apply(lambda x: f"{x:.2f}%")
missing_percentage
```

```
True     63.81%
False    36.19%
Name: emotion_in_tweet_is_directed_at, dtype: object
```

Fig.1: Check for missing value.

The code above executes to show the percentage of missing and present rows. We have around 64% missing values and only 36% present. So we need to evaluate a way to work towards reducing the missing values.

Dealing with the text column

Dealing with the text column is a crucial part of data preprocessing for NLP tasks. The first step in cleaning the text involves removing capitalization and special characters like "?, ;, .," and converting the entire text to lowercase for consistency. We then tokenize the text, breaking it into individual words (or tokens) to facilitate analysis. A new column is created by joining the cleaned and tokenized words into a single string. Next, we remove stop words common, non-informative words like "the," "is," and "in" that do not contribute much to the sentiment analysis. To further clean the text, we extract mentions, words starting with '@' and hashtags (words starting with "'#')," as they refer to users and tagged topics, respectively. These elements may introduce noise, so we extract them into separate columns—one for users and another for hashtags- to ensure they are analyzed separately and do not skew the sentiment analysis.

```
import re
# Regular expression to extract Twitter usernames
pattern = r"@\w+"
pattern_2 = r'#\w+'

# Extract usernames from the 'Tweets' column
df['Usernames'] = df['tweet_text'].apply(lambda x: re.findall(pattern, x))
df['Tagged_Names'] = df['tweet_text'].apply(lambda x: re.findall(pattern_2, x))

df.head()
```

Fig.2: The Regex pattern.

We write a function that accesses the 'tweet_text' column, converting the entire text to lowercase for uniformity. The next step involves removing usernames (words beginning with @) and tagged names (words beginning with #), as these are typically references to users or topics that are not relevant to sentiment analysis and can introduce noise into the data. A new column, 'clean_tweet_text,' is then created to store the cleaned text without these elements. Additionally, we remove URLs (e.g., those starting with "http," "https," or "www") as they do not contribute to the sentiment analysis and can distort the text. This early cleaning helps streamline the data, ensuring that the text used for analysis focuses solely on the relevant content.

```
]:  # Transform the whole dataset (df[tweet_text]) to lowercase
    df["tweet_text"] = df["tweet_text"].str.lower()
    # Display full text: uncomment the code below to display the whole texts
    #df.style.set_properties(**{'text-align': 'left'})
```

Removing urls, https, www

```
]:  import re
    def remove_urls(text, replacement_text= ' url'):
        url_pattern = re.compile(r'https?:\/\/\S+|www\.\S+')
        urls = url_pattern.findall(text)

        for url in urls:
            text = text.replace(url, replacement_text)
        return text
    # Create new column with tokenized data
    df["clean_tweet_text"] = df["tweet_text"].apply(remove_urls)
```

```
]:  df['tweet_text'][678]

192]:  '#thekills interview now on @mention at #sxsw may be one of the most painfully v
```

Fig.3: Function to implement the Regex Pattern.

```
# Create a function that removes words starting with @ and #
def remove_words_with_at_and_hash(text):
    # Use a regular expression to remove words containing "@" and words starting with
    cleaned_text = re.sub(r'\S*@\S*|#\w+','', text)

    return cleaned_text
# Create new column with tokenized data
df["clean_tweet_text"] = df["clean_tweet_text"].apply(remove_words_with_at_and_hash)
# Display full text: uncomment the code below to display the whole texts
#df.style.set_properties(**{'text-align': 'left'})
df.head()
```

Fig.4: Function to remove: words with @ and #

First, we iterated through the tweet_text column to analyze the content of each tweet and identify probable mentions of products or services. For each tweet, extract relevant product-related terms and store them in a new column called category_words, which will captured the identified product keywords or phrases. Then, create a list of products, which will serve as a reference to match words or phrases from the tweet_text that

are related to these products. Finally, add a new column, tweet_Directed_at, where you will store the names of the products mentioned in the tweets, enabling better tracking of which products are being referenced in the social media data. We then created a list of categories of words containing the products and mapped them on another column called the tweet_Directed_at. We then obtained the value counts to check at the distributions of the words from the created categories.

```
categories = ['iPhone', 'iPad or iPhone App', 'iPad', 'Google','Android',
       'Apple', 'Android App', 'Other Google product or service',
       'Other Apple product or service']
len(categories)
```

```
9
```

```
def extract_category_words(tweet, categories):
    # Tokenize and check for category words
    extracted_words = []
    for category in categories:
        if category.lower() in tweet.lower():
            extracted_words.append(category)
    return " ".join(extracted_words)
df['Tweet_Directed_at'] = df['clean_tweet_text'].apply(lambda x: extract_category_words(x, categories))
# Check for the value counts in the new category column
df['Tweet_Directed_at'].value_counts()
```

Fig.5: Mapping the categories

From the value counts of the tweet Directed at column, we see that the number of missing products for the tweet text column is 1784. This is subject to further scrutiny. From the clean tweet text column, we can confirm that the links, words beginning with @ and # have been removed. We observe that the category words column contains 1,781 null entries. To address this, loop through the category words column to identify which entries are null. For those null entries, check if the corresponding indices have values in the emotion in tweet is directed at column; if they do, fill in the null entries in category words with the values from the emotion in tweet is directed at column. After updating the null entries, check the value counts of the category words column to confirm a reduction in the number of null entries. Once verified, proceed to drop any remaining rows with null values in the category words column. Finally, in the emotion in tweet is directed at column, examine the non-null values and check if the tweet Directed at column is null for these entries. If it is, map the non-null values from emotion in tweet is directed at into the absent entries in the tweet Directed at column.

```
import pandas as pd

def update_tweet_directed_at(df):
    # Condition where 'emotion_in_tweet_is_directed_at' has data,
    # but 'Tweet_Directed_at' is blank or NaN
    condition_a_data_b_blank = (df['emotion_in_tweet_is_directed_at'].notna() &
                                (df['emotion_in_tweet_is_directed_at'] != "") &
                                (df['Tweet_Directed_at'].isna() | (df['Tweet_Directed_at'] == "")))

    # Update 'Tweet_Directed_at' with 'emotion_in_tweet_is_directed_at' where the condition is true
    df.loc[condition_a_data_b_blank, 'Tweet_Directed_at'] = df.loc[condition_a_data_b_blank, 'emotion_in_tweet

    # Return the modified DataFrame
    return df
```

Fig.6: Updating the tweet_directed_at column.

Cleaning the 'Tweet_Directed_at' column.

Create a mapping for Google products, Appleducts, unknown, and IRR. Map this dictionary to create a column showing which company the tweet was directed to. We had now created a column where we were able to map the various products. The missing or null values were now denoted with the word "Unknown." Next, we proceeded to drop the "Unknown" and "IRR" entries, which represented irrelevant categories in the column, as shown in fig. 7 below. After that, we dropped the 'emotion_in_tweet_is_directed_at' column, as the 'Company_product' column was a better representation of this data. With the missing values handled, we moved on to dealing with the semantic analysis of the clean_tweet_text column.

```
    'iPad Android': 'IRR',
    'iPhone Apple': 'Apple Products',
    'Google Apple': 'IRR',
    'Android App': 'Google Products',
    'Google Android': 'Google Products',
    'Other Google product or service': 'Google Products',
    'Other Apple product or service': 'Apple Products',
    'iPad Google': 'IRR',
    'iPhone Android Android App': 'IRR',
    'iPhone iPad Android': 'IRR',
    'Android Apple': 'IRR',
    'iPhone iPad Apple': 'Apple Products',
    'iPhone Google': 'IRR',
    'iPhone iPad Google': 'IRR',
    'iPhone Google Android': 'IRR'
}


# Apply mapping to the dataframe
df['Company_Product'] = df['Tweet_Directed_at'].map(category_mapping)
df.head()
# Display the result
df['Company_Product'].value_counts(normalize= True)
```

```
Apple Products     0.541014
Google Products    0.278501
Unknown            0.157773
IRR                0.022712
Name: Company_Product, dtype: float64
```

Fig.7: Updating the tweet_directed_at colum

We then proceeded to confirm that the 'is there an emotion directed at a brand or product' column had no null entries. After ensuring this, we checked the value counts of the 'is there an emotion directed at a brand or product' column and displayed the percentages of each category. This step allowed us to assess the distribution of emotions directed at brands or products in the dataset.

```
# Get the value counts for the emotion-directed column
Count_emotion = df['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts()

# Calculate the percentage for each category
percentage_emotion = df['is_there_an_emotion_directed_at_a_brand_or_product'].value_count

# Combine both count and percentage into a DataFrame
result_emotion = pd.DataFrame({
    'Count': Count_emotion,
    'Percentage': percentage_emotion.apply(lambda x: f"{x:.2f}%")  # Format percentage to
})

result_emotion
```

0]:

	Count	Percentage
Neutral	3884	52.25%
Positive	2856	38.42%
Negative	555	7.47%
Unknown	138	1.86%

Fig.8: Data Frame showing the counts of categories:

```
# show the clean tweet text for the unknown entries.
df[df['is_there_an_emotion_directed_at_a_brand_or_product'] == 'Unknown']['tweet_text']
```

```
76                          ?üï@mention &quot;apple has opened a pop-up store in austin so the nerds in town for #sxsw can
get their new ipads. {link} #wow
189                                 just what america needs. rt @mention google to launch major new social network called cir
cles, possibly today {link} #sxsw
286                                                                 the queue at the apple store in austin is fou
r blocks long. crazy stuff! #sxsw
308                                       hope it's better than wave rt @mention buzz is: google's previewing a social net
working platform at #sxsw: {link}
344                                                                          syd #sxsw crew your iphone ext
ra juice pods have been procured.
                                                                          ...
7371               it's funny watching a room full of people hold their ipad in the air to take a photo. like a room full o
f tablets staring you down. #sxsw
7379                                                                 @mention yeah, we have @mention ,
google has nothing on us :) #sxsw
7384                                                              @mention yes, the google presentation was not ex
actly what i was expecting. #sxsw
```

Fig.9. Display of the tweet texts for the unknown entries.

In the clean_tweet_text and the tweet_text columns, for the unknown entries, it is not clear what emotion is really being captured in the tweet; some of the tweets might be sarcastic or actually genuine, but we cannot tell. However, from the value count seen above, the number of "Unknown" emotions is not significant (1.8% of the total data is in the "emotion" column). It is, therefore, more prudent to drop them. The categories of interest in this analysis are the positive, neutral, and negative entries. Filter the data frame for the neutral, positive, and negative entries.

Additional Data Cleaning & EDA with NLTK

For this section, we focused on analyzing the Positive and Negative emotions captured in the "is there an emotion directed at a brand or product" column, as our primary objective was to examine these two categories. To begin, we started by preprocessing the clean_tweet_text column by removing URLs, non-alphanumeric characters, and numbers or digits to ensure the text was clean and ready for analysis. We created a function to remove these elements, applying it across the entire dataframe. Specifically, we utilized the RegexpTokenizer from the nltk.tokenize module, which allowed us to tokenize individual words in the text using a custom regular expression pattern.

```
def tokenize_and_display(df):
    # Define the basic token pattern
    basic_token_pattern = r"(?u)\b\w\w+\b"

    # Initialize the tokenizer with the specified pattern
    tokenizer = RegexpTokenizer(basic_token_pattern)

    # Apply the tokenizer to a new column in the DataFrame
    df["text_tokenized"] = df["clean_tweet_text"].apply(tokenizer.tokenize)

    # Display the DataFrame with left-aligned text
    return df["text_tokenized"]

# Call the function
tokenize_and_display(df)
```

Fig.10. Tokenizer Function

```
## Get the list of English stop words
stop_words = stopwords.words('english')
stop_words += list()
# Function to remove stop words
def remove_stop_words(tokens_list):
    """
    Removes stop words from a list of tokens.
    Arguments:
    - tokens_list: List of tokenized words.

    Returns:
    - List of tokens without stop words.
    """
    return [word for word in tokens_list if word.lower() not in stop_words]

# Apply the function to the tokenized text column
df["text_tokenized"] = df["text_tokenized"].apply(remove_stop_words)
df.head()
```

Fig.11 Removing stop words

The RegexpTokenizer split the text into word tokens, excluding punctuation and non-word characters, such as URLs, and putting the remaining words into a list. However, while it successfully removed unwanted characters, it did not eliminate stop words or numbers, so we filtered the column for these patterns as well.

After the text was cleaned and tokenized, we checked the frequency distribution of words using NLTK's FreqDist class to count the occurrences of each word. By applying the most_common() method, we retrieved the 50 most frequent words, which helped us identify key terms and trends in the dataset, providing valuable insights into the primary themes of the tweets.

```
df['text_tokenized'][1973:2000]
```

```
2009              found new mapquest iphone app turn turn gps free time check get reacquainted mq
2010    aha found proof lactation room excuse quot mother room quot brought google last year link
2011                      makes good sense rt apple set open popup shop core sxsw action link via
2012                         uber brand smart gonna move ton ipad2 opening popup shop austin link
2013                               sounds awesome location based amp could incredibly useful link
2014                                          ipad store sold everything except  wifi white
2015                      heard apple opening store downtown austin tomorrow sell ipads attendees
2017                     hah found quot popup quot apple store ipad sale  gold gym  congress link
2018                                  saved productive get anything done ipad dongle get get
2019                                                    found back door apple store ftw
2020                  almost worst fail left iphone kitchen counter realized min driving airport
2021                                                need charge talking sun iphone link boom
2022          interested hear different google buzz rt get ready word quot circles quot trending
2023            read tv recaps online keep shows  unread posts google reader completely detached
2024                                                    interested see google show unveil
2025                                                annoyed schedule app bad ipad fingers fat
2026                                          link hear apple ûªs opening temporary store via
2027                                                          html google booth link
2028                                                hold breath google circles least link
2029                stay tuned next emergency ipad station facebook bullying online privacy link
2030                                      poked liked tweeted google love story alisa icrossing
2031                                ballroom talking cool projects obv love google art project
```

Fig.12 The text tokenized column

From the tweet text column, we observed that the word "link" commonly refers to URLs or HTTPs, and the abbreviation "rt" signifies a retweet. Since these terms are not necessary for our analysis, we decided to add them to our stop words list and remove them from the text. After removing "link" and "rt," we were left with a more refined set of words, which helped in better identifying the key terms. Upon examining the most frequent words, we noticed that terms like "ipad," "google," and "apple" were among the most popular, but we also saw the word "quot" appearing frequently. This raised a concern for further inspection, as it seemed out of place.

After investigating, we determined that the word "quot" was likely part of an HTML entity reference for quotation marks ("), which may have resulted from improper parsing or formatting during content processing. Given this, we decided to add "quot" to our stop words list and remove it. Additionally, we identified the word "sxsw," which referred to the South by Southwest conference, and removed it as it wasn't significant in predicting emotions like positive, negative, or neutral. We also found the word "amp," which appeared due to misinterpretation or encoding of the HTML entity & representing the ampersand symbol (&). Since this was unnecessary for our analysis, we added "amp" to the stop words list and removed it as well.

We proceeded to apply lemmatization, which is a natural language processing technique that reduces words to their base or dictionary form, known as a lemma. Unlike stemming, which simply removes prefixes or suffixes to obtain a root word, lemmatization considers the meaning and context of a word within a sentence. This allows the algorithm to understand the word's role—whether it's a verb, noun, or adjective—and reduce it accordingly. We chose lemmatization over stemming because it provides more accurate and

context-aware results. By ensuring that words are reduced to their root forms while retaining their intended meaning, lemmatization helps eliminate noise in the data. This, in turn, improves the accuracy and precision of tasks such as sentiment analysis, modeling, and text classification, as the words are represented in their most meaningful forms.

```
# Apply lemmatization to the 'text_tokenized' column
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to lemmatize tokens
def lemmatize_tokens(tokens):
    return [lemmatizer.lemmatize(token) for token in tokens]

# Apply lemmatization to the 'text_tokenized' column
df['text_tokenized_lemmatized'] = df['text_tokenized'].apply(lemmatize_tokens)
```

```
#displaying 15 most common tokens
from nltk import FreqDist
tokens_count = [token for tokens in df['text_tokenized_lemmatized'] for token in tol
freq = FreqDist(tokens_count)
# Increase the number in the code below to verify removal
freq.most_common(15)
```

Fig.13 Lemmatization.

EXPLORATORY DATA ANALYSIS

We began by exploring the emotional sentiment associated with Apple and Google by analyzing the most popular words tied to positive and negative emotions. This process involved splitting the dataset into positive, negative, and neutral emotions to understand how each company is perceived in terms of sentiment. After separating the data, we focused specifically on Apple and Google products, examining the distribution of positive, negative, and neutral emotions for both companies. This allowed us to identify the most common words linked to positive and negative emotions for Apple and Google separately. To visually represent these insights, we utilized word clouds, which highlighted the most frequent words associated with each emotion for both companies. Through this approach, we gained a deeper understanding of consumer sentiments toward Apple and Google, revealing how each company is emotionally tied to its products in the public discourse and how these associations influence their overall image.

```
# Looking at the top 10 most popular words, @user_accounts and hashtags
# 25 Most popular words

from nltk import FreqDist
tokens_count = [token for tokens in df_pos_Apple['text_tokenized_lemmatized'] for token in to
freq = FreqDist(tokens_count)
freq.most_common(10)
```

```
[('ipad', 918),
 ('apple', 742),
 ('store', 535),
 ('iphone', 423),
 ('app', 278),
 ('austin', 227),
 ('pop', 209),
 ('new', 206),
 ('one', 135),
 ('line', 134)]
```

Fig.14 Frequency Distribution.

From the frequency distribution (Freq Dist) of words above, we observed that the most frequently occurring terms were the names of company products such as "iPad," "Apple," "iPhone," and "App," which were often associated with positive feedback. However, these product names do not accurately reflect the emotional sentiment itself; they are more specific to the company and contribute noise to the analysis. To better capture the actual sentiment attributed to these products, we decided to eliminate these product names by adding them to the stop words list. By doing so, we removed the influence of these brand-specific terms, allowing the analysis to focus on words that more directly reflect the emotional tone—whether positive, negative, or neutral—that people associate with the companies. This refinement allowed us to gain a clearer understanding of the true emotional sentiment, helping us focus on words that more accurately capture the underlying feelings expressed by users, rather than just product mentions. We then created the word clouds for the frequency distributions.

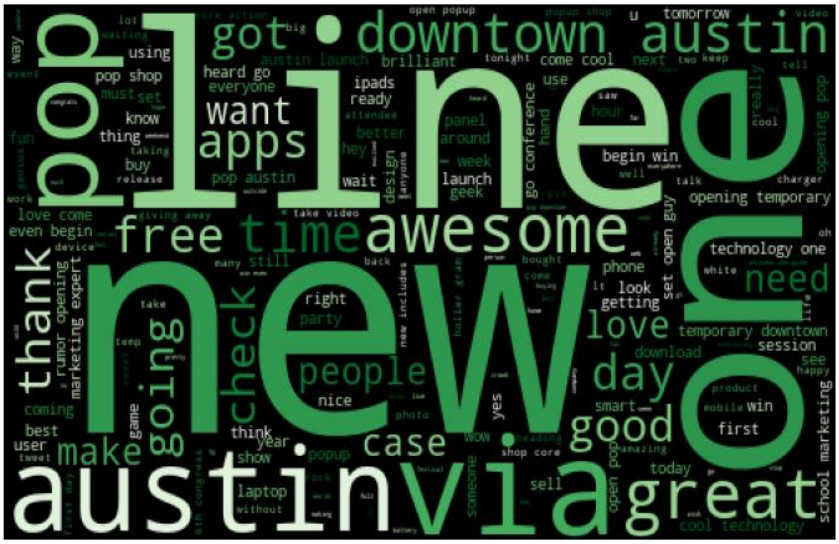

Fig.14 Positive word cloud: Apple.

The word cloud above highlights the most common words attributed to Apple products, such as "great," "love," "really," "good," "nice," "free," "awesome," "smart," "design," "check," "win," "fun," "launch," and "cool technology." These words reflect the overall sentiment towards Apple, portraying the brand as one that is highly regarded for its quality, innovation, and excitement. Words associated with positive experiences (like "great," "awesome," "good," "love") illustrate consumer satisfaction, while terms tied to product launches and excitement ("launch," "check," "win") emphasize the anticipation and enthusiasm surrounding Apple's products. Additionally, words such as "design" and "smart technology" convey the brand's focus on innovation and cutting-edge features. This collective sentiment paints a picture of Apple as a respected and beloved brand. Next, we perform the same analysis for Google, capturing the most positive words directed at the company. By examining the words most frequently associated with Google, we can similarly identify the key themes that shape public perception of the brand, such as its technological innovation, product quality, and consumer satisfaction.

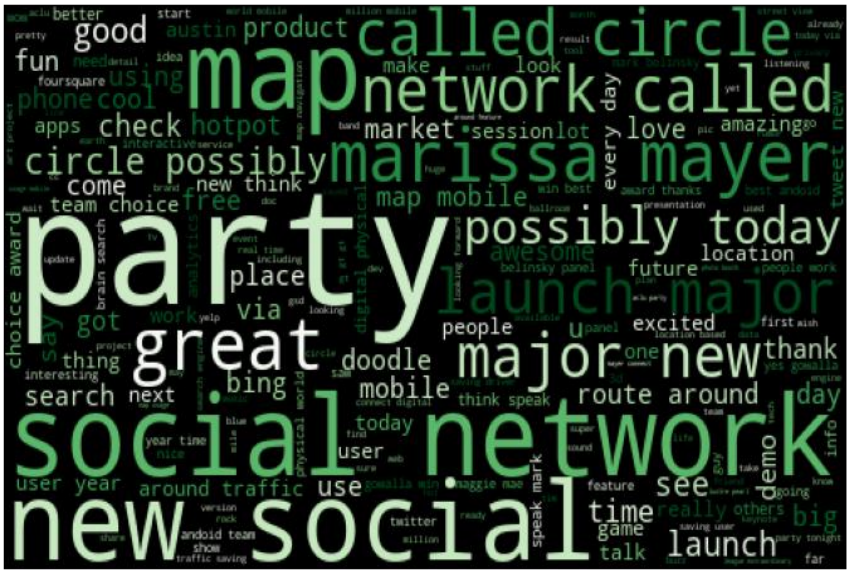

Fig.15. Positive word cloud: Google.

From the word cloud, the most common words directed at Google company products include "great," "awesome," "thank," "fun," "apps," "possibly," "social network," "new," "social," "excited," "check," "map," and "launch." While many of these words are linked to positive emotions, it's important to note that some may not directly represent positive sentiment. For example, terms like "social network" and "new social" suggest that users associate Google with social platforms, but they don't necessarily carry an inherently positive emotional tone. On the other hand, enthusiastic words like "awesome" and "great" clearly reflect strong user satisfaction. However, the inclusion of words like "possibly" and "social network" introduces an element of uncertainty or speculation, indicating that some users may be uncertain or neutral about certain aspects of Google's offerings. This mix of words reflects both the excitement and the ambiguity surrounding Google's products in the public.

To analyze the negative emotions in the dataset, we first filter the data frame to isolate the entries corresponding to negative emotions. Next, we separate the data for both Apple and Google, focusing specifically on the negative emotional responses for each company. To refine the word clouds and better capture the emotional sentiment, we enhance the stop words list by adding company-related terms such as product names (e.g., "iPhone," "iPad," "Apple," "Google") and locations (e.g., "California," "Silicon Valley"). By doing so, we can filter out these company-specific words and ensure that the word cloud

highlights more relevant emotional words tied directly to users' negative sentiments. This approach allows us to focus more precisely on the words that reflect genuine emotional reactions, making the word clouds more representative of how users feel about Apple and Google in a negative context.

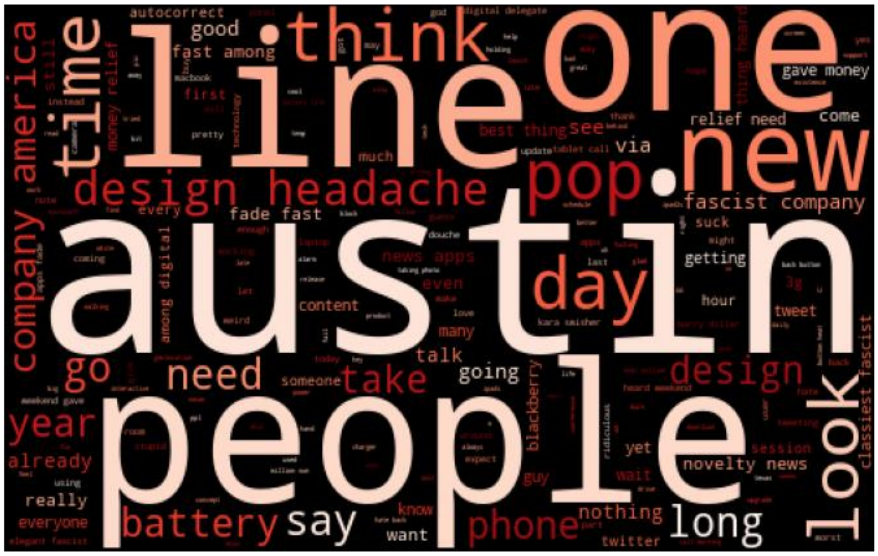

Fig.16. Negative word cloud: Apple.

We see names like Austin, people, new, think as most common words. They do not show any emotion, thus we can add them to the stop list and check for the common words again.

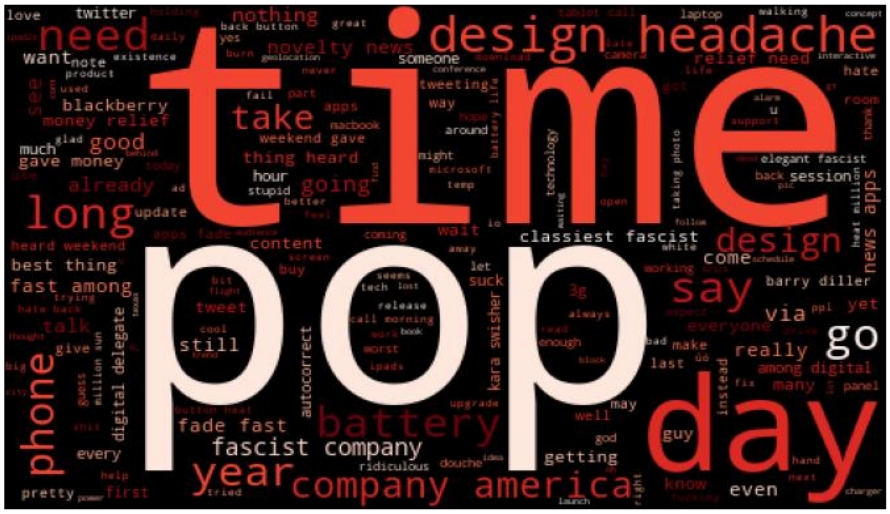

Fig.16. Negative word cloud: Apple.

The word cloud of negative terms directed at Apple products reveals several recurring pain points that customers associate with dissatisfaction, such as "battery life issues," "autocorrect frustrations," and specific design complaints. Terms like "nothing good," "design headache," and "suck" convey general displeasure, while more intense phrases such as "fascist company" and "classies fascist" reflect anger over Apple's perceived corporate control and restrictive policies. Other terms like "fade fast" and "stupid"

highlight concerns about product durability and functionality, while "gave money" indicates regret over the perceived lack of value for the price paid. These insights point to areas where Apple could improve, particularly in terms of product longevity, user experience, and corporate practices. However, the presence of positive words in the background of the cloud suggests that the classification of positive and negative emotions may not have been perfectly refined, which could compromise the accuracy and effectiveness of the sentiment analysis and the overall modeling process.

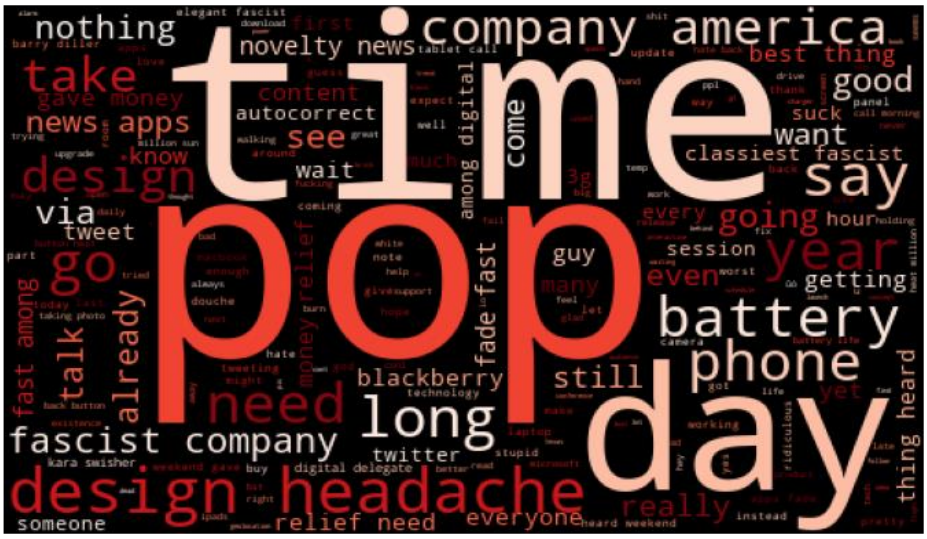

Fig.17. Negative word cloud: Google.

Similar to the analysis of negative emotions in the Apple product's word cloud, the word cloud for Google products does not clearly capture the strong negative sentiment. Words like time, battery, suck, "network," "product," "major social," "social network," "bing," "service," "launch," "lost way," "caring much," "search," and "really" are more neutral or ambiguous. While terms like "really" and "lost way" might imply some level of frustration, they lack the intensity typically associated with more overtly negative expressions. This suggests that the word cloud might not be fully reflective of the emotional tone or sentiment intended by users, and it may introduce some ambiguity into the analysis. Given these challenges, we proceed to explore and test various models that can more accurately predict the emotions conveyed in the tweets, refining the sentiment analysis process for better accuracy.

Modelling

To begin the analysis, we first filtered the data frame to access the two key columns: 'is there an emotion directed at a brand or product' (the target) and 'text tokenized lemmatized' (the feature). For binary classification, we filter for tweets that reflect either positive or negative emotions only. To prepare the text for further analysis, we convert the 'text tokenized lemmatized' column back into sentences to make it compatible with the TF-IDF vectorizer.

Next, we proceed with label encoding the target column, assigning a value of 1 to positive emotions and 0 to negative emotions. This allows us to set up the classification task for supervised learning. We will then use TF-IDF vectorization on the sentences in the 'text_tokenized_lemmatized' column to convert the text data into a numerical format that machine learning models can process. After preparing the data, we train a baseline model using Logistic Regression or Multinomial Naive Bayes as classification algorithms. We evaluate the performance of these models based on metrics such as accuracy, precision, recall, and f1-score.

These metrics are key to understanding how well the model is classifying tweets into positive and negative categories.

Following the baseline evaluation, we can perform hyper-parameter tuning to improve model performance. Using techniques like Grid Search or Randomized Search, we will fine-tune parameters such as regularization strength (for Logistic Regression) or smoothing parameters (for Naive Bayes). Next, we will explore unsupervised learning models like K-Means clustering for both binary classification (positive vs negative) and multiclass classification (positive, negative, and neutral). These models will allow us to explore patterns in the data further and assess how well they perform at classifying the tweets. After completing these steps, we will compare the results and identify the best-performing models for the project. Since the data is high-dimensional, we can use dimensionality reduction to project it to 2D for better visualization. The TF-IDF vectorizer produces a sparse matrix, and to reduce the dimensionality of this matrix, we can apply Truncated Singular Value Decomposition (SVD). Truncated SVD is a PCA-like algorithm that works well with sparse matrices. By setting n_components=100, we retain as much of the explained variation as possible while reducing the dimensions. To start, we will vectorize the 'text tokenized lemmatized' column using the TF-IDF vectorizer and then apply Truncated SVD for dimensionality reduction.

In this step, we used the KMeans algorithm as a baseline model. Since we were performing binary classification to categorize the tweets as either negative or positive, we specified the number of clusters (n_clusters) as 2. We then fitted the model to the data, which allowed us to identify the two clusters corresponding to the two sentiment categories. This served as our starting point for clustering analysis.

```
# Step 1: Convert the text data to TF-IDF features
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(df_model['text_tokenized_lemmatized'])

# X_tfidf is now a sparse matrix with TF-IDF features
# Check the shape of the TF-IDF matrix
print(X_tfidf.shape)
```

```
(3411, 4758)
```

```
# Applying TruncatedSVD for dimensionality reduction
# parse in the n_components = 100
n_components = 100
# Initialize the Truncated SVD
svd = TruncatedSVD(n_components=n_components, random_state=42)
# Fit the TruncatedSVD on the vectorized data
X_reduced = svd.fit_transform(X_tfidf)
```

Fig.18. Word Vectorization.

```
# Apply KMeans clustering on the TF-IDF data
from sklearn.cluster import KMeans

# Defining the number of clusters
n_clusters = 2

kmeans = KMeans(n_clusters=n_clusters, random_state=42)
# Training the model
y_kmeans = kmeans.fit_predict(X_tfidf)

## Print the cluster assignments for each document
print(f"Cluster Assignments: {y_kmeans}")
```

Fig.19. Application of Kmeans.

Once the model was trained, we used it to predict the cluster assignment for each data point using the fit_predict() method. This generated the predicted cluster labels for all the data points in the dataset. To evaluate the performance of the KMeans model, we analyzed the cluster assignments in the context of the actual labels, using metrics such as accuracy, silhouette score, and the Davies-Bouldin index. These evaluation metrics helped us assess how well the model was able to separate the data into meaningful clusters and whether the results were valid for our data.

```
# Import the metrics of evaluating the kmeans model
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Step 4: Evaluate the model using different metrics

# Inertia
print(f"Inertia: {kmeans.inertia_}")

# Silhouette Score (higher is better)
silhouette_avg = silhouette_score(X_tfidf, y_kmeans)
print(f"Silhouette Score: {silhouette_avg}")

# Davies-Bouldin Index (lower is better)
db_index = davies_bouldin_score(X_tfidf.toarray(), y_kmeans)
print(f"Davies-Bouldin Index: {db_index}")

from sklearn.metrics import adjusted_rand_score

ari = adjusted_rand_score(df_model['Emotion_encoded'], y_kmeans)
print(f"Adjusted Rand Index: {ari}")
```

```
Inertia: 3316.040427551093
Silhouette Score: 0.00971701143845769
Davies-Bouldin Index: 6.795870199468917
Adjusted Rand Index: -0.06948142053248782
```

Fig.20. Kmeans Performance metrics.

A high inertia in our results suggests that the clustering is not well-formed, which indicates that the model might benefit from adjustments, such as trying a different number of clusters. The Silhouette Score, which is close to 0, further points to poorly defined clusters, with data points being near the boundaries of different clusters. Ideally, we would expect a higher score (above 0.5) to indicate better separation between clusters. Additionally, the high Davies-Bouldin Index of 6.77 signals that the clusters are not well-separated, suggesting significant overlap between them, which points to poor clustering performance. The negative Adjusted Rand Index (ARI) of -0.069 reinforces the concern that the clustering is actually worse than random, signaling that the model is failing to capture meaningful patterns in the data. Given these results, it is clear that the clustering model is not suitable for our dataset, especially considering the overlap in emotional tones within the tweets. Based on these poor clustering results, we conclude that tuning this model further is unlikely to improve performance, and we opt to explore other unsupervised learning techniques, such as hierarchical agglomerative clustering, for better results.

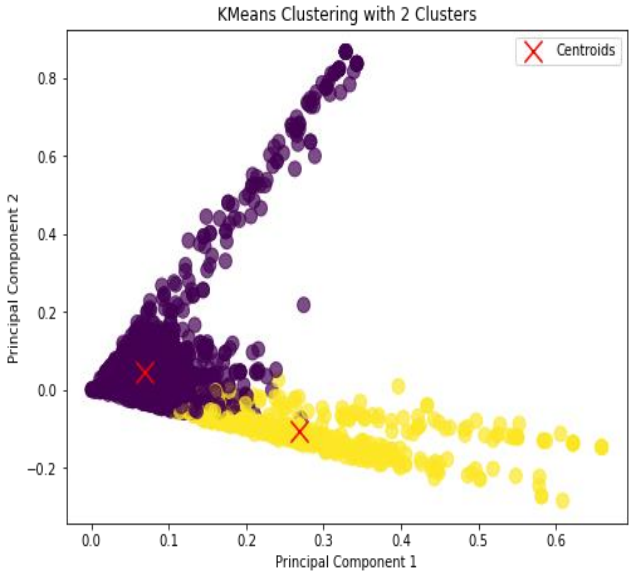

Fig.21. KMeans Clustering.

Hierarchical agglomerative

Hierarchical agglomerative clustering (HAC) is a type of unsupervised machine learning algorithm used to group data points into clusters based on their similarity. It is a hierarchical clustering method that builds a tree-like structure (called a dendrogram) to represent the hierarchy of clusters.

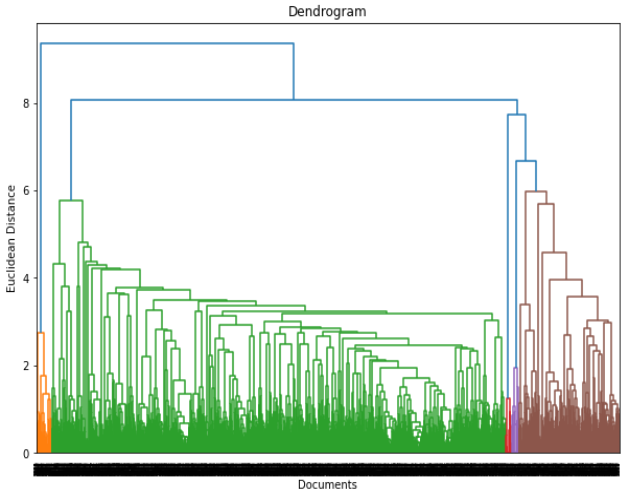

Fig.22. Dendrogram: Hierarchical clustering.

```python
from sklearn.metrics import silhouette_score

# Compute Silhouette Score
silhouette_avg = silhouette_score(X_reduced, y_agg_clust)
print(f"Silhouette Score: {silhouette_avg}")


from sklearn.metrics import davies_bouldin_score

# Compute Davies-Bouldin Index
db_index = davies_bouldin_score(X_reduced, y_agg_clust)
print(f"Davies-Bouldin Index: {db_index}")


from sklearn.metrics import normalized_mutual_info_score

# Assuming you have true labels (y_true)
nmi = normalized_mutual_info_score(df_model['Emotion_encoded'], y_a
print(f"Normalized Mutual Information (NMI): {nmi}")
```

```
Silhouette Score: 0.23445289613493026
Davies-Bouldin Index: 1.2317139939036996
Normalized Mutual Information (NMI): 0.0001445690469051779
```

Fig.23. Hierarchical clustering: Performance metrics.

A Silhouette Score of 0.2344 indicates poor clustering performance, suggesting that our clusters are not well-separated and that the points within each cluster are sparse. This implies that the clustering model has no clear boundaries, and the data points within each cluster are not tightly grouped together. A Davies-Bouldin Index (DBI) of 1.2317 is relatively high, which further supports the idea of overlapping data points. In general, a DBI score below 1 indicates well-separated clusters, but in our case, the high score suggests significant overlap between clusters. The Normalized Mutual Information (NMI) score of 0.0058 is very low, highlighting the weak correspondence between the true labels and the predicted clusters. This indicates that the clusters found by the model do not reflect the actual class labels, suggesting that the model is not capturing meaningful patterns in the data. Given these results and the overlap in the data points, we conclude that unsupervised learning is not effective for our dataset and decide to shift to supervised learning techniques, such as decision trees and logistic regression models, to better analyze the data.

Application of Supervised Machine Learning Models

Given the unsatisfactory results from the unsupervised machine learning models, primarily due to the significant impact of class imbalance in the dataset, we shifted to more appropriate supervised machine learning models. These models are better suited for our task, as we are working with labeled data, making it ideal for supervised learning. The first model we will implement is the Binary Classification Model using Multinomial Naive Bayes. This approach is well-suited for classifying data into two distinct classes based on observed features. It works by calculating the likelihood of each class given the observed features and then assigning the class with the highest probability. Despite its simplicity, Multinomial Naive Bayes is particularly effective in scenarios where the data is sparse and high-dimensional, making it a good fit for text classification tasks like ours. By applying this model, we aim to better handle the imbalance in the data and improve the classification results.

Multinomial Naive Bayes

1. The baseline model

```
Training Score: 0.87            Test Score: 0.85

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.86      0.04      0.08       137
           1       0.85      1.00      0.92       716

    accuracy                           0.85       853
   macro avg       0.85      0.52      0.50       853
weighted avg       0.85      0.85      0.78       853
```

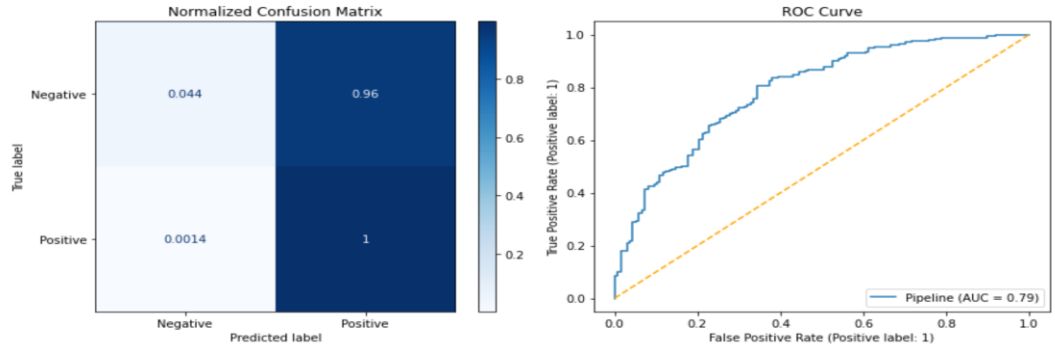

Fig.24.MNB baseline model: Performance metrics

The Training Score of 0.87 indicates that the model correctly classified 87% of the training data, suggesting it learned the patterns well. The Test Score of 0.85 reflects that 85% of the test data was correctly classified, and the slight drop from the training score indicates that the model generalizes well without overfitting. The precision for class 0 (negative class) is 0.86, meaning the model correctly identified 86% of negative cases, while the precision for class 1 (positive class) is 0.85, indicating a similar performance for positive class prediction. However, the recall for class 0 is very low at 0.04, implying the model struggles to identify negative instances, likely predicting the positive class more often. On the other hand, the recall for class 1 is perfect at 1.00, showing the model's excellent ability to identify positive tweets. To improve the model, addressing the class imbalance is crucial to boost recall for class 0. Additionally, hyper parameter tuning, such as experimenting with different smoothing parameters for the Multinomial Naive Bayes model, could further enhance performance by balancing precision and recall.

Although the model's accuracy in identifying positive tweets has decreased from 97% to 84%, the oversampling technique has significantly improved performance for negative tweets, increasing recall from 40% to 60%. Oversampling allows the model to better learn from the underrepresented negative class, leading to a more balanced classification. However, this improvement in recall for negative tweets comes at the cost of overfitting, as the model performs well on the oversampled training data but struggles to generalize effectively to new data. To mitigate overfitting, we can explore methods like regularization, cross-validation, or fine-tuning the oversampling strategy. These techniques will help the model maintain its ability to generalize well to unseen data, thus improving overall performance.

```
Training Score: 0.94            Test Score: 0.8

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.42      0.60      0.49       137
           1       0.92      0.84      0.88       716

    accuracy                           0.80       853
   macro avg       0.67      0.72      0.69       853
weighted avg       0.84      0.80      0.82       853
```

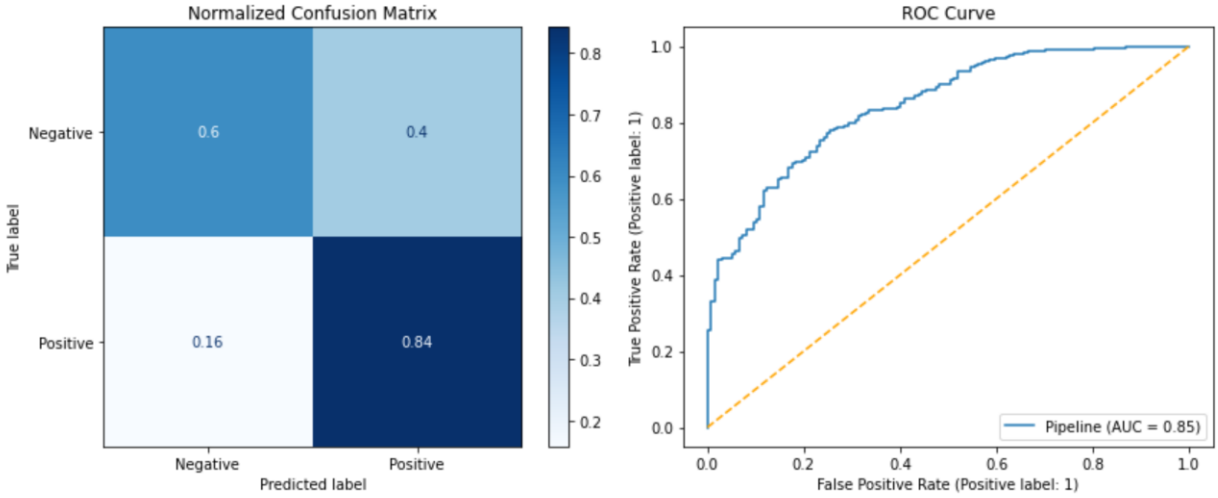

Fig.25. MNB Final/ Tuned model: Performance metrics

After tuning the parameters, the model's ability to identify negative tweets has remained consistent at 60%, maintaining the previous performance. However, this improvement in negative tweet recall comes at the cost of slightly reduced accuracy in identifying positive tweets, which dropped from 88% to 84%. This trade-off is expected, as the model was optimized to maximize the macro recall score during the grid search. The best-performing model among all the Multinomial Naive Bayes configurations was the one using tuned random oversampling, which achieved a macro recall score of 0.72. Having explored various configurations of the MNB model, the next step will be to evaluate a more complex model, such as the Random Forest Classifier, to see if it can provide even better results.

Random Forest Classifier

1. The baseline model

```
Training Score: 1.0            Test Score: 0.87

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.85      0.26      0.39       137
           1       0.87      0.99      0.93       716

    accuracy                           0.87       853
   macro avg       0.86      0.62      0.66       853
weighted avg       0.87      0.87      0.84       853
```

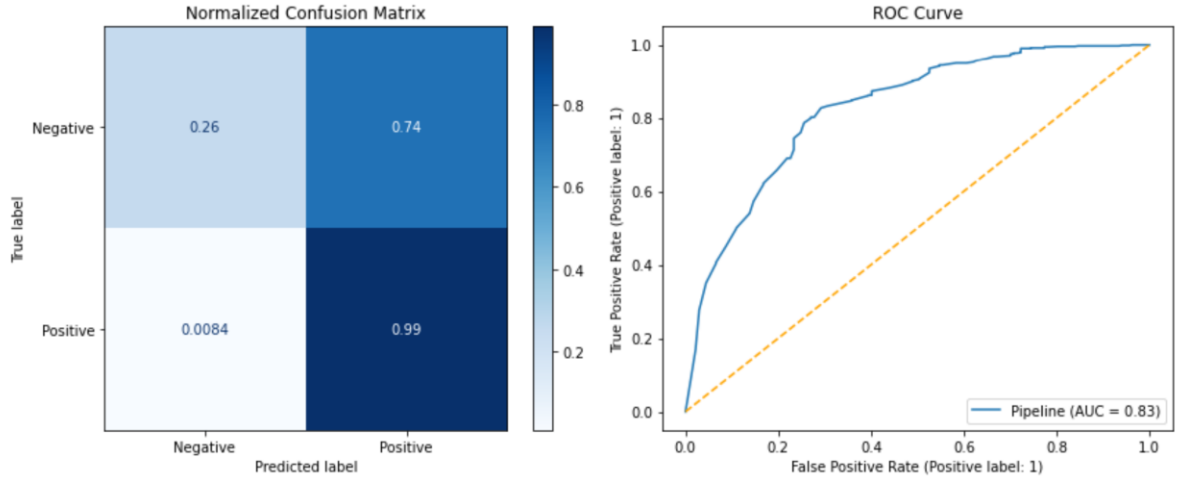

Fig.26. Random Forest Classifier/ Performance metrics

The models achieved an accuracy of 87%, but there was significant overfitting, as indicated by the perfect training score of 1.0. While they performed well in predicting positive tweets (class 1), with high precision of 0.87 and recall of 0.99, they struggled with the negative class (class 0). Although the precision for class 0 was decent at 0.85, the recall was alarmingly low at 0.26, meaning the models missed a significant portion of the actual negative instances. This imbalance in performance suggested that the models were biased toward predicting positive cases, which likely stemmed from overfitting and the class imbalance in the dataset. To address this, steps were taken to balance the models' performance across both classes, such as adjusting the sampling strategy, implementing regularization, and experimenting with other model architectures.

2. Tuned Final model

```
Training Score: 0.87            Test Score: 0.75

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.35      0.61      0.44       137
           1       0.91      0.78      0.84       716

    accuracy                           0.75       853
   macro avg       0.63      0.69      0.64       853
weighted avg       0.82      0.75      0.78       853
```

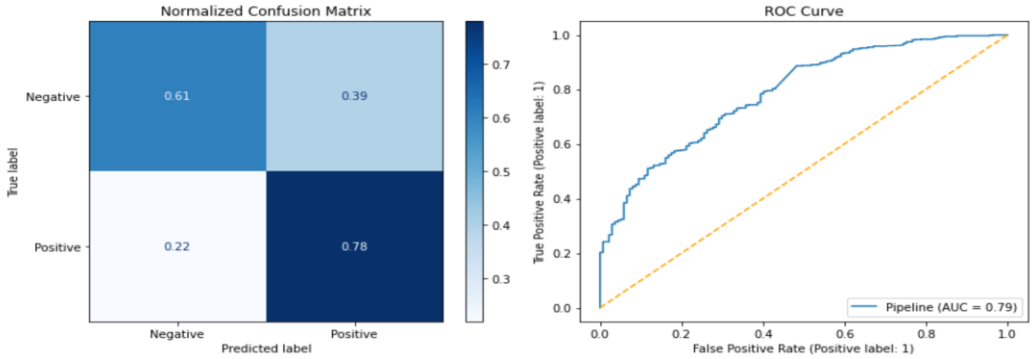

Fig.27. Random Forest Classifier: Final Tuned model, performance metrics

The model showed good performance in identifying positive tweets, with a high precision of 0.91 and recall of 0.78, but struggled significantly with identifying negative tweets, as reflected by a low precision of 0.35 and moderate recall of 0.61. This imbalance in performance is likely a result of the class imbalance in the dataset, where positive tweets are more prevalent. The overall accuracy was 0.75, but the model exhibited overfitting, as indicated by the higher training score (0.87) compared to the test score (0.75). Despite these issues, the model was improved through hyperparameter tuning and undersampling of the majority class. This model provided the best results so far, but to further enhance performance, we can experiment with other models, such as logistic regression.

Binary Classification: Logistic Regression

1. The baseline model

```
Training Score: 1.0            Test Score: 0.85

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.54      0.53      0.54       137
           1       0.91      0.91      0.91       716

    accuracy                           0.85       853
   macro avg       0.73      0.72      0.73       853
weighted avg       0.85      0.85      0.85       853
```

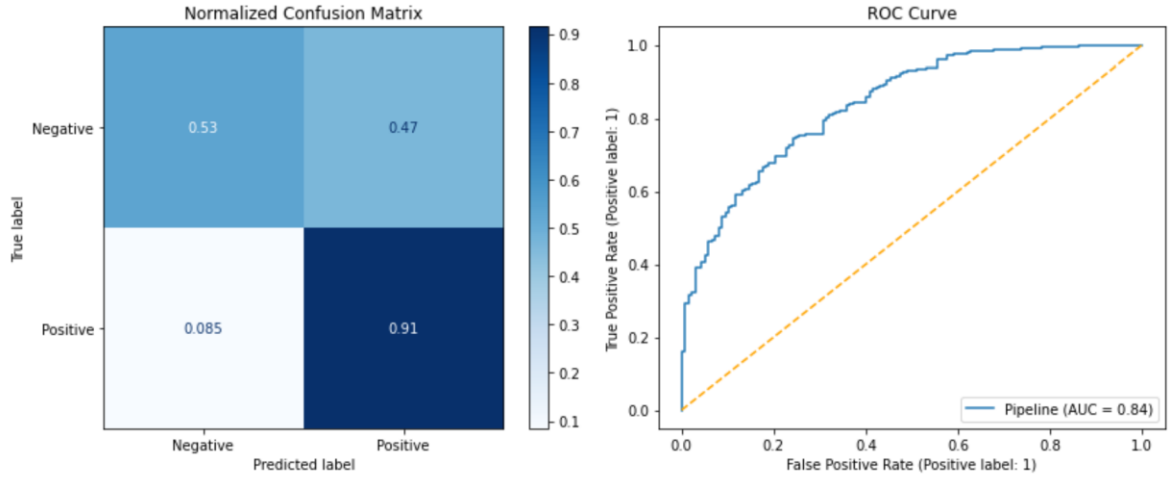

Fig.28. Logistic Regression Baseline Model, performance metrics

The logistic regression model demonstrated a perfect training score of 1.0, indicating overfitting, as it performed flawlessly on the training data but dropped to 0.85 on the test set, suggesting it struggles to generalize. The classification report revealed that the model performed much better on class 1 (positive class), with high precision (0.91) and recall (0.91), yielding a strong F1 score of 0.91. However, for class 0 (negative class), the model faced challenges, with lower precision (0.54) and recall (0.53), indicating difficulty in accurately identifying negative instances. While the model is strong at predicting positive tweets, improvements are necessary for detecting negative tweets, and addressing overfitting will be crucial for improving generalization across both classes.

```
Training Score: 0.83            Test Score: 0.74

CLASSIFICATION REPORT
--------------------------------------------
              precision    recall  f1-score   support

           0       0.36      0.77      0.49       137
           1       0.94      0.74      0.83       716

    accuracy                           0.74       853
   macro avg       0.65      0.76      0.66       853
weighted avg       0.85      0.74      0.78       853
```

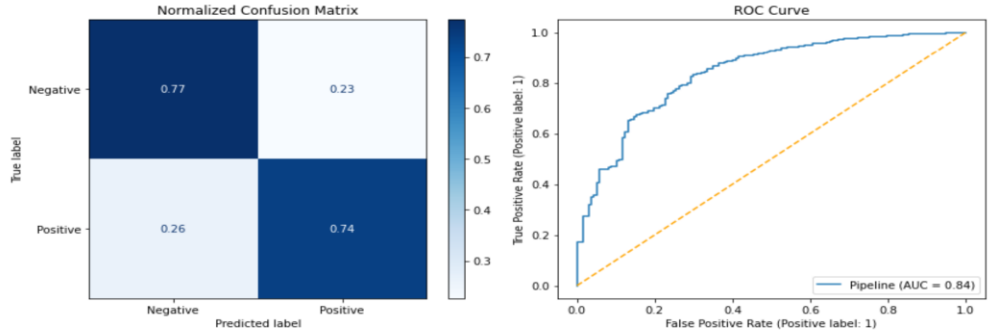

Fig.29. Logistic Regression Tuned Final Model, performance metrics

The model achieved a training score of 0.83 and a test score of 0.74, indicating moderate overfitting, though not as severe as in previous models. The classification report shows excellent performance on class 1 (positive class), with high precision (0.94) and an F1 score of 0.83. However, the model underperforms on class 0 (negative class), with low precision (0.36) and a lower F1 score of 0.49, despite a high recall of 0.77. This suggests that the model is confident in predicting positive cases but struggles to correctly identify negative ones, particularly when it comes to precision. The overall accuracy of 0.74 highlights the imbalance in performance, indicating that adjustments—such as resampling or adjusting class weights—could improve its detection of negative tweets. With an area under the curve (AUC) of 0.84, the model demonstrates strong overall performance, and we will now proceed to apply this as a foundation for multiclass classification in the next phase of the project.

Multiclass Classification. - Logistic Regression

1. The baseline Model.

```
Training Score: 0.89 | Test Score: 0.64

CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.35      0.42      0.38       154
           1       0.70      0.69      0.69       954
           2       0.64      0.62      0.63       716

    accuracy                           0.64      1824
   macro avg       0.56      0.58      0.57      1824
weighted avg       0.64      0.64      0.64      1824
```

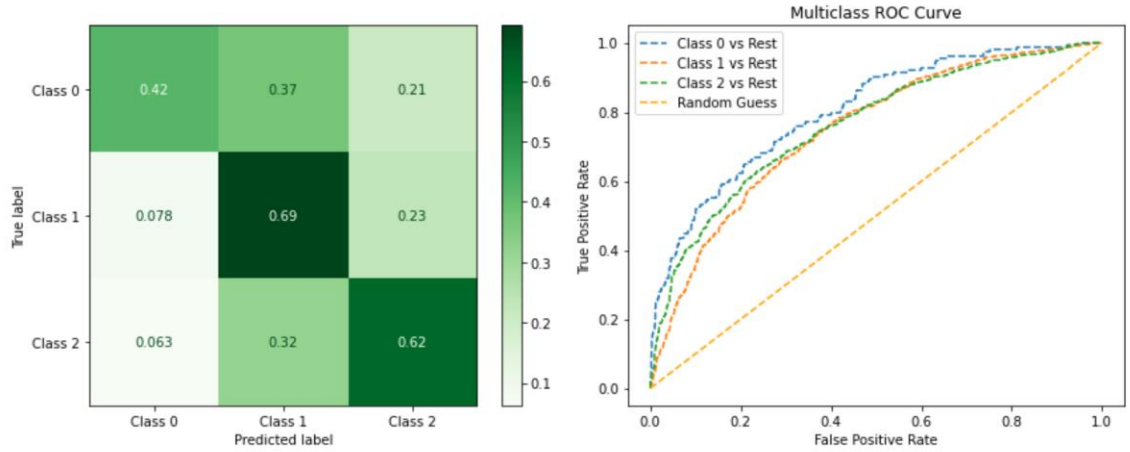

Fig.30. Logistic Regression Baseline Model, Multi class performance metrics

The baseline model showed a training score of 0.89 and a test score of 0.64, indicating that the model performed well on the training data but struggled to generalize to new, unseen data. This performance drop suggested overfitting, where the model memorized the training data but failed to accurately predict on the test set. In the classification report, the precision for class 0 (negative sentiment) was 0.35, and recall was 0.42, indicating that the model had difficulty identifying negative tweets accurately. For class 1 (neutral sentiment), the precision was 0.70, and recall was 0.69, showing decent performance but still room for improvement. Class 2 (positive sentiment) had a precision of 0.64 and recall of 0.62, which was moderate but again far from ideal. The overall accuracy of 0.64 suggested that the model was not effectively distinguishing between the three sentiment classes, and there was significant room for improvement in both precision and recall across all classes.

2. Final Tuned model

```
Training Score: 0.61 | Test Score: 0.54

CLASSIFICATION REPORT
-------------------------------------------
              precision    recall  f1-score   support

           0       0.22      0.62      0.32       154
           1       0.67      0.54      0.60       954
           2       0.59      0.51      0.55       716

    accuracy                           0.54      1824
   macro avg       0.49      0.56      0.49      1824
weighted avg       0.60      0.54      0.56      1824
```

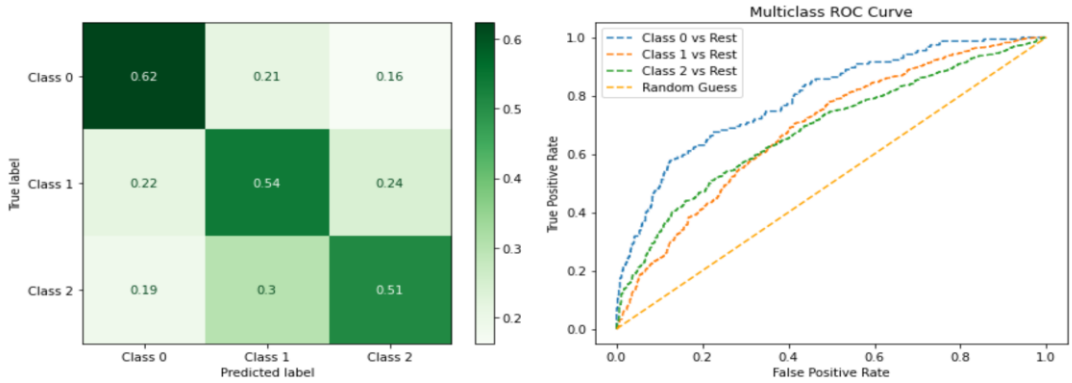

Fig.31. Logistic Regression Tuned Final Model, Multi class performance metrics.

The model showed a training score of 0.61 and a test score of 0.54, indicating moderate performance on the training set but significant struggles with generalizing to the test data. In the classification report, the precision for class 0 (negative sentiment) was 0.22, with a recall of 0.62, suggesting that the model identified a decent portion of the negative tweets but misclassified many, resulting in numerous false positives. For class 1 (neutral sentiment), the model performed better with a precision of 0.67 and a recall of 0.54, indicating that it correctly identified many neutral tweets, though there was still room for improvement. Class 2 (positive sentiment) had a precision of 0.59 and recall of 0.51, reflecting moderate performance but indicating struggles with both precision and recall for positive tweets. The overall accuracy of 0.54 showed that the model had limited success in predicting sentiment across all classes. The random under sampled model performed worse in terms of test score, confirming that the hyper-parameter tuned model with a test score of 0.62% was the preferred choice for better performance.

Conclusion.

In conclusion, we successfully met both the primary and secondary objectives set for this sentiment classification project. The Logistic Regression model emerged as the best-performing model, outperforming others in terms of both accuracy and generalization, providing a robust solution for classifying tweets as positive or negative towards Apple and Google's products and services. Although we explored unsupervised machine learning approaches, they proved ineffective for our dataset, failing to deliver the desired classification results. Through thorough data preprocessing and cleaning, we effectively removed noise from the tweet text, including hashtags, mentions, and URLs, ensuring a cleaner input for sentiment analysis. We also identified the most positive and negative words associated with Apple and Google's products, offering valuable insights into public sentiment. These insights will be crucial for businesses looking to improve customer satisfaction and refine their marketing strategies.

Recommendations

1. Improve Product Durability and Battery Life: Negative sentiment analysis reveals complaints about battery life and product durability. Apple could focus on improving these aspects in future product iterations to reduce frustration among users

2. Explore context-driven sentiment analysis: Since both companies' word clouds show a mix of emotional tones (positive, negative, and neutral) and some ambiguity, a context-sensitive sentiment analysis approach would be useful. This would involve analyzing the context in which terms like "really," "launch," and "social network" are used to better understand how users perceive these aspects of the brands.

3. Address Perceptions of Corporate Control**:** Strong negative emotions tied to terms like "fascist company" reflect frustrations with Apple's perceived corporate practices, such as restrictions on customization or the App Store policies.

4. Communicate Better on Value for Price**:** Words like "gave money" suggest some users feel they aren't receiving adequate value for the premium prices they pay.

Next Steps

Collect more data: To improve the model's generalizability, we should gather a larger and more diverse dataset from Twitter, especially focusing on increasing the number of negative tweets that are currently under represented.

Refine labeling process: We should establish clear, objective labeling guidelines with examples to reduce subjectivity and ensure consistent classification, especially for challenging cases like sarcasm or mixed sentiments.

Use consensus labeling: Implementing an approach where multiple annotators label each tweet and taking the average sentiment label could mitigate individual biases and improve the quality of the labels.

Consider contextual analysis: Since tweet context can affect sentiment interpretation, integrating contextual understanding through advanced techniques like BERT or other NLP models could improve the accuracy of sentiment classification.