Phase 4 Project -

Natural Language Processing Model for Twitter Reviews on Android, Apple, and Google

Name: Andrew Levinton Student Pace: Self Pace Instructor name: Ahbhineet Kukarni

Tech Stack

- Python
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- Natural Language Toolkit

Business Problem

In this study, we will be analyzing tweets from twitter to help present findings on sentiment towards product related to Android, Apple, and Google. Many companies want to keep track of public sentiment about various products. Positive sentiment might drive production for creators or inventory management for retailers. It might drive the attention of influencers, news organizations, and consumers. Twitter is a platform where individuals and companies express themselves and have conversations and is full of information about how people feel about particular products.

I. Business Understanding

The goals of the business problem will be:

1. Analyze tweets from twitter to gain insight of sentiment towards various products.
2. Determine which features of tweets best show as the best predict if tweets can be categorized as positive, negative, or neither.
3. Maximize positive sentiment for products from the following companies.

Business Questions to consider:

1. Do customers have positive or negative sentiment towards Apple, Android, or Google products?
2. Which products might retailers want to keep in their stores to maxmize profit in inventory management?
3. Which products might influencers want to highlight in their posts to gain the most attraction?

The problem with sentiment

Emotions are also relative. Consdier the phrase, "iPhones are better than Android phones." Is this a positive or a negative tweet? It depends! For this reason purposeful data preprocessing will be essential to building data categorization before training it on the model itself. There is a LOT of cleaning that needs to occur, with tweets containing many various symbols and characters that will distract the model from providing an accurate prediction.

II. Data Understanding

In [1]:
```
#general libraries
import pandas as pd
import numpy as np
import random

import os

# machine learning modeling
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTrans
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, classific
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.model_selection import learning_curve
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
import xgboost as xgb
from xgboost import XGBClassifier

# nlp word processing
import re
from nltk import pos_tag
from nltk.tokenize import regexp_tokenize, word_tokenize, RegexpTokenizer
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer


#plotting
import seaborn as sns
import matplotlib.pyplot as plt

# Pipelines
from imblearn.pipeline import Pipeline, make_pipeline

# SMOTE
from imblearn.over_sampling import SMOTE

pd.set_option('display.max_colwidth', None)

plt.style.use('ggplot')
```

Encodings

In the EDA, I have created a list of encodings (utf-8, latin-1, utf-16) to try. The code attempts to read the CSV file using each encoding until it succeeds or exhausts all the encodings. Once the successful encoding is found, the loop is exited, and the DataFrame (tweet_df) is displayed. This process is in the EDA version of this notebook. If you run this code vs the one in the EDA, it will produce the same results.

Importing new data

After initial EDA on the `tweet_df` dataset, the data for a particular class was severely lacking and more data was added in an attempt to improve the training data of the model, and also a side effect was to make a prettier wordcloud :)

Note

In the EDA there is more thorough processing of each dataframe, including renaming of columns, filling of missing values, etc. To look at the full process, explore the EDA for more detail. If you run this notebook vs the EDA, it will produce the same results.

In [2]:
```
tweet_df = pd.read_csv('./data/judge-1377884607_tweet_product_company.csv', encoding='la
tweet_df_appl = pd.read_csv('./data/Apple-Twitter-Sentiment-DFE.csv', encoding='latin-1'
```

In [3]:
```
tweet_df_appl['product'] = 'Apple'
tweet_df_appl = tweet_df_appl[['text','product','sentiment']]
tweet_df_appl.columns = ['tweet','product','sentiment']
tweet_df_appl.head()
```

Out[3]:

	tweet	product	sentiment
0	#AAPL:The 10 best Steve Jobs emails ever...http://t.co/82G1kL94tx	Apple	3
1	RT @JPDesloges: Why AAPL Stock Had a Mini-Flash Crash Today $AAPL #aapl\nhttp://t.co/hGFcjYa0E9	Apple	3
2	My cat only chews @apple cords. Such an #AppleSnob.	Apple	3
3	I agree with @jimcramer that the #IndividualInvestor should own not trade #Apple #AAPL, it's extended so today's pullback is good to see	Apple	3
4	Nobody expects the Spanish Inquisition #AAPL	Apple	3

In [4]:
```
#rename columns
tweet_df.columns = ['tweet','product','sentiment']
# Display the DataFrame
tweet_df.head()
```

Out[4]:

	tweet	product	sentiment
0	.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead! I need to upgrade. Plugin stations at #SXSW.	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate for its design. Also, they're giving free Ts at #SXSW	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/conferences) & Matt Mullenweg (Wordpress)	Google	Positive emotion

In [5]:
```
tweet_df['sentiment'].value_counts()
```

Out[5]:
```
No emotion toward brand or product    5389
Positive emotion                      2978
Negative emotion                       570
I can't tell                           156
Name: sentiment, dtype: int64
```

Checking for missing values

In [6]:
```
tweet_df.isnull().sum()
```

Out[6]:
```
tweet           1
product      5802
sentiment       0
dtype: int64
```

Addressing Product Values

In the dataframe we see a a large amount of missing product values as well as some varying names in the companies. For example, we see that 'Apple' has certain product values like 'iPad' or 'iPhone App' and Google has product values like 'Other Google Product or Service'.

Since we are just interested in analyzing the sentiment towards the companies themselves, the product values are changed to just be associated with Apple, Google, and Android. This is addressed by first filling the product column by writing a function from word text in the assosicated tweets then renaming the rest of the columns that contain the various product contained by these three companies.

Initial Obersvations

We see that we have three columns available in the dataframe:

1. tweet - The text of the tweet. We see that this column contains the "review" or tweet about product from each customer.
2. product - This column represents the product or brand from the tweet.
3. sentiment - This column represents the sentiment or emotion toward the brand or product.

The **sentiment** will be our target column.

Functions

The functions shown below are developed to aid in data cleaning and preprocessing. The purpose is to simplify the data to enable a more accurate spread of the data distribution as well as avoid repeated code within the notebook.

The code provided includes several functions and operations on the tweet_df DataFrame. Here's a brief explanation of each function and operation:

1. `Renaming Sentiment Column Values` : This operation uses a lambda function and string comparisons to rename the values in the 'sentiment' column of the tweet_df DataFrame. It converts values like 'negative emotion' to 'negative', 'positive emotion' to 'positive', and so on.
2. `Renaming Product Column Values` : Similar to the sentiment column renaming, this operation uses a lambda function and string comparisons to rename the values in the 'product' column of the tweet_df DataFrame. It converts various forms of 'iPhone', 'iPad', 'Apple', 'Android', 'Google', and other variations to consistent labels like 'Apple', 'Android', 'Google', and 'Product Unknown' for missing values.
3. `Filling Null and Missing Values` : This operation fills the missing values in the 'product' column with the label 'Product Unknown' using fillna(), and then removes any remaining rows with null values using dropna().
4. `Plot Distribution Function` : This function, plot_distribution, takes a DataFrame (df) and a column name (col_name) as input. It generates a histogram plot of the distribution of values in the specified column. It sets the title, x and y labels, and shows the plot using matplotlib.

5. `Filling Unknown Product Values` : This operation fills the unknown 'product' values in the 'tweet' column of the DataFrame using a lambda function and string checks. It assigns labels like 'Apple', 'Google', 'Android', or 'Unknown' based on the presence of specific keywords in the 'tweet' column.

In [7]:

```
#renaming sentiment column values
tweet_df['sentiment'] = tweet_df['sentiment'].str.lower().apply(lambda x: 'negative' if x
                                                                else 'positive'
                                                                else 'none' if
                                                                else x)
#renaming product values
tweet_df['product'] = tweet_df['product'].str.lower().apply(lambda x: 'Apple' if x=='iph
                                                                else 'Android'
                                                                else 'Google'
                                                                else x)
#filling nulls of missing values
tweet_df['product'].fillna('Product Unknown', inplace = True)
tweet_df.dropna(inplace = True)

tweet_df_appl['sentiment'] = tweet_df_appl['sentiment'].apply(lambda x: 'negative' if x
                                                               else 'none' if x == '3'\
                                                               else 'positive' if x == '5
                                                              else 'unknown')

#function for plotting distributions
def plot_distribution(df, col_name):
    sentiment_counts = df[col_name].value_counts()
    df[col_name].hist(figsize=(10,6))
    plt.title(f'{col_name.capitalize()} Distribution')
    plt.tight_layout()
    plt.xticks(rotation=45)
    plt.ylabel('count')
    plt.show()


#filling unknown product values
tweet_df['product'] = tweet_df['tweet'].str.lower().apply(lambda x: 'Apple' if 'ipad' in
                                                          or 'apple' in x or 'iphone' in
                                                          or 'itunes' in x or 'mac' in x
                                                          or 'macbook' in x\
                                                          else 'Google' if 'google' in x
                                                          else 'Android' if 'android' in
                                                          else 'Unknown')

def save_text_file(df, df_name):
    # Get the current working directory
    current_directory = os.getcwd()

    # Concatenate the text data
    text_data = df['tweet_text_cleaned'].str.cat(sep=' ')

    # Define the file name
    file_name = f'{df_name}.txt'

    # Combine the directory path and file name
    output_file_path = os.path.join(current_directory, file_name)

    # Writing the text to the file
    with open(output_file_path, 'w', encoding='utf-8') as file:
        file.write(text_data)

    print(f"Text data saved to '{output_file_path}'.")


def plot_cm_senti(model, target_val,target_hat):
    # Obtain the confusion matrix
    confusion_mat = confusion_matrix(target_val,target_hat)
    # Display the confusion matrix
    # Define class labels
```

```
    class_labels = ["Positive", "Negative", "None"]
    plt.figure(figsize=(8, 6))
    sns.heatmap(confusion_mat, annot=True, cmap="Blues", fmt="d")
    plt.title(f"Prediction Accuracy: {round(accuracy_score(target_val, target_hat)*100, 
    plt.xlabel("Predicted Labels", fontsize=15)
    plt.ylabel("True Labels", fontsize=15)
    plt.xticks([0.5,1.5, 2.5], class_labels, fontsize=15)
    plt.yticks([0.5,1.5, 2.5], class_labels, fontsize=15)
    plt.show()

def classify(target_test, target_pred):
    # Generate the classification report
    report = classification_report(target_test, target_pred)

    # Print the classification report
    print("Classification Report:")
    print(report)

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

sw = stopwords.words('english')
'''Initializes the sw variable with a list of stopwords from the NLTK library.
Stopwords are commonly used words (such as articles, prepositions, etc.)
that often don't carry much meaning in a text and can be safely ignored during analysis.

sw.extend(['link', 'rt', 'sxsw', 'get', 'google', 'apple', 'iphone', 'ipad'])
'''Additional words ('link', 'rt', 'sxsw', 'get', 'google', 'apple', 'iphone', 'ipad')
are appended to the list of stopwords. These words are typically specific to the context
or domain and are considered irrelevant for analysis.'''

punctuation = '!"$%&\'()*+,-./:;<=>?[\\]^_`{|}~“!#'
'''Defines a string containing various
punctuation marks and special characters that need
to be removed from the text.'''

no_accents_re = re.compile('^[a-z]+$')
'''This line compiles a regular expression pattern that matches
words consisting only of lowercase letters without any accents.'''

accents = ['á', 'â', 'ã', 'à', 'å', 'ª', 'ç', 'è', '¼', '¾', 'î', 'ï', 'ì', 'ó', 'ö', 'ð
'''This list contains various accent characters that need to be removed from the words.'

twitter_re = re.compile('[@][a-zA-Z]*')
'''Compile a regular expression pattern that matches strings consisting only of digits.'

num_re = re.compile('^\d+$')
'''The text string (txt) is split into a list of words based on whitespace.'''

def remove_punctuation(text, punctuation):
    for char in punctuation:
        text = text.replace(char, '')
    return text

def remove_twitter_handles(text, twitter_re):
```

```
    words = text.split()
    filtered_words = []
    for word in words:
        if not twitter_re.match(word):
            filtered_words.append(word)
    return ' '.join(filtered_words)

def remove_numbers(text, num_re):
    words = text.split()
    filtered_words = []
    for word in words:
        if not num_re.match(word):
            filtered_words.append(word)
    return ' '.join(filtered_words)

def remove_accents(text, accents, no_accents_re):
    words = text.split()
    filtered_words = []
    for word in words:
        if no_accents_re.match(word):
            filtered_words.append(word)
    return ' '.join(filtered_words)

def txt_clean(txt, stop_words=sw):
    t = txt.lower()
    t = remove_punctuation(t, punctuation)
    t = remove_twitter_handles(t, twitter_re)
    t = remove_numbers(t, num_re)
    t = remove_accents(t, accents, no_accents_re)
    t = t.split()
    t = [w for w in t if w not in stop_words and w]
    t = pos_tag(t)
    t = [(w[0], get_wordnet_pos(w[1])) for w in t]
    lem = WordNetLemmatizer()
    t = [lem.lemmatize(w[0], w[1]) for w in t]
    return ' '.join(t)

pd.set_option('display.max_colwidth', None)
```

In [8]:
```
tweet_df.head()
```

Out[8]:

	tweet	product	sentiment
0	.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead! I need to upgrade. Plugin stations at #SXSW.	Apple	negative
1	@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate for its design. Also, they're giving free Ts at #SXSW	Apple	positive
2	@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.	Apple	positive
3	@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw	Apple	negative
4	@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/conferences) & Matt Mullenweg (Wordpress)	Google	positive

In [9]:
```
tweet_df_appl.head()
```

Out[9]:

	tweet	product	sentiment
0	#AAPL:The 10 best Steve Jobs emails ever...http://t.co/82G1kL94tx	Apple	none
1	RT @JPDesloges: Why AAPL Stock Had a Mini-Flash Crash Today $AAPL #aapl\nhttp://t.co/hGFcjYa0E9	Apple	none
2	My cat only chews @apple cords. Such an #AppleSnob.	Apple	none
3	I agree with @jimcramer that the #IndividualInvestor should own not trade #Apple #AAPL, it's extended so today's pullback is good to see	Apple	none
4	Nobody expects the Spanish Inquisition #AAPL	Apple	none

In [10]:
```
tweet_df = pd.concat([tweet_df,tweet_df_appl])
tweet_df.head()
```

Out[10]:

	tweet	product	sentiment
0	.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead! I need to upgrade. Plugin stations at #SXSW.	Apple	negative
1	@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate for its design. Also, they're giving free Ts at #SXSW	Apple	positive
2	@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.	Apple	positive
3	@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw	Apple	negative
4	@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/conferences) & Matt Mullenweg (Wordpress)	Google	positive

In [64]:
```
tweet_df.shape
```

Out[64]:
```
(12896, 4)
```

In [11]:
```
tweet_df['sentiment'].value_counts()
```

Out[11]:
```
none        7706
positive    3401
negative    1789
unknown       82
Name: sentiment, dtype: int64
```

Previously, before the second dataframe was added, we had a very small sample size of negative tweets making the models run on them very spotty. This did not improve the model as much as I would have liked, but I had to run with it.

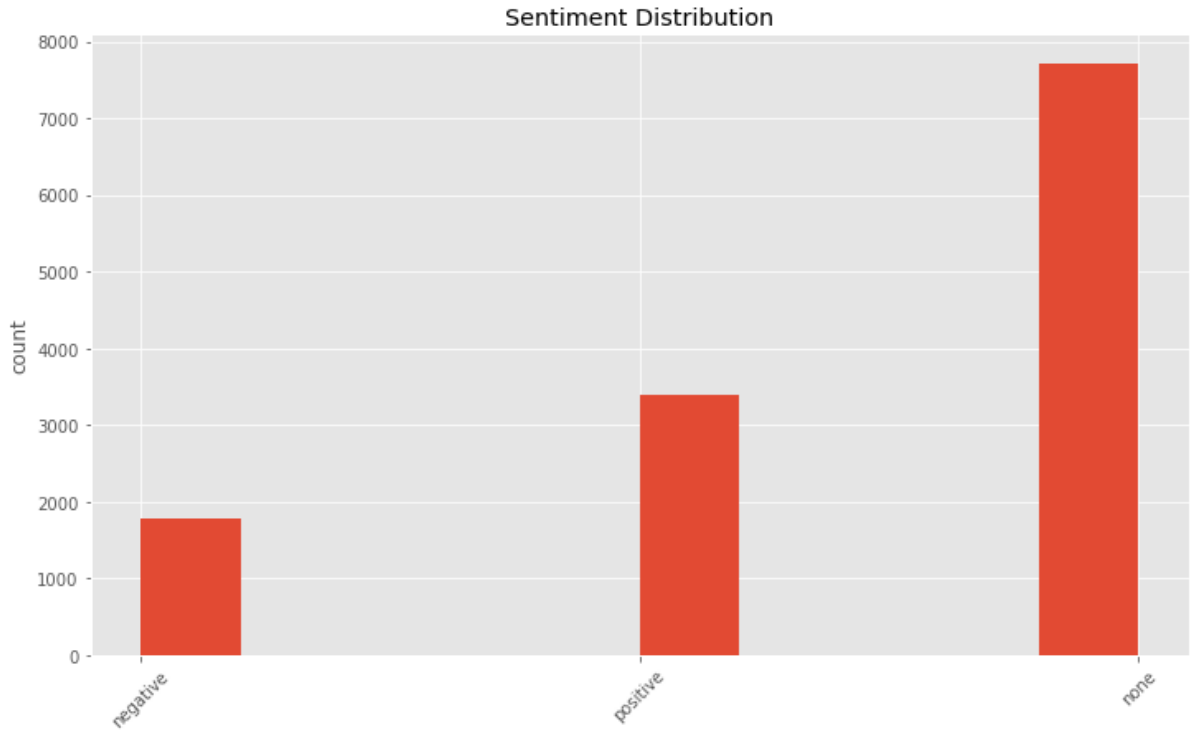
We removed the 'unknown' category as it only omitted a very small percentage of the dataframe, and our target was the three classes.

In [12]:
```
tweet_df = tweet_df[(tweet_df['sentiment'] == 'none') |\
                    (tweet_df['sentiment'] == 'positive') |\
                    (tweet_df['sentiment'] == 'negative')]
```

In [13]:
```
tweet_df['sentiment'].value_counts()
```

Out[13]:
```
none        7706
positive    3401
negative    1789
Name: sentiment, dtype: int64
```

In [14]:
```
plot_distribution(tweet_df, 'sentiment')
```



Understanding Class Imbalance

By examining the distribution of the target column, you can determine if there is a class imbalance issue. Class imbalance occurs when one class is significantly more prevalent than the others. This can impact the model's performance, as it may become biased towards the majority class. Identifying class imbalance early on helps in choosing appropriate strategies to handle it.

SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a popular technique used in the context of imbalanced classification problems. Imbalanced classification occurs when the distribution of classes in the training data is highly skewed, meaning one class (the minority class) has significantly fewer samples compared to the other class(es) (the majority class).

The main use case and purpose of SMOTE are to address the imbalanced class distribution and improve the performance of machine learning models, especially in scenarios where the minority class is of particular interest and misclassification of the minority class is more critical.

Model Selection and Evaluation

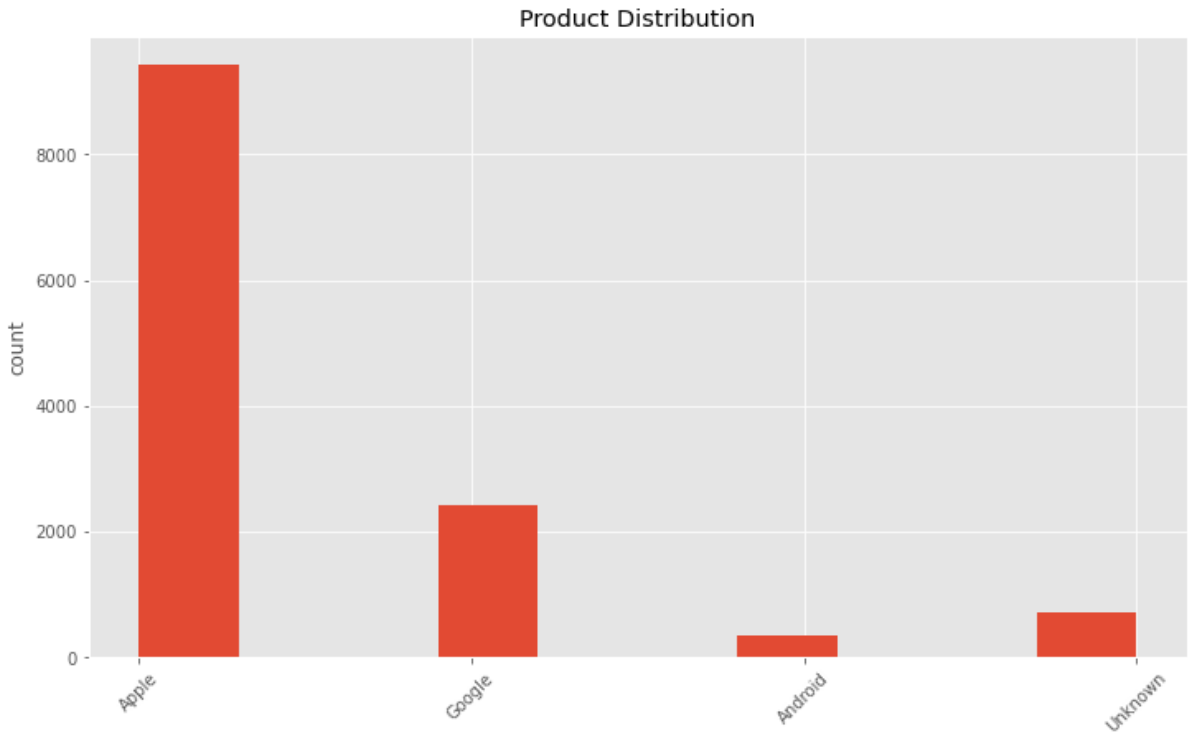
The distribution of the target column influences the choice of evaluation metrics and the selection of an appropriate model. Understanding the distribution allows for better evaluation and comparison of different models, as well as assessing their effectiveness in capturing the desired sentiment patterns.

Observing the product column

In [15]:
```
tweet_df['product'].value_counts()
```

Out[15]:
```
Apple      9417
Google     2417
Unknown     717
Android     345
Name: product, dtype: int64
```

In [16]:
```
plot_distribution(tweet_df, 'product')
```



Product Distribution

- Apple is the most discussed product: With a count of 5613, Apple appears to be the most frequently mentioned product in the dataset. This suggests that Apple products may be more popular or widely used, or they could simply be more often associated with specific sentiments or opinions.
- Google has significant attention: Google comes in second with a count of 2417 mentions. This indicates that Google products or services are also a major topic of discussion in the dataset, though they are not as dominant as Apple. The analysis might delve into understanding the sentiment patterns around Google products.
- Unknown category requires investigation: The category labeled as "Unknown" with 717 mentions is a concern since the sentiment analysis results may not be conclusive without knowing the actual products mentioned. It could be due to insufficient data or misclassification during data preprocessing. Further investigation is needed to identify and categorize these products to ensure a comprehensive analysis.
- Android has a relatively lower count: With only 345 mentions, Android appears to have a smaller presence in the sentiment analysis. This could suggest that Android-related discussions are less frequent or less emotionally charged in comparison to Apple and Google products.

Sentiment breakdown by target

Analyzing the sentiment breakdown by product is crucial for assessing product performance, identifying issues, and designing targeted improvement strategies. By evaluating the sentiment distribution for each product, organizations can understand customer satisfaction levels, pinpoint areas of improvement, and allocate resources effectively. This breakdown aids decision-making, allows for benchmarking against competitors, and helps in prioritizing efforts for product development, marketing, and customer support. By leveraging sentiment breakdown data, businesses can enhance customer satisfaction, address challenges, and achieve overall success.

In [17]:
```
# sentiment breakdown by target
grouped = tweet_df.groupby(['product', 'sentiment']).count().reset_index()
grouped
```

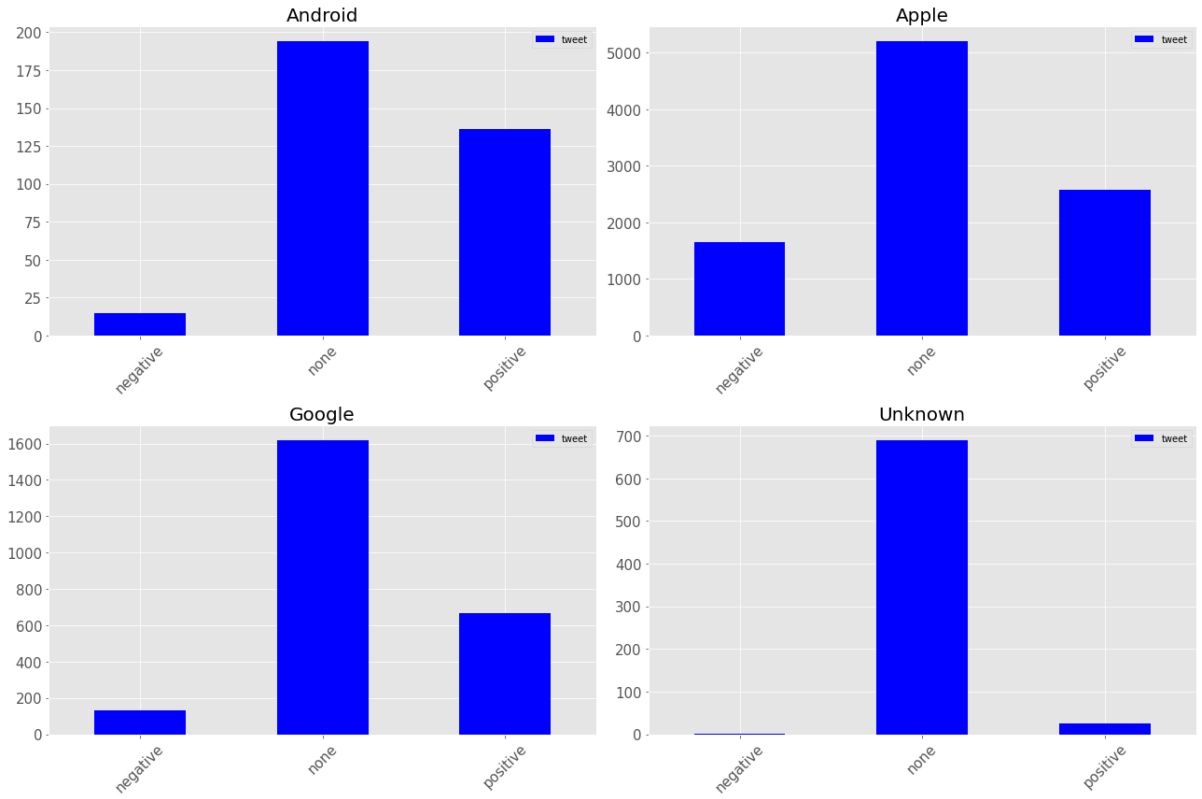
Out[17]:

	product	sentiment	tweet
0	Android	negative	15
1	Android	none	194
2	Android	positive	136
3	Apple	negative	1644
4	Apple	none	5202
5	Apple	positive	2571
6	Google	negative	129
7	Google	none	1620
8	Google	positive	668
9	Unknown	negative	1
10	Unknown	none	690
11	Unknown	positive	26

In [18]:
```
grouped_android = grouped[grouped['product'] == 'Android'][['sentiment','tweet']]
grouped_Apple = grouped[grouped['product'] == 'Apple'][['sentiment','tweet']]
grouped_Google = grouped[grouped['product'] == 'Google'][['sentiment','tweet']]
grouped_Unknown = grouped[grouped['product'] == 'Unknown'][['sentiment','tweet']]
```

Visual of Distribution of Sentiment Breakdown by Product

In [19]:
```
products = [grouped_android, grouped_Apple, grouped_Google, grouped_Unknown]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

axes = axes.flatten()

category_names = ['Android', 'Apple', 'Google', 'Unknown']

xticks = ['negative', 'none', 'positive']

for i, category in enumerate(products):
    ax = axes[i]
    category.plot(kind='bar', ax=ax, color='b')
    ax.set_xlabel('')
    ax.set_ylabel('')
    ax.set_title(category_names[i], fontsize=20)
    ax.tick_params(axis='x', rotation=45, labelsize=15)
    ax.tick_params(axis='y', labelsize=15)
    ax.set_xticklabels(xticks)

plt.tight_layout()
plt.show()
```



Data Preparation

I. Text Cleaning - Word Tokenization

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

The first step of the NLP process is gathering the data (a sentence) and breaking it into understandable parts (words).

Text Before Cleaning:

"~~RT~~ ~~@LaurieShook:~~ I'm look~~ing~~ forward ~~to the~~ #SMCDallas pre ~~#SXSW~~ party Wed~~.,~~ ~~and~~ hop~~ing~~ I'll win ~~an #iPad~~ result~~ing~~ ~~from~~ ~~my~~ shameless promotion~~.~~ #ChevySMC"

Text After Cleaning:

"im look forward smcdallas pre party wed hop ill win result shameless promotion chevysmc"

In [20]:
```
tweet_df['tweet_text_cleaned'] = tweet_df['tweet'].map(txt_clean)
```

In [21]:
```
tweet_df['tweet'][25]
```

Out[21]:
```
25    RT @LaurieShook: I'm looking forward to the #SMCDallas pre #SXSW party Wed., and
hoping I'll win an #iPad resulting from my shameless promotion.  #ChevySMC
25                                                    I had to do made the #switch fr
om iPhone 6 to the galaxy note edge. @apple keep up http://t.co/1Vve1htP0n (http://t.c
o/1Vve1htP0n)
Name: tweet, dtype: object
```

In [22]:
```
tweet_df['tweet_text_cleaned'][25]
```

Out[22]:
```
25    im look forward smcdallas pre party wed hop ill win result shameless promotion ch
evysmc
25                                                      make switch galaxy note edg
e keep
Name: tweet_text_cleaned, dtype: object
```

II. Data Preprocessing - Train Test Split

The train-test split is a process used in machine learning to evaluate model performance. It involves dividing a dataset into two parts: a training set and a testing set. The training set is used to train the model, while the testing set is used to assess its performance on unseen data. The process includes randomizing the dataset, selecting a split ratio, dividing the data into the two subsets, training the model on the training set, and evaluating its performance on the testing set. This approach helps estimate how well the model will generalize to new, unseen data and allows for model optimization before real-world deployment.

Train-test split is typically performed at the beginning of the machine learning process to avoid data leakage. Data leakage refers to a situation where information from the testing set inadvertently influences the training process or model evaluation, leading to overly optimistic performance estimates. It can result in models that generalize poorly to new, unseen data.

Machine Learning models require numeric values, so we will replace them. `Positive` will receive the numeric label `0`, `Negative` will receive the numeric label `1`, and `none` will receive the numeric label `2`.

In [23]:
```
tweet_df['sentiment'] = tweet_df['sentiment'].apply(lambda x: 0 if x=='positive'\
                                                    else 1 if x=='negative'\
                                                    else 2 if x=='none'\
                                                   else x)
```

In [24]:
```
tweet_df.head()
```

Out[24]:

	tweet	product	sentiment	tweet_text_cleaned
0	.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead! I need to upgrade. Plugin stations at #SXSW.	Apple	1	hr tweet riseaustin dead need upgrade plugin station
1	@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate for its design. Also, they're giving free Ts at #SXSW	Apple	0	know awesome ipadiphone app youll likely appreciate design also theyre give free t
2	@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.	Apple	0	wait also sale
3	@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw	Apple	1	hope year festival isnt crashy year app
4	@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/conferences) & Matt Mullenweg (Wordpress)	Google	0	great stuff fri marissa mayer tim oreilly tech booksconferences amp matt mullenweg wordpress

In [25]:
```
tweet_df['sentiment'].value_counts()
```

Out[25]:
```
2    7706
0    3401
1    1789
Name: sentiment, dtype: int64
```

Split data into x and y

In [26]:
```
X = tweet_df['tweet_text_cleaned']
y = tweet_df['sentiment']

# train test split for training set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42, test_size =
# train test split for validation set
X_t, X_val, y_t, y_val = train_test_split(X, y, random_state = 42, test_size = 0.25)
```

Tokenization

Tokenization is a fundamental step in Natural Language Processing (NLP) that involves breaking down text into smaller units called tokens. These tokens can be words, sentences, or even subword units, depending on the level of granularity required for a specific NLP task.

Count Vectorizer

A **CountVectorizer** is a popular text preprocessing technique in natural language processing (NLP). It is used to convert a collection of text documents into a matrix of token (word) counts. In other words, it transforms a set of textual data into a numerical format that machine learning models can understand and process.

Vector Representation: The count vectorizer represents each document as a numeric vector where the value in each position corresponds to the count of the word at that index in the vocabulary. The vector is typically sparse because most documents contain only a subset of the entire vocabulary.

Plotting Unigrams

What is a unigram?

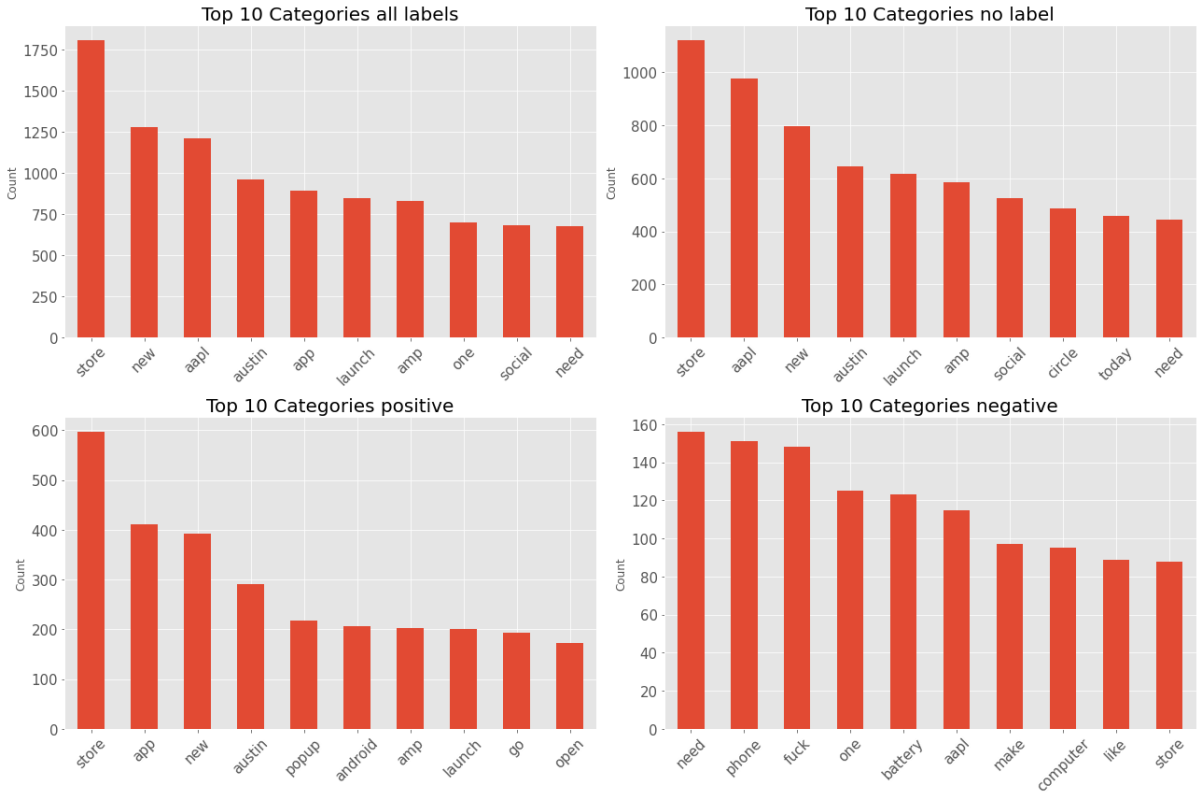
In natural language processing, an n-gram is a sequence of n words. For example, “statistics” is a unigram (n = 1), “machine learning” is a bigram (n = 2), “natural language processing” is a trigram (n = 3). For longer n-grams, people just use their lengths to identify them, such as 4-gram, 5-gram, and so on. In this part of the project, we will focus only on language models based on unigrams i.e. single words.

In [27]:

```
#initialize count vectorizer for unigrams
vectorizer = CountVectorizer(ngram_range = (1,1))
#fit_transform the vectorizer to the data
X_count = vectorizer.fit_transform(X)

X_count = pd.DataFrame.sparse.from_spmatrix(X_count)
X_count.columns = sorted(vectorizer.vocabulary_)
X_count.set_index(y.index, inplace=True)

all_uni_labels = X_count.sum().sort_values(ascending = False)[0:10]
uni_positive = X_count[tweet_df['sentiment'] == 0].sum().sort_values(ascending = False)[
uni_negative = X_count[tweet_df['sentiment'] == 1].sum().sort_values(ascending = False)[
uni_no_label = X_count[tweet_df['sentiment'] == 2].sum().sort_values(ascending = False)[

sentiments_uni = [all_uni_labels, uni_no_label, uni_positive, uni_negative]
```

In [28]:

```
sentiments_uni = [all_uni_labels, uni_no_label, uni_positive, uni_negative]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

axes = axes.flatten()

category_names = ['all labels', 'no label', 'positive', 'negative']


for i, category in enumerate(sentiments_uni):
    ax = axes[i]
    category.plot(kind='bar', ax=ax)
    ax.set_ylabel('Count')
    ax.set_title(f'Top 10 Categories {category_names[i]}', fontsize=20)
    ax.tick_params(axis='x', rotation=45, labelsize=15)
    ax.tick_params(axis='y', labelsize=15)

plt.tight_layout()
plt.show()
```



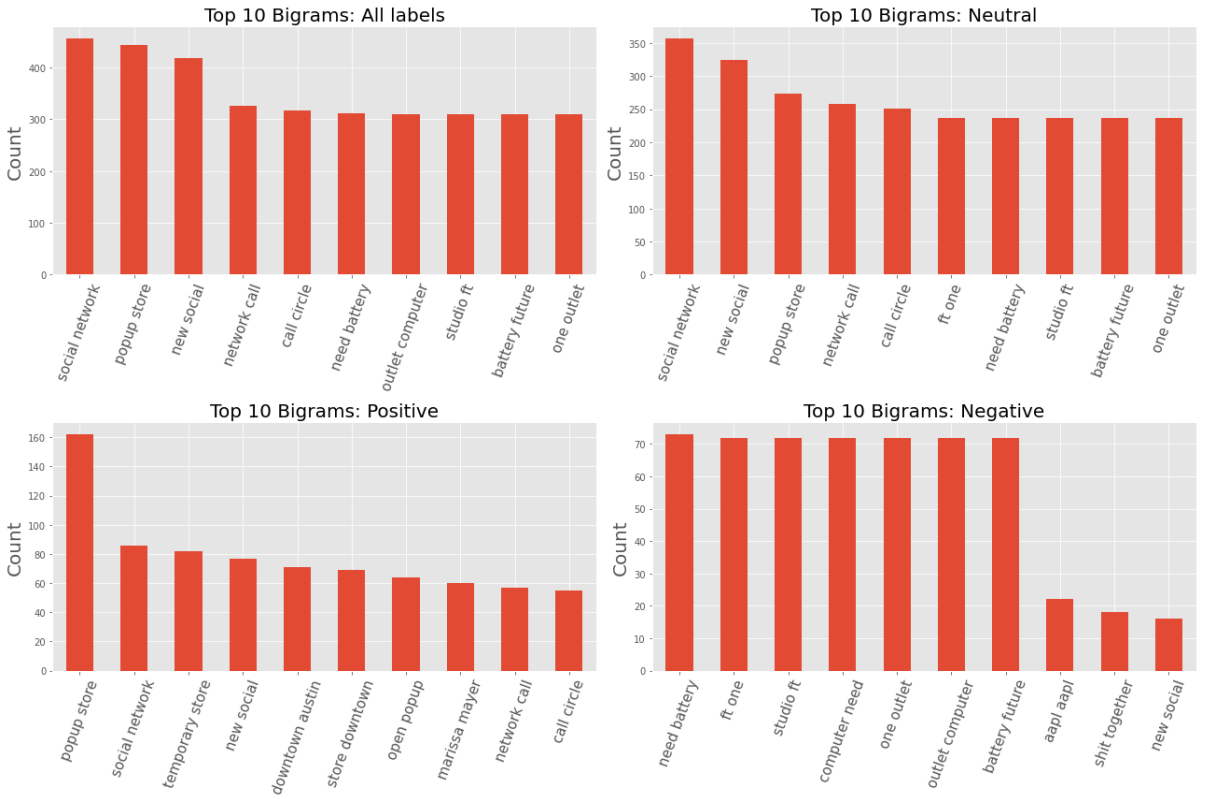
Plotting Bigrams

What is a bigram?

A bigram is an n-gram where "n" is equal to 2, meaning it consists of two consecutive words in a given text. When processing a sentence or a document, bigrams capture adjacent word pairs, allowing for a more fine-grained analysis of language patterns and relationships between words.

In [29]:
```
cv = CountVectorizer(ngram_range = (2,2))
X_count = cv.fit_transform(X)
X_count  = pd.DataFrame.sparse.from_spmatrix(X_count)
X_count.columns = sorted(cv.vocabulary_)
X_count.set_index(y.index, inplace=True)

all_bi_labels = X_count.sum().sort_values(ascending = False)[0:10]
bi_positive = X_count[tweet_df['sentiment'] == 0].sum().sort_values(ascending = False)[0
bi_negative = X_count[tweet_df['sentiment'] == 1].sum().sort_values(ascending = False)[0
bi_no_label = X_count[tweet_df['sentiment'] == 2].sum().sort_values(ascending = False)[0
```

In [80]:
```
sentiments_bi = [all_bi_labels, bi_no_label, bi_positive, bi_negative]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

axes = axes.flatten()

category_names = ['all labels', 'neutral', 'positive', 'negative']


for i, category in enumerate(sentiments_bi):
    ax = axes[i]
    category.plot(kind='bar', ax=ax)
    ax.set_ylabel('Count', fontsize=20)
    ax.set_title(f'Top 10 Bigrams: {category_names[i].capitalize()}', fontsize=20)
    ax.tick_params(axis='x', rotation=70, labelsize=15)

plt.tight_layout()
plt.show()
```



What is a trigram?

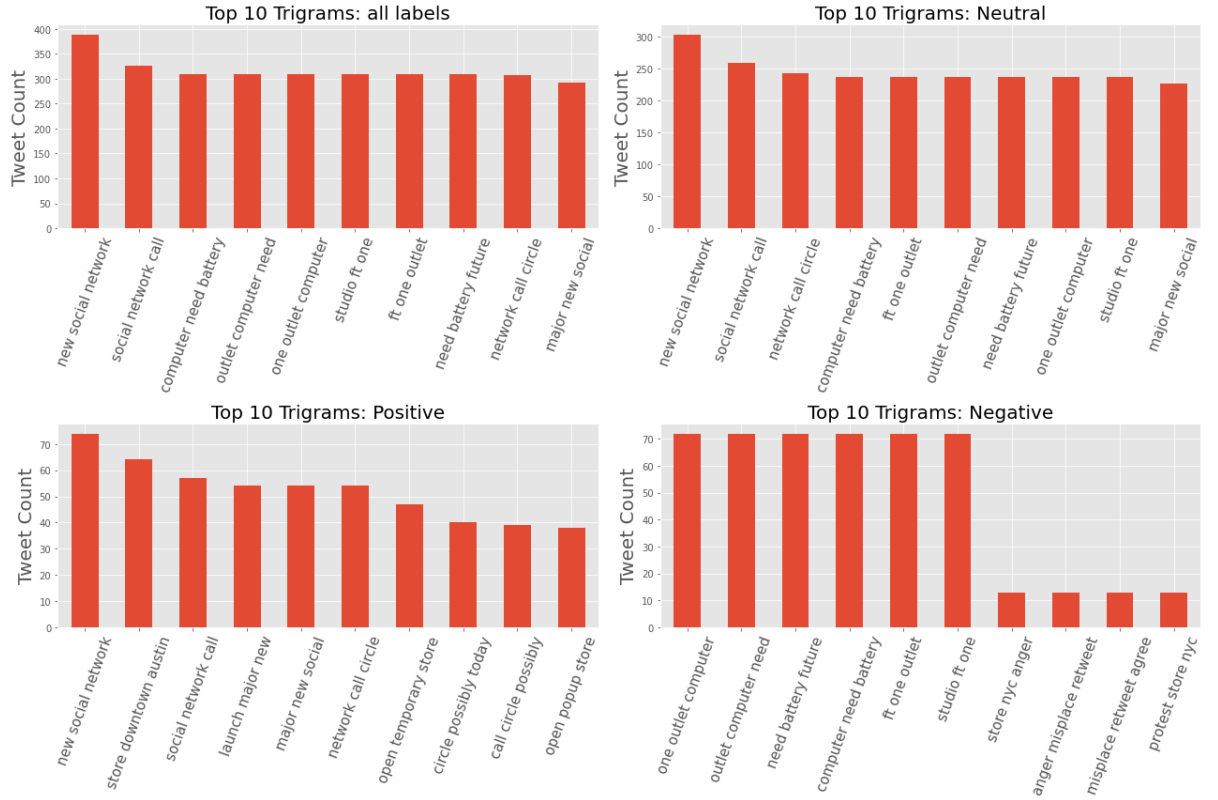
You probably get the idea by now.

In [31]:

```
cv = CountVectorizer(ngram_range = (3,3))
X_count = cv.fit_transform(X)
X_count  = pd.DataFrame.sparse.from_spmatrix(X_count)
X_count.columns = sorted(cv.vocabulary_)
X_count.set_index(y.index, inplace=True)

all_tri_labels = X_count.sum().sort_values(ascending = False)[0:10]
tri_positive = X_count[tweet_df['sentiment'] == 0].sum().sort_values(ascending = False)[
tri_negative = X_count[tweet_df['sentiment'] == 1].sum().sort_values(ascending = False)[
tri_no_label = X_count[tweet_df['sentiment'] == 2].sum().sort_values(ascending = False)[
```

In [78]:

```
sentiments_bi = [all_tri_labels, tri_no_label, tri_positive, tri_negative]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

axes = axes.flatten()

category_names = ['all labels', 'Neutral', 'Positive', 'Negative']


for i, category in enumerate(sentiments_bi):
    ax = axes[i]
    category.plot(kind='bar', ax=ax)
    ax.set_ylabel('Tweet Count', fontsize=20)
    ax.set_title(f'Top 10 Trigrams: {category_names[i]}', fontsize=20)
    ax.tick_params(axis='x', rotation=70, labelsize=15)

plt.tight_layout()
plt.show()
```



Count Vectorizer - Unigrams

In [33]:
```
cv = CountVectorizer(ngram_range = (1,1))
X_t_vec = cv.fit_transform(X_t)
X_t_vec  = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(cv.vocabulary_)
X_t_vec.set_index(y_t.index, inplace=True)

X_val_vec = cv.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(cv.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.sum(axis = 0).sort_values(ascending = False)[:16]
```

Out[33]:
```
store      1328
new         982
aapl        910
austin      722
app         669
launch      650
amp         634
one         535
social      530
need        506
circle      469
today       469
android     454
popup       449
go          441
via         425
dtype: int64
```

Count Vectorizer - Bigrams

In [34]:
```
cv = CountVectorizer(ngram_range = (2,2))
X_t_vec = cv.fit_transform(X_t)
X_t_vec  = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(cv.vocabulary_)
X_t_vec.set_index(y_t.index, inplace=True)

X_val_vec = cv.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(cv.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.sum(axis = 0).sort_values(ascending = False)[:16]
```

Out[34]:
```
social network     355
popup store        332
new social         322
network call       251
call circle        240
one outlet         234
battery future     234
computer need      234
outlet computer    234
ft one             234
need battery       234
studio ft          234
major new          218
launch major       213
aapl aapl          206
possibly today     178
dtype: int64
```

Count Vectorizer - Trigrams

In [35]:
```
cv = CountVectorizer(ngram_range = (3,3))
X_t_vec = cv.fit_transform(X_t)
X_t_vec  = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(cv.vocabulary_)
X_t_vec.set_index(y_t.index, inplace=True)

X_val_vec = cv.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(cv.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.sum(axis = 0).sort_values(ascending = False)[:16]
```

Out[35]:
```
new social network        302
social network call       251
network call circle       235
studio ft one             234
ft one outlet             234
outlet computer need      234
computer need battery     234
one outlet computer       234
need battery future       234
major new social          217
launch major new          213
call circle possibly      174
circle possibly today     172
anger misplace retweet    114
misplace retweet agree    114
store nyc anger           114
dtype: int64
```

TFIDF Vectorizer

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used text preprocessing technique in natural language processing (NLP). It is used to convert a collection of text documents into numerical vectors while taking into account the importance of words in each document and their significance in the entire corpus. TF-IDF is commonly used for information retrieval, text mining, and text classification tasks.

Term Frequency (TF) Calculation: For each document in the text collection, the term frequency is computed for each word. The term frequency of a word in a document is the number of times that word appears in that document. It is calculated as follows: TF(word, document) = (Number of occurrences of 'word' in 'document') / (Total number of words in 'document')

Inverse Document Frequency (IDF) Calculation: IDF measures the importance of a word in the entire corpus by penalizing words that occur frequently across all documents. IDF is calculated for each word as follows: IDF(word) = log((Total number of documents) / (Number of documents containing 'word')) + 1

The "+ 1" is added to prevent division by zero in case a word appears in all documents.

TF-IDF Calculation: Finally, the TF-IDF score for each word in each document is computed by multiplying the TF value with the IDF value for that word: TF-IDF(word, document) = TF(word, document) * IDF(word)

Vector Representation: The TF-IDF vectorizer represents each document as a numeric vector where each position corresponds to a word in the vocabulary. The value in each position is the TF-IDF score of the corresponding word in that document. The resulting vectors are typically sparse since most documents contain only a subset of the entire vocabulary.

In [36]:
```
tfidf = TfidfVectorizer(ngram_range = (1,1))
X_t_vec = tfidf.fit_transform(X_t)
X_t_vec = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(tfidf.vocabulary_)
X_t_vec.set_index(y_t.index, inplace = True)

X_val_vec = tfidf.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(tfidf.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.sum(axis = 0).sort_values(ascending = False)[:16]
```

Out[36]:
```
store      271.243608
new        195.041960
aapl       183.777520
austin     165.821743
launch     163.231807
app        140.326152
one        135.052512
social     134.587046
need       132.549141
circle     131.014023
popup      126.179037
amp        120.857879
today      117.851592
open       116.978654
network    109.291619
via        109.100597
dtype: float64
```

In [37]:
```
tfidf = TfidfVectorizer(ngram_range = (2,2))
X_t_vec = tfidf.fit_transform(X_t)
X_t_vec = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(tfidf.vocabulary_)
X_t_vec.set_index(y_t.index, inplace = True)

X_val_vec = tfidf.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(tfidf.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.mean(axis = 0).sort_values(ascending = False)[:16]
```

Out[37]:
```
social network     0.009277
computer need      0.009144
one outlet         0.009144
battery future     0.009144
outlet computer    0.009144
need battery       0.009144
ft one             0.009144
studio ft          0.009144
new social         0.008646
popup store        0.008153
network call       0.007351
call circle        0.007077
major new          0.006561
launch major       0.006478
possibly today     0.005620
circle possibly    0.005596
dtype: float64
```

In [38]:
```
tfidf = TfidfVectorizer(ngram_range = (3,3))
X_t_vec = tfidf.fit_transform(X_t)
X_t_vec = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(tfidf.vocabulary_)
X_t_vec.set_index(y_t.index, inplace = True)

X_val_vec = tfidf.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(tfidf.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)

X_t_vec.sum(axis = 0).sort_values(ascending = False)[:16]
```

Out[38]:
```
outlet computer need      95.530100
studio ft one             95.530100
need battery future       95.530100
ft one outlet             95.530100
computer need battery     95.530100
one outlet computer       95.530100
new social network        82.778818
social network call       73.247177
network call circle       69.777197
major new social          65.285982
launch major new          64.646487
call circle possibly      55.283790
circle possibly today     54.988682
anger misplace retweet    40.193342
store nyc anger           40.193342
misplace retweet agree    40.193342
dtype: float64
```

Word Cloud Data Prep

In this section of the notebook, text is prepped and stored in the directory to be sent to another notebook in the repository to create a visual for the presentation. This part of the notebook does not serve in the actual exploratory data analysis, data preparation, or modeling of the NLP process.

In [39]:
```
tweet_df['tweet_text_cleaned']
```

Out[39]:
```
0                                          hr tweet riseaustin dead need upgrade p
lugin station
1                know awesome ipadiphone app youll likely appreciate design also theyr
e give free t
2                                                                                  w
ait also sale
3                                                        hope year festival isnt cr
ashy year app
4       great stuff fri marissa mayer tim oreilly tech booksconferences amp matt mullen
weg wordpress
                                                    ...
3881                              via fc warm social medium hire social medium guru l
a finally twe
3882                                                                       avocado
emoji may ask
3883                                       could agree great thing happen appleandibm
ibmandtwitter
3884                          photo longer download automatically laptop sync support u
nhelpful idea
3885                                                        excite name app store b
est list year
Name: tweet_text_cleaned, Length: 12896, dtype: object
```

Writing all cleaned text as one string

In [40]:
```
tweet_df_pos = tweet_df[tweet_df['sentiment'] == 0]
tweet_df_neg = tweet_df[tweet_df['sentiment'] == 1]
tweet_df_neu = tweet_df[tweet_df['sentiment'] == 2]
```

Saving as txt file

In [41]:
```
save_text_file(tweet_df, 'all_tweet_text')
```

```
Text data saved to 'C:\Users\alevi\Documents\Flatiron\dsc-data-science-env-config\Cours
e_Folder\Phase_4\Phase_4_Project_NLP-Twitter-Sentiment\all_tweet_text.txt'.
```

In [42]:
```
save_text_file(tweet_df_pos, 'positive_tweet_text')
```

```
Text data saved to 'C:\Users\alevi\Documents\Flatiron\dsc-data-science-env-config\Cours
e_Folder\Phase_4\Phase_4_Project_NLP-Twitter-Sentiment\positive_tweet_text.txt'.
```

In [43]:
```
save_text_file(tweet_df_neg, 'negative_tweet_text')
```

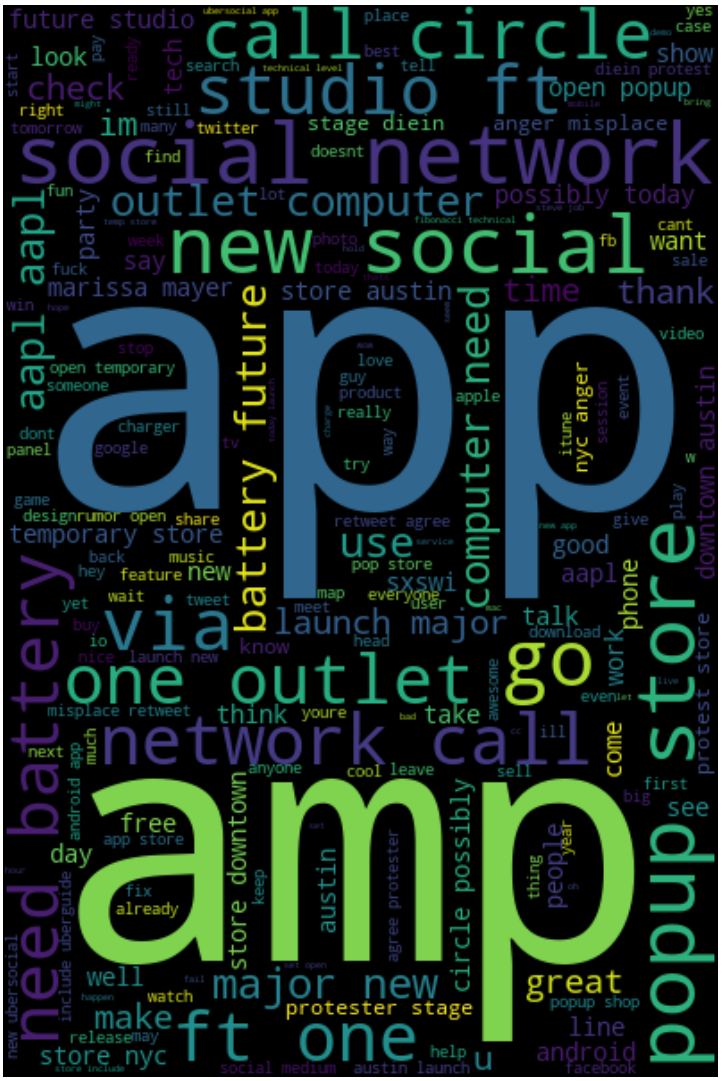
```
Text data saved to 'C:\Users\alevi\Documents\Flatiron\dsc-data-science-env-config\Cours
e_Folder\Phase_4\Phase_4_Project_NLP-Twitter-Sentiment\negative_tweet_text.txt'.
```

In [44]:
```
save_text_file(tweet_df_neu, 'neutral_tweet_text')
```

```
Text data saved to 'C:\Users\alevi\Documents\Flatiron\dsc-data-science-env-config\Cours
e_Folder\Phase_4\Phase_4_Project_NLP-Twitter-Sentiment\neutral_tweet_text.txt'.
```

In [45]:

```
from pathlib import Path
import matplotlib.pyplot as plt
from wordcloud import STOPWORDS, WordCloud, ImageColorGenerator
from IPython.display import display, Image
import numpy as np
import colorsys


TXT_FILE = Path.cwd() / "all_tweet_text.txt"

# Read text
text = open(TXT_FILE, mode="r", encoding="utf-8").read()
stopwords = STOPWORDS

wc = WordCloud(background_color="black", stopwords=stopwords, height=600, width=400)
wc.generate(text)


output_path = Path.cwd() / "wordcloud_output.png"
wc.to_file(output_path)


display(Image(output_path))
```



In [46]:
```
from PIL import Image
```

In [47]:

```
TXT_FILE = Path.cwd() / "all_tweet_text.txt"
MASK_FILE = Path.cwd() / "images/twitter_mask.png"  # Provide the path to your custom mas

# Read text
text = open(TXT_FILE, mode="r", encoding="utf-8").read()
stopwords = STOPWORDS

# Load the custom mask image
tweet_mask = np.array(Image.open(MASK_FILE))

# Transform the mask image
def transform_format(val):
    if val == 0:
        return 255
    else:
        return val

transformed_tweet_mask = np.ndarray((tweet_mask.shape[0], tweet_mask.shape[1]), np.int32
for i in range(len(tweet_mask)):
    transformed_tweet_mask[i] = list(map(transform_format, tweet_mask[i]))


wc = WordCloud(background_color="white", mode='RGBA', max_font_size=50, max_words=500, s
               mask=transformed_tweet_mask, width=800, height=400, scale=5)
wc.generate(text)


def get_shade_of_blue(word, font_size, position, orientation, random_state=None, **kwargs

    hue = 240
    saturation = 100
    lightness = random_state.randint(30, 70)

    # Convert HSL to RGB
    rgb = colorsys.hls_to_rgb(hue/360, lightness/100, saturation/100)
    return f"rgb({int(rgb[0]*255)}, {int(rgb[1]*255)}, {int(rgb[2]*255)})"



wc.recolor(color_func=get_shade_of_blue, random_state=np.random.RandomState(42))


output_path = Path.cwd() / "WordCloud_all.png"
wc.to_file(output_path)

# Display the WordCloud
plt.figure(figsize=(80, 50))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```


social network
popup store
new social
network call
call circle
amp
app
go
via
need battery
studio ft
ft one
one outlet
outlet computer
computer need
battery future
launch major
aapl aapl
temporary store
circle possibly
possibly today
store austin
stage diein
protester stage
store downtown
anger misplace
protest store
nyc anger
marissa mayer
future studio
check
thank
great
time
make
major
new
im
use
say
see
good
people
sxswi
store nyc
android
phone
work
misplace retweet
launch new
pop store
free
austin
party
talk
show
come
tech
know
google
aapl
today
app store
look
take
apple
think
line
day
want
well
downtown
austin
open popup
social medium
open temporary
retweet agree
agree protester
austin launch
diein protest

In [48]:

```
TXT_FILE = Path.cwd() / "neutral_tweet_text.txt"
MASK_FILE = Path.cwd() / "images/twitter_mask.png"  # Provide the path to your custom mas

# Read text
text = open(TXT_FILE, mode="r", encoding="utf-8").read()
stopwords = STOPWORDS

# Load the custom mask image
tweet_mask = np.array(Image.open(MASK_FILE))

# Transform the mask image
def transform_format(val):
    if val == 0:
        return 255
    else:
        return val

transformed_tweet_mask = np.ndarray((tweet_mask.shape[0], tweet_mask.shape[1]), np.int32
for i in range(len(tweet_mask)):
    transformed_tweet_mask[i] = list(map(transform_format, tweet_mask[i]))


wc = WordCloud(background_color="white", mode='RGBA', max_font_size=50, max_words=500, s
               mask=transformed_tweet_mask, width=800, height=400, scale=5)
wc.generate(text)


def get_shade_of_blue(word, font_size, position, orientation, random_state=None, **kwargs

    hue = 240
    saturation = 100
    lightness = random_state.randint(30, 70)

    # Convert HSL to RGB
    rgb = colorsys.hls_to_rgb(hue/360, lightness/100, saturation/100)
    return f"rgb({int(rgb[0]*255)}, {int(rgb[1]*255)}, {int(rgb[2]*255)})"



wc.recolor(color_func=get_shade_of_blue, random_state=np.random.RandomState(42))


output_path = Path.cwd() / "WordCloud_neutral.png"
wc.to_file(output_path)

# Display the WordCloud
plt.figure(figsize=(80, 50))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```


social network
popup store
new social
app
amp
network call
call circle
studio ft
via
ft one
one outlet
go
outlet computer
computer need
need battery
battery future
major new
launch major
circle possibly
possibly today
austin
android
store downtown
protest store
temporary store
anger misplace
marissa mayer
store nyc
nyc anger
people
misplace retweet
protester stage
pop store
downtown austin
store austin
stage diein
future studio
retweet agree
open temporary
open popup
sxswi
aapl
agree protester
free
google
panel
time
line
want
day
take
think
make
session
tweet
work
see
check
win
look
good
thank
find
im
year
social medium
launch new
today launch
fibonacci technical
come
party
apple
watch
talk
tech
show
hey
facebook
well
great
week
case
diein protest
popup shop
next
map
first
still
itune
know
head
help
video
use
guy
release
say
back
share

In [49]:

```
TXT_FILE = Path.cwd() / "positive_tweet_text.txt"
MASK_FILE = Path.cwd() / "images/twitter_mask.png"  # Provide the path to your custom mas

# Read text
text = open(TXT_FILE, mode="r", encoding="utf-8").read()
stopwords = STOPWORDS

# Load the custom mask image
tweet_mask = np.array(Image.open(MASK_FILE))

# Transform the mask image
def transform_format(val):
    if val == 0:
        return 255
    else:
        return val

transformed_tweet_mask = np.ndarray((tweet_mask.shape[0], tweet_mask.shape[1]), np.int32
for i in range(len(tweet_mask)):
    transformed_tweet_mask[i] = list(map(transform_format, tweet_mask[i]))

# Create WordCloud with the transformed mask
wc = WordCloud(background_color="white", mode='RGBA', max_font_size=50, max_words=500, s
               mask=transformed_tweet_mask, width=800, height=400, scale=5)
wc.generate(text)

# Define the color function to create shades of blue
def get_shade_of_blue(word, font_size, position, orientation, random_state=None, **kwarg

    hue = 240
    saturation = 100
    lightness = random_state.randint(30, 70)


    rgb = colorsys.hls_to_rgb(hue/360, lightness/100, saturation/100)
    return f"rgb({int(rgb[0]*255)}, {int(rgb[1]*255)}, {int(rgb[2]*255)})"


wc.recolor(color_func=get_shade_of_blue, random_state=np.random.RandomState(42))


output_path = Path.cwd() / "WordCloud_positive.png"
wc.to_file(output_path)

# Display the WordCloud
plt.figure(figsize=(80, 50))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```


app
popup store
great
thank
new
android
party
austin
via
amp
launch
check
awesome
one
see
love
sxswi
im
go
good
time
free
use
make
map

In [50]:

```
TXT_FILE = Path.cwd() / "negative_tweet_text.txt"
MASK_FILE = Path.cwd() / "images/twitter_mask.png"  # Provide the path to your custom mas

# Read text
text = open(TXT_FILE, mode="r", encoding="utf-8").read()
stopwords = STOPWORDS

# Load the custom mask image
tweet_mask = np.array(Image.open(MASK_FILE))

# Transform the mask image
def transform_format(val):
    if val == 0:
        return 255
    else:
        return val

transformed_tweet_mask = np.ndarray((tweet_mask.shape[0], tweet_mask.shape[1]), np.int32
for i in range(len(tweet_mask)):
    transformed_tweet_mask[i] = list(map(transform_format, tweet_mask[i]))

# Create WordCloud with the transformed mask
wc = WordCloud(background_color="white", mode='RGBA', max_font_size=50, max_words=500, s
               mask=transformed_tweet_mask,  width=800, height=400, scale=5)
wc.generate(text)

# Define the color function to create shades of blue
def get_shade_of_blue(word, font_size, position, orientation, random_state=None, **kwarg
    # Generate shades of blue using the HSL color space
    hue = 240  # Hue value for blue (240 degrees)
    saturation = 100  # Constant value for saturation (full saturation)
    lightness = random_state.randint(30, 70)  # Vary lightness between 30 and 70 (0 is d

    # Convert HSL to RGB
    rgb = colorsys.hls_to_rgb(hue/360, lightness/100, saturation/100)
    return f"rgb({int(rgb[0]*255)}, {int(rgb[1]*255)}, {int(rgb[2]*255)})"

# Recolor the WordCloud using the custom color function
# Pass an instance of np.random.RandomState as random_state
wc.recolor(color_func=get_shade_of_blue, random_state=np.random.RandomState(42))

# Save the WordCloud as a file
output_path = Path.cwd() / "WordCloud_negative.png"
wc.to_file(output_path)

# Display the WordCloud
plt.figure(figsize=(80, 50))  # Increase the figure size for better resolution
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```

IV. Modeling

Baseline: Shallow Multinomial Naive Bayes

A Multinomial Naive Bayes model is a probabilistic classification algorithm commonly used in natural language processing (NLP) tasks. It is an extension of the Naive Bayes algorithm, which assumes that features (words in NLP) are conditionally independent given the class label.

Explanation of Multinomial Naive Bayes:

- `Probability Model` : In Multinomial Naive Bayes, we model the probability of observing a document (sequence of words) given a class label. The document is represented as a bag-of-words, which means the order of words is disregarded, and only the frequency of each word matters.
- `Feature Representation` : In NLP tasks, we often use a bag-of-words representation, where each document is represented as a vector of word counts. The elements of this vector are the frequencies of each word in the document.

- `Conditional Independence` : The "naive" assumption in Naive Bayes is that the features (word counts) are conditionally independent given the class label. In NLP, this implies that the occurrence of one word does not affect the occurrence of other words in the same document, given the class label.
- `Parameter Estimation` : To build a Multinomial Naive Bayes model, we estimate the probabilities of observing each word in the vocabulary given a specific class. These probabilities are learned from the training data, and they represent the likelihood of seeing each word in documents of a particular class.

Justification for using Multinomial Naive Bayes in NLP:

- `Efficiency and Scalability` : Multinomial Naive Bayes is computationally efficient and scales well to large datasets and high-dimensional feature spaces. In NLP, where the vocabulary can be extensive, this efficiency is beneficial.
- `Good Performance with Sparse Data` : NLP data often results in a sparse feature representation because most documents contain only a small fraction of the entire vocabulary. Multinomial Naive Bayes handles sparse data effectively and can still make reasonably accurate predictions.
- `Text Classification` : In many NLP tasks, like sentiment analysis, spam detection, and document categorization, Multinomial Naive Bayes has been shown to perform well and achieve competitive results.
- `Handling Multiple Classes` : Multinomial Naive Bayes can handle more than two classes (multiclass classification) with minimal additional computational cost. This makes it suitable for tasks involving multiple categories or labels.
- `Reasonable Assumption for NLP` : The independence assumption, though simplistic, often works surprisingly well in NLP tasks. While words in natural language are not entirely independent, the conditional independence assumption captures some level of correlation between words and allows for a straightforward

Confusion matrix evaluation for False Negatives - Recall

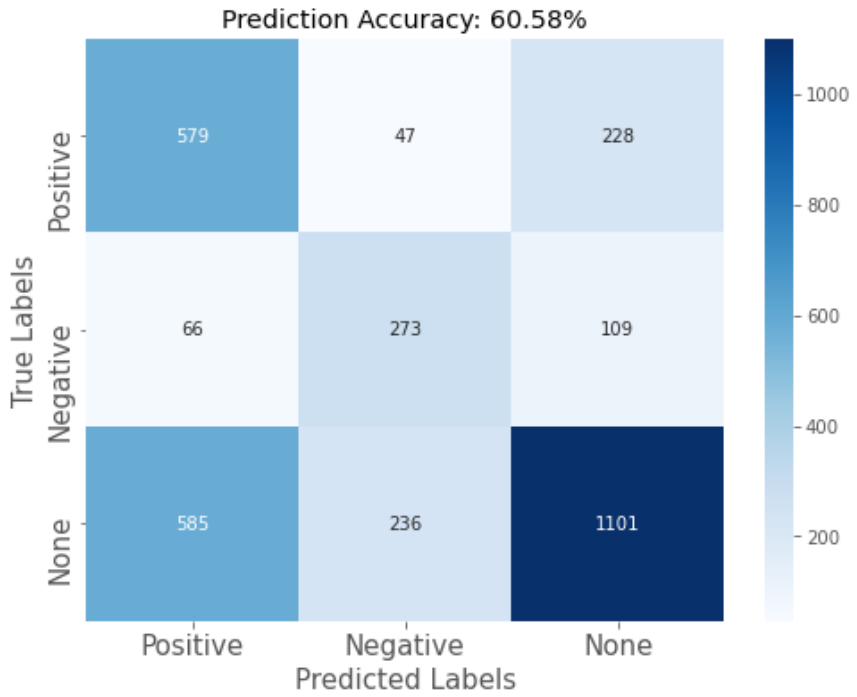
- **Recall**, also known as sensitivity or true positive rate, measures the proportion of actual positive cases correctly identified by the model. It focuses on minimizing false negatives, which means it aims to avoid classifying positive cases as negative.
- By emphasizing recall in the context of tweet sentiment analysis, the model aims to correctly identify as many positive, negative, or neutral tweets as possible, minimizing the chances of missing important cases. This enables the business to take appropriate actions, such as adjusting their business plan to minimize negative tweets and maximize positive tweets.

MNB: Count Vectorizer

In [51]:
```
mnb = MultinomialNB()

mnb_pipe_count = make_pipeline(CountVectorizer(),\
                           SMOTE(),\
                           mnb)

mnb_pipe_count.fit(X_t, y_t)
y_hat = mnb_pipe_count.predict(X_val)
accuracy_score(y_val, y_hat)
```

Out[51]:
```
0.6057692307692307
```

In [52]:
```
plot_cm_senti(mnb_pipe_count, y_val, y_hat), classify(y_val, y_hat)
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.68      0.56       854
           1       0.49      0.61      0.54       448
           2       0.77      0.57      0.66      1922

    accuracy                           0.61      3224
   macro avg       0.58      0.62      0.58      3224
weighted avg       0.65      0.61      0.61      3224
```

Out[52]: (None, None)

Observations

Class "Positive":

- Precision: 0.47 - The model correctly identifies positive tweets 47% of the time among all predicted positive tweets.
- Recall: 0.67 - The model captures only 67% of actual positive tweets among all actual positive tweets.

Class "Negative":

- Precision: 0.50 - The model correctly identifies negative tweets 50% of the time among all predicted negative tweets.
- Recall: 0.62 - The model captures only 62% of actual negative tweets among all actual negative tweets.

Class "No Emotion":

- Precision: 0.77 - The model correctly identifies tweets with no emotion 77% of the time among all predicted no emotion tweets.
- Recall: 0.58 - The model captures 58% of actual no emotion tweets among all actual no emotion tweets.

- Accuracy:
- Overall accuracy: 0.61 - The model's predictions are correct for 61% of all tweets.

Analysis:

The model performs reasonably well for the "No Emotion" class, with a high recall, precision, and F1-Score. This indicates that the model is effective at identifying tweets with no discernible sentiment. However, the model struggles with the "Positive" and "Negative" classes, as indicated by low recall. This suggests that the model has difficulty distinguishing between these sentiment classes.

Iteration 2: XGBoost Pipeline

XGBoost (eXtreme Gradient Boosting) is a powerful and widely used machine learning algorithm designed for supervised learning tasks, including both classification and regression problems. It is an implementation of the gradient boosting framework that has gained popularity due to its exceptional performance and scalability.

The purpose of XGBoost is to create an accurate and robust predictive model by combining multiple weak predictive models, typically decision trees. Here are some key purposes and advantages of using XGBoost:

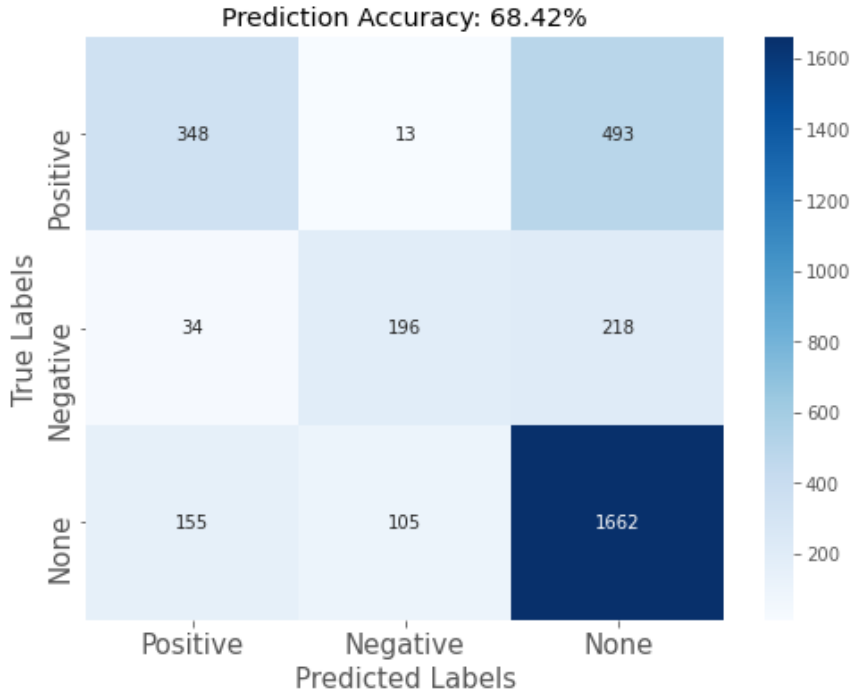
- `High Prediction Accuracy` : XGBoost is known for its exceptional accuracy and predictive power. It leverages gradient boosting techniques to iteratively train a series of weak learners (decision trees) that sequentially correct the mistakes made by the previous models. This iterative process allows XGBoost to capture complex relationships within the data, resulting in improved predictive performance.
- `Handling Complex Data Patterns` : XGBoost can effectively handle complex data patterns, including non-linear relationships and interactions between features. The algorithm can automatically capture and model these intricate relationships through the ensemble of decision trees, making it a suitable choice for a wide range of machine learning problems.
- `Regularization and Control over Model Complexity` : XGBoost provides various regularization techniques to control the complexity of the model and prevent overfitting. Regularization methods, such as L1 and L2 regularization, can be applied to the model's weights or the structure of the decision trees. This helps prevent the model from becoming too complex and provides a way to balance between overfitting and underfitting.
- `Feature Importance Analysis` : XGBoost offers built-in methods to assess the importance of features in the predictive model. By examining the contribution of each feature in the ensemble of decision trees, you can gain insights into the most influential features for making predictions. This analysis aids in feature selection, understanding the underlying data, and interpreting the model's behavior.
- `Scalability and Efficiency` : XGBoost is designed to be highly scalable and efficient, enabling it to handle large datasets and perform computations in parallel. The algorithm includes optimizations such as parallel tree construction, approximate algorithms for split finding, and efficient memory usage. These features make XGBoost suitable for both small and large-scale machine learning tasks.
- `Flexibility and Customization` : XGBoost offers a wide range of hyperparameters that can be tuned to optimize the model's performance for specific tasks. You can adjust parameters related to the tree structure, learning rate, regularization, and more. This flexibility allows you to customize the algorithm to suit your specific needs and achieve the best results.

XGB 1: Count Vectorizer

In [53]:
```
xgb_pipe = XGBClassifier(random_state=42,max_depth = 15, tree_method='hist', n_estimator

xgb_pipe_count = make_pipeline(CountVectorizer(),\
                          SMOTE(),\
                          xgb_pipe)

xgb_pipe_count.fit(X_t, y_t)
y_hat = xgb_pipe_count.predict(X_val)
accuracy_score(y_val, y_hat)
```

Out[53]: 0.68424317617866

In [54]:
```
plot_cm_senti(xgb_pipe_count, y_val, y_hat), classify(y_val, y_hat)
```



Classification Report:

	precision	recall	f1-score	support
0	0.65	0.41	0.50	854
1	0.62	0.44	0.51	448
2	0.70	0.86	0.77	1922
accuracy			0.68	3224
macro avg	0.66	0.57	0.60	3224
weighted avg	0.68	0.68	0.67	3224

Out[54]: (None, None)

Iteration 3: Grid Search with Cross Validation

Using GridSearchCV

- `GridSearchCV` is a class in scikit-learn that performs an exhaustive search over a specified parameter grid to find the best hyperparameters for a given machine learning model. It is a technique for hyperparameter tuning, which involves finding the optimal combination of hyperparameters that maximizes the model's performance.
- `Cross-validation:` GridSearchCV performs cross-validation, which is a technique for evaluating the model's performance on multiple subsets of the training data. It splits the training data into multiple folds, trains the model on a subset of the folds, and evaluates its performance on the remaining fold. This process is repeated for each fold, and the results are averaged to get an overall performance estimate.

In [55]:
```
nb_tfidf_pipe = make_pipeline(TfidfVectorizer(),
                              SMOTE(random_state=42),
                              MultinomialNB()
)
nb_tfidf_pipe
```

Out[55]:
```
Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer()),
                ('smote', SMOTE(random_state=42)),
                ('multinomialnb', MultinomialNB())])
```

In [56]:
```
tfidf_grid = {
    'tfidfvectorizer__min_df': [1, 2],
    'tfidfvectorizer__max_df': [0.01, 0.05, 0.25],
    'tfidfvectorizer__ngram_range': [(1,1), (1,2), (2,2), (2,3), (3,3)],
    'tfidfvectorizer__norm': ('l1', 'l2'),
    'multinomialnb__alpha': [0.1, 0.5, 1.0, 2.0]
}
```

In [57]:
```
gs_nb_tfidf = GridSearchCV(nb_tfidf_pipe, tfidf_grid, cv=3)
```

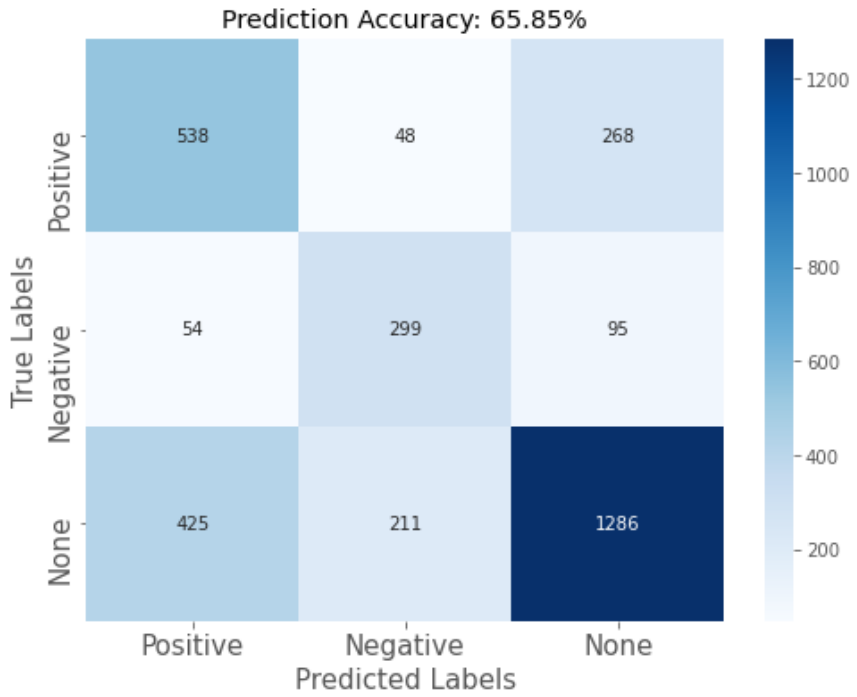
In [58]:
```
gs_nb_tfidf.fit(X_train, y_train)
```

Out[58]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer()),
                                       ('smote', SMOTE(random_state=42)),
                                       ('multinomialnb', MultinomialNB())]),
             param_grid={'multinomialnb__alpha': [0.1, 0.5, 1.0, 2.0],
                         'tfidfvectorizer__max_df': [0.01, 0.05, 0.25],
                         'tfidfvectorizer__min_df': [1, 2],
                         'tfidfvectorizer__ngram_range': [(1, 1), (1, 2),
                                                          (2, 2), (2, 3),
                                                          (3, 3)],
                         'tfidfvectorizer__norm': ('l1', 'l2')})
```

In [59]:
```
y_pred_nb_tfidf_train = gs_nb_tfidf.predict(X_test)
accuracy_score(y_test, y_pred_nb_tfidf_train)
```

Out[59]:
```
0.658498759305211
```

In [60]:
```
plot_cm_senti(gs_nb_tfidf, y_val, y_pred_nb_tfidf_train), classify(y_val, y_pred_nb_tfid
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.63      0.58       854
           1       0.54      0.67      0.59       448
           2       0.78      0.67      0.72      1922

    accuracy                           0.66      3224
   macro avg       0.61      0.66      0.63      3224
weighted avg       0.68      0.66      0.66      3224
```

Out[60]:
```
(None, None)
```

Testing Model on validation set

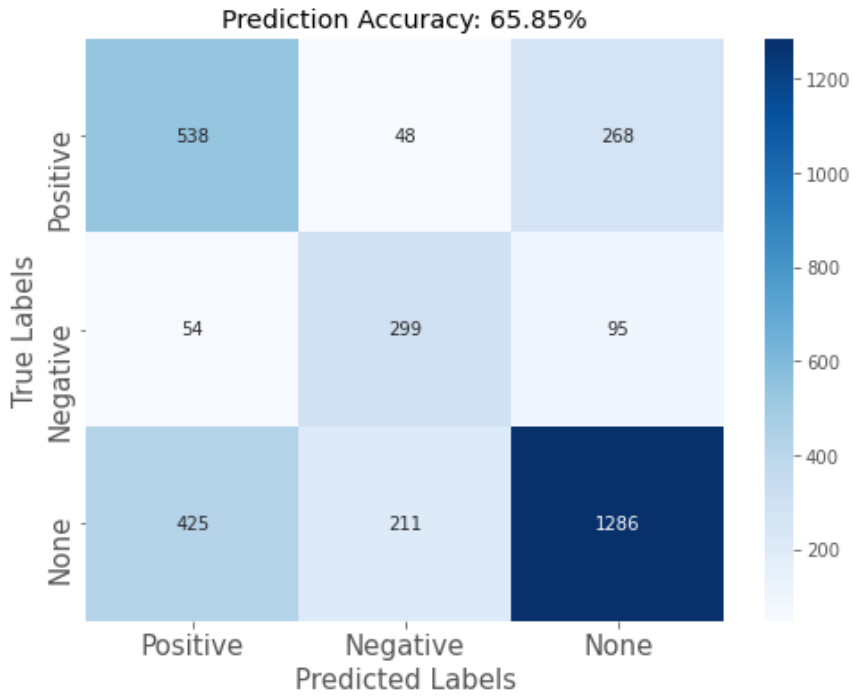
In [61]:
```
gs_nb_tfidf.fit(X_t, y_t)
```

Out[61]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer()),
                                       ('smote', SMOTE(random_state=42)),
                                       ('multinomialnb', MultinomialNB())]),
             param_grid={'multinomialnb__alpha': [0.1, 0.5, 1.0, 2.0],
                         'tfidfvectorizer__max_df': [0.01, 0.05, 0.25],
                         'tfidfvectorizer__min_df': [1, 2],
                         'tfidfvectorizer__ngram_range': [(1, 1), (1, 2),
                                                          (2, 2), (2, 3),
                                                          (3, 3)],
                         'tfidfvectorizer__norm': ('l1', 'l2')})
```

In [62]:
```
y_pred_nb_tfidf = gs_nb_tfidf.predict(X_val)
accuracy_score(y_val, y_pred_nb_tfidf)
```

Out[62]:
```
0.658498759305211
```

In [63]:
```
plot_cm_senti(gs_nb_tfidf, y_val, y_pred_nb_tfidf), classify(y_val, y_pred_nb_tfidf)
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.63      0.58       854
           1       0.54      0.67      0.59       448
           2       0.78      0.67      0.72      1922

    accuracy                           0.66      3224
   macro avg       0.61      0.66      0.63      3224
weighted avg       0.68      0.66      0.66      3224
```

Out[63]: (None, None)

Best Model - Gridsearch with Multinomial Bayes, Tfidf Vectorizer and SMOTE

After many iterations, the Multinomial Bayes model has proved to be the best model. This model although not performing the highest in accuracy overall has the best results with positive and negative tweets, without losing too much accuracy on the neutral tweets. Predictions on the rest of the model did a great job on neutral tweets, but was lacking in identifying negative tweets, which was the goal of this study. Overall this model also only lost a mere 3% on the overall accuracy compared to the XGboost model.

- When finding a model for predicting the sentiment of a tweet, the multinomial bayes using gridsearch cross validation showed the best recall for learning how to maximize false negatives.
- The gridsearch model performed at a 66% accuracy
- For class 0 (positive), a recall of 0.63 indicates that the classifier correctly identified 63% of the actual positive instances. In other words, 63% of the tweets that were positive were correctly classified as positive.
- For class 1 (negative), a recall of 0.67 indicates that the classifier correctly identified 67% of the actual negative instances. In other words, 67% of the tweets that were negative were correctly classified as negative.

- For class 2 (neutral), a recall of 0.67 indicates that the classifier correctly identified 67% of the actual neutral

Recommendations

For positive sentiment:

- look for tweets containing these key words or phrases:
  - 'new social network'
  - 'store downtown austin'
  - 'social network call'
  - 'launch major new'
  - 'major new social'

For Negative sentiment:

- look for tweets containing these key words or phrases:
  - 'one outlet computer'
  - 'outlet computer need'
  - 'need battery future'
  - 'computer need battery'
  - 'store nyc anger'
  - 'protest store nyc'

Future Work

Moving forward, some essential moves that will be needed in order to improve the model and business usage will be to increase the data sample and find one that is more balanced. The companies that may be interested in this will also want to look at tweets by product as well to focus more on which products are producing the worst or best sentiment and may be the leader in affecting stock prices by tweets.

In [ ]: