

Flatiron Phase 4 Project

Aaron Galbraith

<https://www.linkedin.com/in/aarongalbraith> (<https://www.linkedin.com/in/aarongalbraith>)

<https://github.com/aarongalbraith> (<https://github.com/aarongalbraith>)

Submitted: October 10, 2023

Contents

- [Business Understanding](#)
- [Data Understanding](#)
- [Data Preparation](#)
- [Exploration](#)
- [Modeling](#)
- [Evaluation](#)
- [Recommendations](#)
- [Further Inquiry](#)

Business Understanding

Apple launched the iPad 2 on March 11, 2011, the same day that the 2011 SXSW Festival began in Austin, TX. Apple also launched a pop-up store in Austin specifically to sell these and other products to the swell of crowds who attended the festival that year.* Apple product launches for their lines of iPods, iPhones, and iPads were a very big deal at the time, and much media coverage was devoted to the frenzy that accompanied each launch, e.g. Apple customers eagerly waiting in long lines for the newest product on the first day it was available for sale.

Apple can simply look to its accounting to see how successful its sales were in Austin during SXSW. But there is more to be learned than just how many dollars it made in the short term. By heavily promoting its product launches in an environment such as this festival, Apple encourages its customers (and loyal fans, and potential customers, and even detractors) to join in a conversation about them. This creates a great opportunity for Apple to get candid feedback on a massive scale about what it's doing that excites people as well as what disappoints people. This feedback can obviously inform future choices Apple makes in developing and launching its products.

Following the festival, Apple wished to gain insight into how its presence at the festival had been received. Tweets with the hashtag #sxsw were collected and labeled according to 1) what sentiment if any they expressed and 2) which if any tech brands or products (limited to Apple and Google) were mentioned. Apple wanted to know what it could learn not only from its own festival presence but also from Google's presence at the same festival.

*Essentially none of this information accompanied the dataset. Every single tweet contained the hashtag #sxsw, and a frequency analysis of the tweets indicated they took place in 2011. Further research yielded websites such as <https://techcrunch.com/2011/03/10/ipad-2-sxsw/> (<https://techcrunch.com/2011/03/10/ipad-2-sxsw/>) and <https://googleblog.blogspot.com/2011/03/google-at-sxsw-2011-austin-here-we-come.html> (<https://googleblog.blogspot.com/2011/03/google-at-sxsw-2011-austin-here-we-come.html>)

Data Understanding

Import files

Here we'll import all the tools we'll need (and quite a few more that we won't need).

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import seaborn as sns

from unidecode import unidecode

import nltk
from nltk.tokenize import RegexpTokenizer
from nltk import FreqDist
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import TweetTokenizer

from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, plot_roc_curve, plot_precision_recall_curve
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier
from sklearn.model_selection import train_test_split

from xgboost import XGBClassifier
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
import imblearn.pipeline
from operator import itemgetter
import string
from wordcloud import WordCloud

SEED = 19

do_grids = False
```

Load and briefly explore data set

```
In [2]: # read csv into dataframe
df = pd.read_csv('../data/tweets.csv', encoding='latin-1')
# show overview of data
df.info()
```

#	Column	Non-Null Count
0	tweet_text	9092 non-null
1	emotion_in_tweet_is_directed_at	3291 non-null
2	is_there_an_emotion_directed_at_a_brand_or_product	9093 non-null

dtypes: object(3)
memory usage: 213.2+ KB

```
In [3]: # show row and column counts
df.shape
```

```
Out[3]: (9093, 3)
```

```
In [4]: # show how many unique values for each feature
df.nunique()
```

```
Out[4]: tweet_text          9065
emotion_in_tweet_is_directed_at      9
is_there_an_emotion_directed_at_a_brand_or_product    4
dtype: int64
```

From the above we see that there are evidently some duplicated tweets; there are 4 different "emotion" labels; and there are 9 different product or brand labels.

```
In [5]: # show value counts for one feature
df.emotion_in_tweet_is_directed_at.value_counts()
```

```
Out[5]: emotion_in_tweet_is_directed_at
iPad                  946
Apple                 661
iPad or iPhone App     470
Google                430
iPhone                297
Other Google product or service 293
Android App            81
Android                78
Other Apple product or service 35
Name: count, dtype: int64
```

Apple products seem to be mentioned more than Google products.

```
In [6]: # show normalized value counts for one feature
round(df.is_there_an_emotion_directed_at_a_brand_or_product.value_counts(normalize=True), 2)
```

```
Out[6]: is_there_an_emotion_directed_at_a_brand_or_product
No emotion toward brand or product      0.59
Positive emotion                      0.33
Negative emotion                      0.06
I can't tell                          0.02
Name: proportion, dtype: float64
```

There are very few negative emotions expressed. The majority are neutral

```
In [7]: # show breakdown of sentiment labels for tweets that have no product or brand
round(df[df.emotion_in_tweet_is_redirected_at.isna()] \
    .is_there_an_emotion_directed_at_a_brand_or_product.value_counts(normalize=True), 2)
```

```
Out[7]: is_there_an_emotion_directed_at_a_brand_or_product
No emotion toward brand or product      0.91
Positive emotion                      0.05
I can't tell                          0.03
Negative emotion                      0.01
Name: proportion, dtype: float64
```

For tweets not associated with a brand, most are labeled neutral, but a few are not.

```
In [8]: # show examples of tweets that DO express emotion but are NOT directed at a brand
df[(df.is_there_an_emotion_directed_at_a_brand_or_product != 'No emotion toward brand or product') &
   (df.emotion_in_tweet_is_redirected_at.isna())]
].tweet_text
```

```
Out[8]: 46      Hand-Held %U+Hobo%U: Drafthouse launches %U+H...
        64      Again? RT @mention Line at the Apple store is ...
        68      Boooo! RT @mention Flipboard is developing an ...
        90      Thanks to @mention for publishing the news of ...
       102      %U+@mention "Apple has opened a pop-up st...
                               ...
       9043     Hey is anyone doing #sxsw signing up for the g...
       9049     @mention you can buy my used iPad and I'll pic...
       9052     @mention You could buy a new iPad 2 tmrw at th...
       9054     Guys, if you ever plan on attending #SXSW, you...
       9058     "Do you know what Apple is really good at...
Name: tweet_text, Length: 504, dtype: object
```

Clearly some/most of the tweets we can see here are associated with brands.

Summary of data

There are 9,093 records and 3 features. As there are only 9,065 unique tweets, it appears that there are some duplicates.

A little more than one third (3,291) of the tweets are identified as being directed at a particular product or brand associated with either Google or Apple, while the majority do not identify a product or brand.

Relatively few records have been identified as having a negative or "I can't tell" emotion.

For the 5,802 records that don't identify a product or brand, about 9% of them were identified as having something other than "no emotion". Upon investigation of these, it appears that some of them mention "Apple" or "iPad" after all, so evidently some tweets have not been successfully associated with a product or brand.

Data Preparation

Renaming features

The column names are a bit cumbersome, so we'll give them new names that are easier to deal with.

```
In [9]: # rename columns
df.rename(columns={'tweet_text': 'text',
                  'emotion_in_tweet_is_redirected_at': 'brand',
                  'is_there_an_emotion_redirected_at_a_brand_or_product': 'sentiment',
                  inplace = True)
```

Missing values

```
In [10]: # show records with missing text
df[df.text.isna()]
```

```
Out[10]:   text  brand          sentiment
            6    NaN    NaN  No emotion toward brand or product
```

We can't do anything with a record whose text is missing, so we'll drop it.

```
In [11]: # drop records with missing text values
df.dropna(subset=['text'], inplace=True)
```

Edit values

As these tasks may increase the number of duplicate records, we should perform them before we look for those duplicates.

Lower case

It's not likely that we'll lose anything important by shifting all the text to lower case, especially given the nature of tweeting.

```
In [12]: # shift all text to lower case  
df['text'] = df['text'].str.lower()
```

Rename and merge sentiments

The sentiment labels could be more succinct. We'll change them.

Also, since there were so few "unknown" sentiments, we'll just group those together with "neutral" sentiments.

```
In [13]: sentiment_rename = {  
    "No emotion toward brand or product": "neutral",  
    "Positive emotion": "positive",  
    "Negative emotion": "negative",  
    "I can't tell": "neutral"  
}  
  
df.sentiment = df.sentiment.apply(lambda x: sentiment_rename[x])
```

Merge brand labels

```
In [14]: # show breakdown of brand before merging  
df.brand.value_counts()
```

```
Out[14]: brand  
iPad                946  
Apple               661  
iPad or iPhone App  470  
Google              430  
iPhone              297  
Other Google product or service 293  
Android App         81  
Android             78  
Other Apple product or service 35  
Name: count, dtype: int64
```

```
In [15]: # assign either apple or google label and fill in missing values with other
df['brand'].replace(['iPad', 'Apple', 'iPad or iPhone App', 'iPhone', 'Other',
                     inplace=True)
df['brand'].replace(['Google', 'Other Google product or service', 'Android',
                     inplace=True)
df['brand'].fillna('other',
                     inplace=True)
```

```
In [16]: # show breakdown of brand after merging
df.brand.value_counts()
```

```
Out[16]: brand
other      5801
apple      2409
google     882
Name: count, dtype: int64
```

Detect missing brand labels

As noted earlier, we suspect many of the tweets labeled "other" actually refer to a certain product or brand. We'll use some helpful keywords to reclassify some of the tweets that are not yet associated with either brand.

In the event that some tweets happen to mention both brands, we'll make a label for "both", and we'll label everything else "neither".

```
In [17]: # make key word lists for apple and google
apple_words = ['ipad', 'apple', 'iphone', 'itunes', 'ipad2']
google_words = ['google', 'android', 'blogger', 'marissa', 'mayer', 'sketch'
# add hashtags
apple_hash = []
for word in apple_words:
    apple_hash.append('#'+word)
apple_words.extend(apple_hash)
google_hash = []
for word in google_words:
    google_hash.append('#'+word)
google_words.extend(google_hash)
# consolidate all terms in one list
brand_words = apple_words + google_words
```

```
In [18]: # make a function that relabels brand values by finding what keywords are in
def brand_fix(text, brand):
    # only relabel records that do not have one of the two brands already a
    if brand != 'other':
        return brand
    else:
        apple, google = False, False
        # look for apple keyword
        for word in apple_words:
            if word in text:
                apple = True
                break
        # look for google keyword
        for word in google_words:
            if word in text:
                google = True
                break

        # return correct new label
        if apple & ~google:
            return 'apple'
        elif google & ~apple:
            return 'google'
        elif apple & google:
            return 'both'
        else:
            return 'neither'
```

```
In [19]: # run above function to relabel brand values
df['brand'] = df.apply(lambda x: brand_fix(x.text, x.brand), axis=1)
# show breakdown of brand after running function
df.brand.value_counts()
```

```
Out[19]: brand
apple      5394
google     2845
neither     663
both       190
Name: count, dtype: int64
```

```
In [20]: df[df.brand == 'apple'].sentiment.value_counts(normalize=True)
```

```
Out[20]: sentiment
neutral     0.527994
positive    0.394883
negative    0.077123
Name: proportion, dtype: float64
```

```
In [21]: df[df.brand == 'google'].sentiment.value_counts(normalize=True)
```

```
Out[21]: sentiment
neutral     0.655185
positive    0.291740
negative    0.053076
Name: proportion, dtype: float64
```

We were able to label a vast majority of the unassociated tweets with a brand that the tweet mentions.

Duplicates

Now we'll address duplicated tweets

```
In [22]: # show how many records are duplicates  
df.duplicated().value_counts()
```

```
Out[22]: False    9050  
         True     42  
         Name: count, dtype: int64
```

Let's see if there is a difference if we only select for duplicated text (not product or sentiment).

```
In [23]: # show how many records are duplicates for the text value only  
df.duplicated(subset=['text']).value_counts()
```

```
Out[23]: False    9047  
         True     45  
         Name: count, dtype: int64
```

It looks like 3 text records are duplicated with either different sentiments or different associated brands. Let's look at what sentiment labels these were given, as separate groups.

```
In [24]: # show sentiment identification for groups of duplicated tweets
for i, index in enumerate(df.drop_duplicates()[df.duplicated(subset=['text'])]):
    print(
        'duplicate group', i+1, '\n',
        df.loc[index].text, '\n\n',
        df[df.text == df.loc[index].text].sentiment.value_counts(),
        '\n\n-----\n'
    )

duplicate group 1
win free ipad 2 from webdoc.com #sxsw rt

    sentiment
neutral      4
positive     2
Name: count, dtype: int64

-----
duplicate group 2
rt @mention marissa mayer: google will connect the digital & physical worlds through mobile - {link} #sxsw

    sentiment
neutral      5
positive     4
Name: count, dtype: int64

-----
duplicate group 3
rt @mention rt @mention it's not a rumor: apple is opening up a temporary store in downtown austin for #sxsw and the ipad 2 launch {link}

    sentiment
neutral      2
positive     1
Name: count, dtype: int64

-----
<ipython-input-24-7c19d15d088f>:2: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
for i, index in enumerate(df.drop_duplicates()[df.duplicated(subset=['text'])]).index:
```

```
In [25]: # show some of the duplicated tweets  
df[df.duplicated()].text.head(25)
```

```
Out[25]: 467      before it even begins, apple wins #sxsw {link}  
468      before it even begins, apple wins #sxsw {link}  
664      if you're in a room full of people w/good wi-f...  
775      google to launch major new social network call...  
776      google to launch major new social network call...  
798      google to launch major new social network call...  
2231     marissa mayer: google will connect the digital...  
2232     marissa mayer: google will connect the digital...  
2559     counting down the days to #sxsw plus strong ca...  
3810         win free ipad 2 from webdoc.com #sxsw rt  
3811         win free ipad 2 from webdoc.com #sxsw rt  
3812         win free ipad 2 from webdoc.com #sxsw rt  
3814         win free ipad 2 from webdoc.com #sxsw rt  
3950     really enjoying the changes in gowalla 3.0 for...  
3962     #sxsw is just starting, #ctia is around the co...  
4897     oh. my. god. the #sxsw app for ipad is pure, u...  
4954         40% of google maps use is mobile #sxsw  
5338     rt @mention %÷% go beyond borders! %÷_ {link} ...  
5341     rt @mention %÷% happy woman's day! make love, ...  
5842     rt @mention google launching secret new social...  
5880     rt @mention google to launch major new social ...  
5881     rt @mention google to launch major new social ...  
5882     rt @mention google to launch major new social ...  
5883     rt @mention google to launch major new social ...  
5884     rt @mention google to launch major new social ...  
Name: text, dtype: object
```

It's a tough call what to do with these duplicates. Some of them, like the first one, could be multiple people sharing the same article, and it could be meaningful to count all such instances, as they represent *more* of that sentiment. Some others, however, like the "win free ipad 2", appear to be from a business promoting itself. In that case, we wouldn't want to skew our results by counting all such instances.

In any event, due to the nature of tweeting, it is certainly plausible that these duplicated tweets were not erroneously duplicated, but rather they were actually separate, if identical, tweets when they were posted.

Several duplicates we see here start with "rt". We know that "rt" means "retweet", which specifically is a way for Twitter users to amplify a tweet they agree with.

Let's compromise on the duplicates by keeping all retweets but dropping the other duplicates.

```
In [26]: # combine ALL retweets with NON-DUPLICATED other tweets
df = pd.concat([
    # retweets
    df[df.text.str.startswith('rt')], 
    # non-retweets, minus duplicates
    df[~df.text.str.startswith('rt')].drop_duplicates()
    # reset the index
]).reset_index()
# drop newly created index column
df.drop(columns='index', inplace=True)
```

```
In [27]: # show row and column counts
df.shape
```

```
Out[27]: (9071, 3)
```

Begin NLP

Now that we have the data set we want to work with, we'll use natural language processing techniques to help us analyze it.

First we'll create a list of all the tweets. As we tokenize and lemmatize, etc, we can always come back to this for the full context.

```
In [28]: # make list of all tweet texts
tweets = df.text.to_list()
```

Then we'll create a list of all the tokens. To do this, we'll use a tokenizer that is specifically designed to parse tweets from Twitter and a lemmatizer.

```
In [29]: # make tokenizer
tokenizer = TweetTokenizer(
    preserve_case=False,
    strip_handles=True
)

# create list of tokens from data set
tokens = tokenizer.tokenize(', '.join(tweets))

# remove hashtags and accents
tokens = [unidecode(word) for word in tokens if not word.startswith('#')] +
[unidecode(word[1:]) for word in tokens if word.startswith('#')]
```

We'll take this opportunity while lemmatizing to get rid of hashtags.

```
In [30]: # make lemmatizer
lemmatizer = WordNetLemmatizer()

# lemmatize the list of words
tokens_lemmatized = [lemmatizer.lemmatize(word) for word in tokens]
```

Let's look at the most frequently occurring tokens.

```
In [31]: # show the most frequently occurring tokens
FreqDist(tokens_lemmatized).most_common(25)
```

```
Out[31]: [(' ', 12561),
('sxsw', 9573),
('.', 5890),
('the', 4423),
('link', 4314),
('}', 4288),
('{', 4285),
('to', 3580),
('at', 3097),
('rt', 2952),
('ipad', 2670),
('a', 2567),
('for', 2544),
('google', 2451),
('!', 2368),
('apple', 2223),
('in', 1936),
(':', 1830),
('of', 1711),
('is', 1705),
('"', 1696),
('and', 1635),
('?', 1611),
('iphone', 1573),
('store', 1518)]
```

This list is utterly dominated by stopwords. In addition to punctuation characters, some twitter-specific terms appear here, as well as some ordinary stopwords, and of course sxsw. Let's start a stopwords list and put it to use.

```
In [32]: # obtain the standard list of stopwords
nltk.download('stopwords', quiet=True)
# start our own list of stopwords with these words
stop_list = stopwords.words('english')
# add to this list some twitter-specific terms
stop_list.extend(['sxsw', 'link', 'rt'])
# add punctuation characters
for char in string.punctuation:
    stop_list.append(char)
# add empty string
stop_list.extend(['', 'ha', 'wa'])
```

```
In [33]: # make stopped list of tokens
tokens_stopped = [word for word in tokens_lemmatized if word not in stop_li
```

```
In [34]: # show the most frequently occurring tokens
FreqDist(tokens_stopped).most_common(25)
```

```
Out[34]: [('ipad', 2670),
('google', 2451),
('apple', 2223),
('iphone', 1573),
('store', 1518),
('2', 1370),
('new', 1087),
('austin', 956),
('app', 819),
('launch', 688),
('circle', 685),
('social', 644),
('...', 639),
('android', 588),
('today', 571),
('network', 471),
('get', 453),
('line', 442),
('via', 435),
('u', 434),
('pop-up', 422),
('party', 401),
('free', 383),
('called', 358),
('mobile', 343)]
```

This looks much better.

It seems quite probable that the "2" here is often occurring when tweets include a space in the expression "ipad 2". Let's look at them in context to see if we're right.

First we'll write a function to pull some random tweets that feature a given term or phrase.

```
In [35]: # a function that displays several randomly chosen tweets that include a given term
def tweet_samples(term, count=5):
    relevant_tweets = [tweet for tweet in tweets if term in tweet]
    if len(relevant_tweets) > 0:
        count = min(count, len(relevant_tweets))
        random_tweets = np.random.choice(relevant_tweets, count)
        for tweet in random_tweets:
            print(tweet)
    else:
        print('No tweets contain this phrase.')
```

```
In [36]: tweet_samples('2', count=5)
```

```
the ipad 2 launch is all the buzz at #sxsw 2011 {link} via @mention @mention
#apple #tablet #marketing #advertising #ipod
catch 22%û_i mean ipad 2 at #sxsw : {link}
rt @mention text "sxsw" to 24587 for chance to win ipad! drawing
3/17 at friends #sxsw. must be present to win! see ya there! {link}
expect mob scenes on w 6th st & congress {link} #sxsw #ipad2
#imanidiot to wait in line for an #ipad2 during #sxsw. but will have one!
@mention apple store, sxsw {link}
```

Indeed, it looks like a lot of those 2s are really part of iPad 2.

Let's write a quick function to join those words together in future lemmatization.

```
In [37]: def ipad_fix(word_list):
    new_word_list = []
    for i in range(len(word_list)):
        if i < len(word_list) - 1 and word_list[i] == 'ipad' and word_list[i+1] == '2':
            new_word_list.append('ipad2')
        elif i > 0 and word_list[i-1] == 'ipad' and word_list[i] == '2':
            continue
        else:
            new_word_list.append(word_list[i])
    return new_word_list
```

```
In [38]: tokens_stopped = ipad_fix(tokens_stopped)
```

```
In [39]: # show the most frequently occurring tokens  
FreqDist(tokens_stopped).most_common(25)
```

```
Out[39]: [('google', 2451),  
 ('apple', 2223),  
 ('iphone', 1573),  
 ('ipad', 1541),  
 ('store', 1518),  
 ('ipad2', 1423),  
 ('new', 1087),  
 ('austin', 956),  
 ('app', 819),  
 ('launch', 688),  
 ('circle', 685),  
 ('social', 644),  
 ('...', 639),  
 ('android', 588),  
 ('today', 571),  
 ('network', 471),  
 ('get', 453),  
 ('line', 442),  
 ('via', 435),  
 ('u', 434),  
 ('pop-up', 422),  
 ('party', 401),  
 ('free', 383),  
 ('called', 358),  
 ('mobile', 343)]
```

By the way, we can use term frequency to try to figure out what year all of this took place. We'll start around the year the first iPhone was released (2007) and include the year the dataset was created (2013).

```
In [40]: # iterate over a likely timespan  
for year in range(2007,2014):  
    # show how frequently a given year was mentioned  
    print(str(year)+':', FreqDist(tokens_stopped)[str(year)])
```

```
2007: 0  
2008: 0  
2009: 1  
2010: 4  
2011: 75  
2012: 0  
2013: 0
```

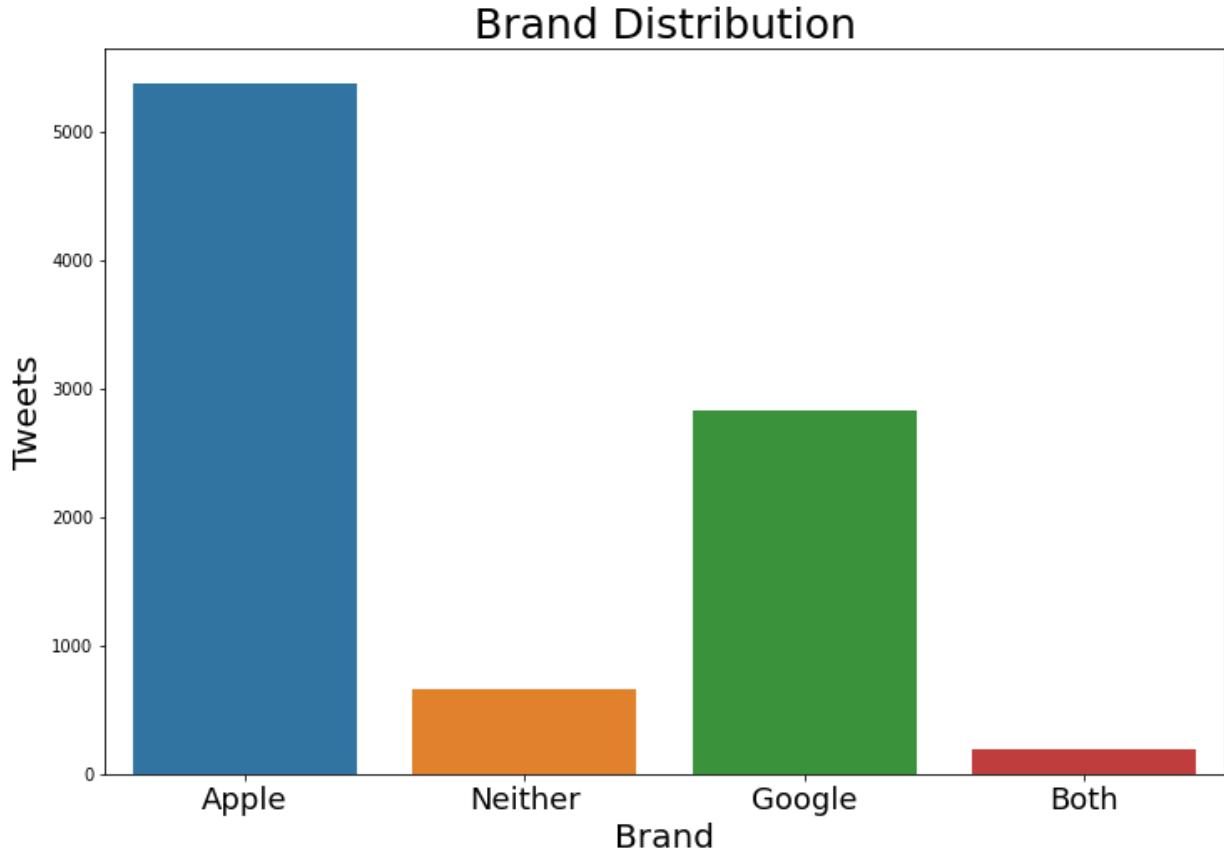
This is how we deduced that the data comes from 2011 (which allowed us to learn more context for the SXSW conference in Austin, TX from that particular year).

Exploration

Overview

```
In [41]: labels = ['Apple', 'Neither', 'Google', 'Both']
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(df['brand'])
plt.title('Brand Distribution', fontsize=25)
# ax.set_yticklabels([0,500,1000,1500,2000,2500,3000,3500], fontsize=18)
ax.set_xticklabels(labels, fontsize=18)
plt.xlabel('Brand', fontsize=20)
plt.ylabel('Tweets', fontsize=20)
# plt.legend(loc=1, prop={'size': 15})
plt.show()

/Users/stubbletrouble/opt/anaconda3/envs/learn-env/lib/python3.8/site-pac
kages/seaborn/_decorators.py:36: FutureWarning: Pass the following variab
le as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
    warnings.warn(
```



The most striking aspect of this image is that Apple was mentioned roughly twice as often as Google.

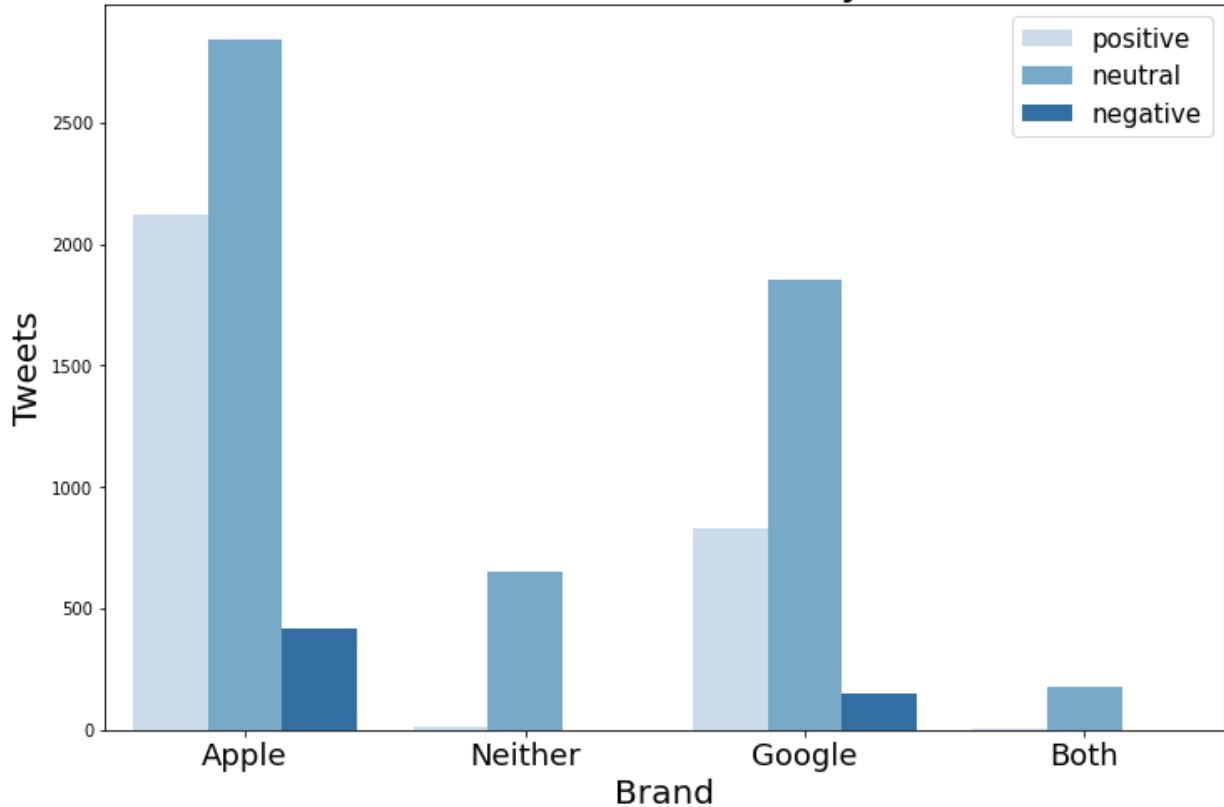
It also looks like we've done a pretty good job of associating as many tweets as possible with one or both brands. The original data set had far too many records without a brand label for us to do any meaningful exploration.

```
In [42]: labels = ['Apple', 'Neither', 'Google', 'Both']
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(df['brand'], hue=df['sentiment'], palette='Blues')
plt.title('Sentiment Distribution by Brand', fontsize=25)
# ax.set_yticklabels([0,500,1000,1500,2000,2500], fontsize=18)
ax.set_xticklabels(labels, fontsize=18)
plt.xlabel('Brand', fontsize=20)
plt.ylabel('Tweets', fontsize=20)
plt.legend(loc=1, prop={'size': 15})
plt.show()

/Users/stubbletrouble/opt/anaconda3/envs/learn-env/lib/python3.8/site-pac
kages/seaborn/_decorators.py:36: FutureWarning: Pass the following variab
le as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.

warnings.warn(
```

Sentiment Distribution by Brand



Tweets that mentioned both or neither brand almost exclusively lacked an identifiable sentiment. These will only factor into some parts of our analysis.

Apple and Google had a similar distribution of positive, neutral and negative tweets — neutral tweets dominated both brands, and positive tweets outnumbered negative ones by something like a 5:1 ratio. This will create some class imbalance issues during modeling.

Word clouds

We'll be making several word clouds, so let's create a function that streamlines creating and plotting them.

```
In [43]: # a function that generates a word cloud of a given list of words
def make_wordcloud(wordlist, colormap='Greens', title=None):
    # instantiate wordcloud
    wordcloud = WordCloud(
        width=600,
        height=400,
        colormap=colormap,
        collocations = True
    )
    return wordcloud.generate(', '.join(wordlist))

def plot_wordcloud(wordcloud):
    # plot wordcloud
    plt.figure(figsize = (12, 15))
    plt.imshow(wordcloud)
    plt.axis('off');
```

Let's look at a word cloud of all the stopped words.

```
In [44]: # word cloud of stopped words  
plot_wordcloud(make_wordcloud(tokens_stopped))
```



This is as expected, but largely dominated by brand names. Let's try it again without any of the brand words.



This is not very illuminating just yet. We'll create some new word lists that focus just on the positives and negatives of each brand.

```
In [46]: # a function that takes a list of tweets and 1) tokenizes them, 2) lemmatizes
def make_tokens(tweets):
    # make list of tokens
    tokens = tokenizer.tokenize(', '.join(tweets))
    # lemmatize tokens
    tokens_lemmatized = [lemmatizer.lemmatize(word[1:]) for word in tokens]
    [lemmatizer.lemmatize(word) for word in tokens if not word.startswith('
    # stop tokens
    tokens_lemmatized = ipad_fix(tokens_lemmatized)
    tokens_stopped = [word for word in tokens_lemmatized if \
        word not in stop_list and \
        word not in brand_words]

    return tokens_stopped
```

```
In [47]: # make tokens lists for all / positive / negative tweets, for both apple and google
tokens_apple_all = make_tokens(df[df.brand == 'apple'].text.to_list())
tokens_apple_pos = make_tokens(df[(df.brand == 'apple') & (df.sentiment == 1)].text.to_list())
tokens_apple_neg = make_tokens(df[(df.brand == 'apple') & (df.sentiment == -1)].text.to_list())
tokens_google_all = make_tokens(df[df.brand == 'google'].text.to_list())
tokens_google_pos = make_tokens(df[(df.brand == 'google') & (df.sentiment == 1)].text.to_list())
tokens_google_neg = make_tokens(df[(df.brand == 'google') & (df.sentiment == -1)].text.to_list())
```

```
In [48]: # show positive and negative wordclouds for apple side by side
fig, ax = plt.subplots(figsize=(32,24), ncols=2)
# plt.suptitle('Positive and Negative Word Clouds for Apple', fontsize=25)
ax[0].imshow(make_wordcloud(tokens_apple_pos))
ax[0].axis('off')
ax[1].imshow(make_wordcloud(tokens_apple_neg, colormap='Reds'))
ax[1].axis('off')
plt.tight_layout();
```



A lot of the positive buzz about Apple seems to be about the pop-up store in downtown Austin — terms like "temporary store", etc. There are even positive sentiments about the "line" there.

Some negative terms that stand out are "design headache" and "battery", as well as several phrases about a "fascist company" or an "elegant fascist". Also someone named "Kara Swisher" shows up.

Let's explore some context.

```
In [49]: tweet_samples('temporary store')
```

apple to open temporary store for #sxsw. interesting. {link}
rt @mention apple to open temporary store for #sxsw at 6th and congress
{link}
i really could use one of those tiny macbook airs. did i read that apple
set up a temporary store at sxsw? where is it? #sxsw
omarg: it's not a rumor: apple is opening up a temporary store in downtown
austin for #sxsw and the ipad 2 launch {link}
apple to open temporary store for #sxsw. interesting. {link}

```
In [50]: tweet_samples('line', count=10)
```

rt @mention line at popup apple store at 6th & congress still over a block long. #sxsw // fucking californians.
no surprise, the line for the google & bing q&a is wrapped through the austin hilton. what would you ask, @mention #sxsw
how could i forget my cell phone today for #sxsw what a fail glad i have my ipad #lifelinetotheworld
temporary apple store will be a block from my office! can't wait to see the line tomorrow. photos to come... #sxsw #ipad2
<10 minute line for ipad 2 at the temporary apple store #sxsw
the ipad 2 line is only about 20 minutes long #sxsw
200 people in line at pop up apple store at #sxsw
mashable was there to talk to line takers, many of whom spent the better part of the ... #ipad #sxsw #gadgets {link}
i've been having meetings while i'm in line at the #sxsw popup apple store for the ipad2. i love this place!
march 11. austin, tx. will people line up at the #sxsw registration or at the apple store? ^_^

```
In [51]: tweet_samples('design headache')
```

checking out ipad design headaches (2 tablets, call in the morning).
{link} #sxsw #tapworthy #fb
room is full here at re ipad design headaches talk. #sxsw
i just watched "ipad design headaches" at #sxsw. buttons are a hack. #tapworthy
headed for ipad design headaches (2 tablets, call in the morning) #sxsw
{link}
room is full here at re ipad design headaches talk. #sxsw

```
In [52]: tweet_samples('battery')
```

thanks to my good friend, the great @mention for gifting me an exogear ip hone backup battery case. priceless at #sxsw
so true rt @mention if there was a popup store in austin that sold nothing but iphone battery extenders, it would make so much money. #sxsw
tip: turn off push & set fetch to manual mode for email (mail, contacts & cal > fetch new data) - saves battery. #iphone #sxsw
see vanessa at the mophie booth 1406 for ipad/iphone backup battery. life saver! #sxsw
scored a #mophie juice pack at the #tradeshow #sxsw. double your iphone battery!

```
In [53]: tweet_samples('kara swisher')
```

apple...“the classiest fascist company in america”; kara swisher
#sxsw
rt @mention %“the classiest fascist company in existence” - kara swisher #sxsw #apple
rt @mention “apple: the most elegant fascist corporation in america today.” -- kara swisher #sxsw #flipboard
rt @mention %“the classiest fascist company in existence” - kara swisher #sxsw #apple
%“the classiest fascist company in existence” - kara swisher #sxsw #apple

Summary of Apple's word clouds

A lot of people are talking about Apple's temporary pop-up store and reporting on the experience of waiting in line, not necessarily complaining about it.

Evidently Kara Swisher is a person who made a snarky comment about Apple being fascist.

It looks like some media was using the phrase "design headache" in reference to the (new or old?) iPad, but the phrase was used beyond just the tweets that linked to certain articles.

Some people mentioned their iPhone batteries dying. This appears to be a common problem.

Let's do the same for Google.

```
In [54]: # show positive and negative wordclouds for google side by side
fig, ax = plt.subplots(figsize=(32,24), ncols=2)
ax[0].imshow(make_wordcloud(tokens_google_pos))
ax[0].axis('off')
ax[1].imshow(make_wordcloud(tokens_google_neg, colormap='Reds'))
ax[1].axis('off')
plt.tight_layout();
```



We'll want to investigate some prominent terms here to see what people are saying about "party", "map", "circle", and new social network". We'll again use the function that shows sample tweets for each term or phrase.

```
In [55]: tweet_samples('party', count=10)

chimpit party, plutopia, google, maybe a movie. #sxsw starts today for me :)
the bus is leaving the google party and headed to the power plant party at 214 w ave. #sxsw
rt @mention #sxsw @mention in her 80s splendor dancin to the spazzmatics at aclu google #partylikeits1986 ! {link}
google party @mention maggie maes rocks like it's 1986. #sxsw
attending the @mention party at #sxsw with some very interested people!!!! #wiings
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
congrats to @mention for winning the ipad 2 raffled at the #smileyparty. check your inbox for details! #sxsw
did @mention just effectively kill their 3rd party developer base? {link}
#sxsw #ismparty
#redbull bpm dj ipad app is powering this party. #sxsw #dtas @mention republic live {link}
{link} samsung galaxy s ii appears at fcc and team android #sxsw party {link} #android #followback
```

```
In [56]: tweet_samples('maps')

rt @mention "40% of all google maps usage is on mobile" - marissa mayer #sxsw
rt @mention could a game improve ecodriving habits? rt @mention %u00b35 million miles a day are driven while being google maps navigation assisted.&quot;#sxsw
why the heck doesn't google maps use browser geo-location? am i missing it somehow? using chrome. @mention #sxsw #startupbus
learned from sponsored #sxsw software: 1) austin = nice walking city; 2) mapquest = piece of shit mapping software. (go google maps!) #fb
#marissagoogles #sxsw google maps for mobile using vectors is fast and three dimensional
```

```
In [57]: tweet_samples('circles', count=10)
```

```
so we get to see google fail at social on another day rt @mention okay, n  
o google circles debuting at #sxsw today  
will google circles take on facebook? {link} #circles #sxsw  
will google circles take on facebook? {link} #circles #sxsw  
how about this: will be late rt @mention google circles will be _____  
#sxsw  
hmm. rt @mention we interrupt your regularly scheduled #sxsw geek progra  
mming with big news {link} #google #circles  
[just fyi...] breaking: google to launch major new social network called  
circles...today? {link} via @mention #sxsw #socialmedia  
false alarm: google circles not coming now&and probably not ever {link}  
more {link} #sxsw #sxswi  
i'm in! rt @mention rt @mention google to launch major new social network  
called circles, possibly today {link} #sxsw  
really? so, no google me or circles for now? rt @mention we're not launch  
ing any products at #sxsw but we're doing plenty else.  
interesting rt @mention @mention to launch major new social network calle  
d circles, possibly today {link} #sxsw
```

```
In [58]: tweet_samples('new social')
```

```
via @mention google to launch major new social network called circles, po  
ssibly today {link} #sxsw  
will we see google's new social network "circles" today? {link}  
#sxsw #sxswi  
can google do social? we may shortly find out rt @mention google to launc  
h major new social network called circles at #sxsw {link}  
#sxsw news: google to launch major new social network called circles, pos  
sibly today {link}  
rt @mention google set to launch new social network #circles today at #sx  
sw
```

```
In [59]: tweet_samples('bing')
```

```
rt @mention google and bing meet at #sxsw after their lovers tiff last mo  
nth but it seems to have been disappointingly friendly {link}  
the ppl have spoken "great #sxsw session w/ bing, google and danny s  
ullivan. bring it back next year with a larger room!" : #qagb #sxswi  
no surprise, the line for the google & bing q&a is wrapped throug  
h the austin hilton. what would you ask, @mention #sxsw  
despite drawing giant crowd, google-bing q&a discussion is very insid  
e baseball. #sxsw  
this google/bing q&a panel is like the world's most expensive seo con  
sultation. #sxsw
```

Summary of Google's word clouds

It's clear that Google's attempt at a new social network called Circles was not successful.

However, there was a lot of genuinely positive sentiment about Google Maps and its potential.

Modeling

Here we will classify only tweets that have positive or non-positive sentiments.

```
In [60]: df_posnon = df.copy()
# convert sentiment to 0s and 1s
# both 'negative' and 'neutral' will have zero values
df_posnon.sentiment = df_posnon.sentiment.apply(lambda x: 1 if x == 'positive' else 0)
```

```
In [61]: df_posnon.sentiment.value_counts()
```

```
Out[61]: sentiment
0    6102
1    2969
Name: count, dtype: int64
```

Tokenizer

We'll use what we developed during exploration to make a custom tokenizer function. This will perform all of the following:

- tokenize tweets
- remove accents and hashtags
- lemmatize tokens
- join "ipad" and "2" where appropriate

This tokenizer will *not* remove stopwords, as we may wish to vary our choice of stopwords in the models.

```
In [62]: # make tokenizer
def custom_tokenize(document):
    # instantiate tokenizer
    tokenizer = TweetTokenizer(
        preserve_case=False,
        strip_handles=True
    )

    # create list of tokens from data set
    tokens = tokenizer.tokenize(document)

    # remove hashtags and accents
    tokens = [unidecode(word) for word in tokens if not word.startswith('#')
              [unidecode(word[1:])] for word in tokens if word.startswith('#')]

    # remove stop words
    tokens = [word for word in tokens if not word in stop_list]

    # instantiate lemmatizer
    lemmatizer = WordNetLemmatizer()

    # lemmatize the list of words
    tokens_lemmatized = [lemmatizer.lemmatize(word) for word in tokens]

    # perform ipad fix
    tokens_lemmatized = ipad_fix(tokens_lemmatized)

return tokens_lemmatized
```

Train and test sets

We'll split the data into train and test sets.

```
In [63]: # split the data into target (sentiment) and predictor (text)
X, y = df_posnon['text'], df_posnon['sentiment']

# split the data into train and test sets
# set random state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

Metric choice = accuracy

As we've chosen to develop a binary classifier, and our classes aren't wildly out of balance (roughly a 2:1 ratio), we will be satisfied with a model that merely achieves accuracy, which is the measure of the percentage of tweets that it correctly identifies as positive or non-positive.

If we were developing a multi-class classifier, or if we were focusing on just positive versus negative, then there would be a significant class imbalance in either case. With a more significant class imbalance, it becomes easier (and less impressive) to achieve high accuracy. For example, if

Plurality check

It will be instructive to recall the percentage of the plurality in the target feature (sentiment). Models should be evaluated in relation to this.

```
In [64]: # save this value to compare to future model crossval scores
plurality_cv = round(y_train.value_counts(normalize=True)[1],4)
# show the sentiment breakdown
round(y_train.value_counts(normalize=True),4)
```

```
Out[64]: sentiment
0      0.6736
1      0.3264
Name: proportion, dtype: float64
```

```
In [65]: y_pred_train = y_train.copy().apply(lambda x: 0)
train_acc = round(accuracy_score(y_pred_train, y_train),4)
y_pred_test = y_test.copy().apply(lambda x: 0)
test_acc = round(accuracy_score(y_pred_test, y_test),4)
print(
    'Plurality Calculation',
    '\n\nTraining Score: '+str(train_acc),
    '\nTest Score:      '+str(test_acc)
)
```

```
Plurality Calculation

Training Score: 0.6736
Test Score:      0.6689
```

Results function

We'll create a function that takes the pipeline we've created and displays only the results we're interested in.

```
In [66]: def model_results(pipeline):
    # fit the pipeline to the training data
    pipeline.fit(X_train, y_train)
    # generate predictions for the test data
    y_pred = pipeline.predict(X_test)
    # display the training and test accuracy scores
    print(f"Training Score: {round(pipeline.score(X_train, y_train),4)} \
\nTest Score:      {round(pipeline.score(X_test, y_test),4)}")

    # plot the normalized confusion matrix
    plot_confusion_matrix(estimator=pipeline, X=X_test, y_true=y_test, cmap
        normalize='true',
        display_labels=['Non-positive', 'Positive'])
```

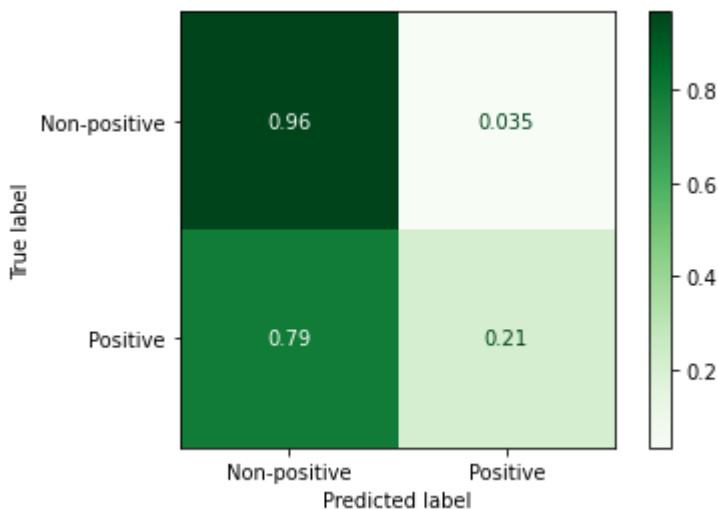
Naive Bayes (BASELINE MODEL)

Rough model

```
In [67]: # Multinomial Naive Bayes

# create pipeline
pipeline_nb = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', MultinomialNB())
])
print('Naive Bayes')
# create model from pipeline and display results
model_results(pipeline_nb)
```

```
Naive Bayes
Training Score: 0.7937
Test Score:      0.7152
```



This is already better than the 66.9% plurality. But the comparison of true non-positives (96%) to true positives (21%) suggests the model is very conservative about labeling tweets as positive. The gap between training and test scores also suggests some overfitting has occurred, which we wish to avoid.

We'll do a grid search in an effort to tune the model

```
In [68]: if do_grids == True:

    param_grid = {'clf_alpha': [0.001, 0.01, 0.1, 1]}
    gridsearch = GridSearchCV(pipeline_nb, param_grid=param_grid,
                             scoring='accuracy')

    gridsearch.fit(X_train, y_train)
    gridsearch.best_params_
```

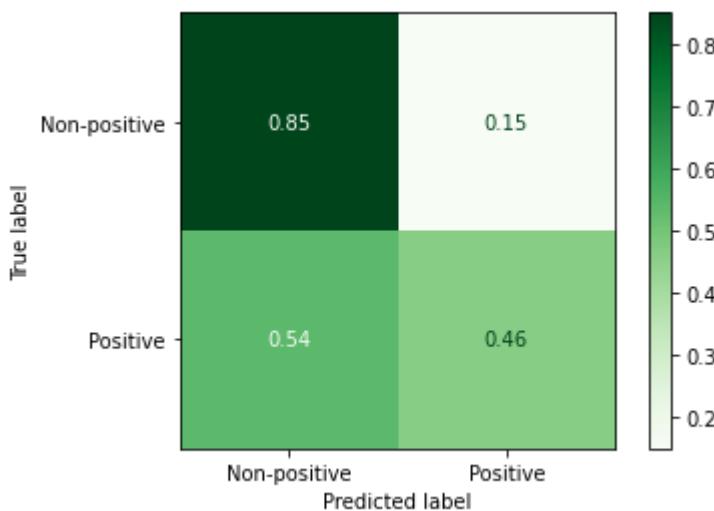
We'll use this alpha size in the tuned model.

Tuned Naive Bayes

```
In [69]: # Tuned Multinomial Naive Bayes

# create pipeline
pipeline_nb = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', MultinomialNB(alpha=0.1))
])
print('Naive Bayes')
# create model from pipeline and display results
model_results(pipeline_nb)
```

```
Naive Bayes
Training Score: 0.8899
Test Score:      0.7218
```



The confusion matrix suggests a somewhat more nuanced model, as the true positives and true non-positives are closer than before, but the widened gap between training and test scores suggests *more* overfitting.

We'll discard this tuned model and instead experiment with oversampling to see whether this fixes any issues relating to the class imbalance.

Naive Bayes with oversampling

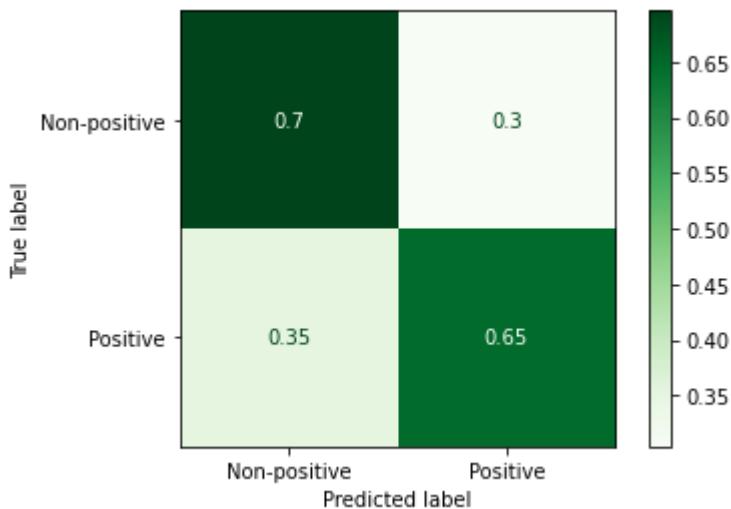
```
In [70]: # Multinomial Naive Bayes with oversampling

# create pipeline
pipeline_nb_os = imblearn.pipeline.Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set oversampler
    ('os', RandomOverSampler(random_state=SEED)),
    # set classifier
    ('clf', MultinomialNB(alpha=0.1))
])
print('Naive Bayes with oversampling')
# create model from pipeline and display results
model_results(pipeline_nb_os)
```

Naive Bayes with oversampling

Training Score: 0.8665

Test Score: 0.6799



The confusion matrix shows that this has brought the true positives and true non-positives much closer together, but unfortunately we're still clearly overfitting the training data.

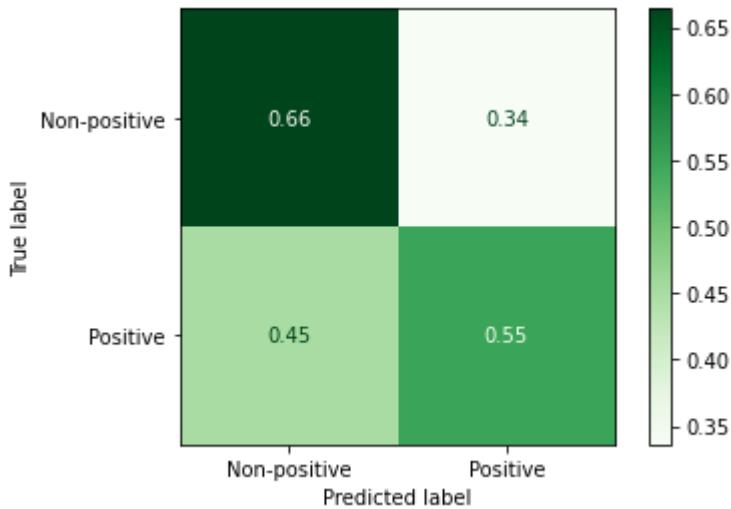
Logistic Regression

rough model

```
In [71]: # logistic regression

# create pipeline
pipeline_lr = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear'))
])
print('Logistic Regression')
# create model from pipeline and display results
model_results(pipeline_lr)
```

Logistic Regression
Training Score: 0.9593
Test Score: 0.6259



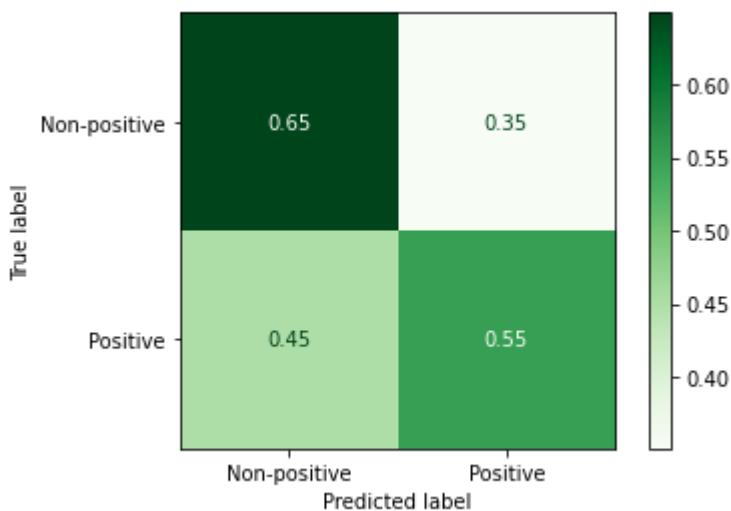
This is wildly overfitting the data.

Logistic Regression with oversampling

```
In [72]: # logistic regression with oversampling

# create pipeline
pipeline_lr_os = imblearn.pipeline.Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set oversampler
    ('os', RandomOverSampler(random_state=SEED)),
    # set classifier
    ('clf', LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear'))
])
print('Logistic Regression')
# create model from pipeline and display results
model_results(pipeline_lr_os)
```

```
Logistic Regression
Training Score: 0.9577
Test Score:      0.6165
```



This is still overfitting the training data. It seems that oversampling is not necessarily going to improve our models.

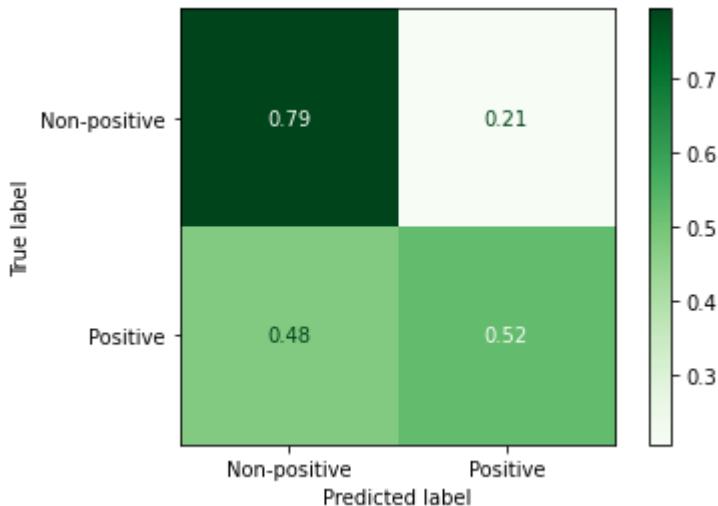
Decision Tree

Rough model

```
In [73]: # decision tree

# create pipeline
pipeline_dt = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', DecisionTreeClassifier(random_state=SEED))
])
print('Decision Tree')
# create model from pipeline and display results
model_results(pipeline_dt)
```

Decision Tree
Training Score: 0.9653
Test Score: 0.7047



These results are not particularly impressive. The confusion matrix looks a bit better than that of the baseline model, but the training/test gap again suggests significant overfitting of the training data.

We'll see if some tuning can help.

```
In [74]: if do_grids == True:

    param_grid = {'clf_criterion': ['gini', 'entropy'],
                  'clf_max_depth': [10, 20, None],
                  'clf_min_samples_leaf': [1, 2, 3]
                 }

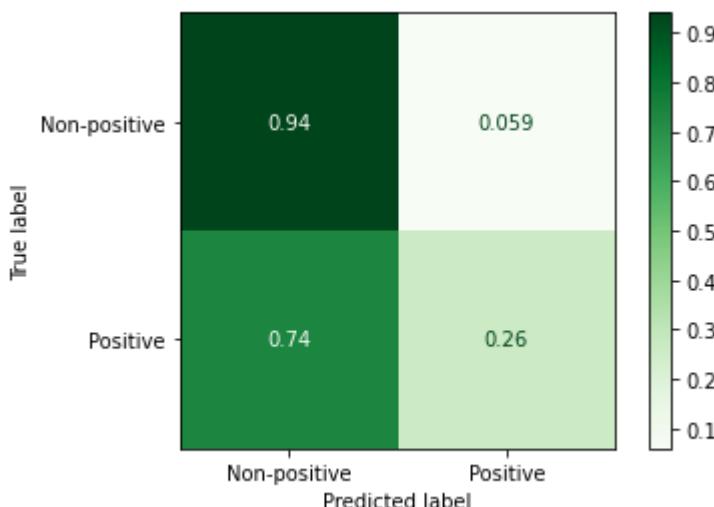
    gridsearch = GridSearchCV(estimator=pipeline_dt, param_grid = param_grid)
    gridsearch.fit(X_train, y_train)
    gridsearch.best_params_
```

Tuned Decision Tree

```
In [75]: # tuned decision tree

# create pipeline
pipeline_dt = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', DecisionTreeClassifier(
        criterion='gini',
        max_depth=20,
        min_samples_leaf=1,
        random_state=SEED))
])
print('Decision Tree')
# create model from pipeline and display results
model_results(pipeline_dt)
```

Decision Tree
Training Score: 0.7926
Test Score: 0.7157



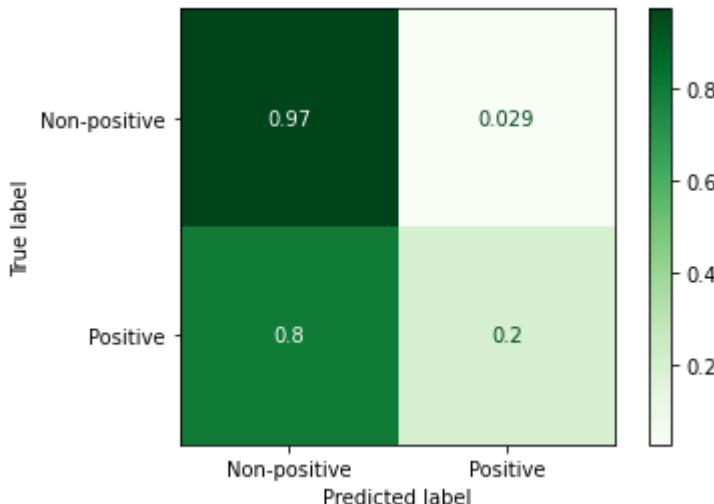
This helped the overfitting issue, but made the confusion matrix more lopsided. These results are practically indistinguishable from those of the baseline model.

Bagged Trees

```
In [76]: # bagged trees

# create pipeline
pipeline_bag = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', BaggingClassifier(
        DecisionTreeClassifier(criterion='entropy', max_depth=10), n_estimators
    ))
])
print('Bagged Trees')
# create model from pipeline and display results
model_results(pipeline_bag)
```

```
Bagged Trees
Training Score: 0.7472
Test Score:      0.7174
```



This has a nice overall result and doesn't appear to overfit the training data much at all. The confusion matrix, however, seems again quite lopsided.

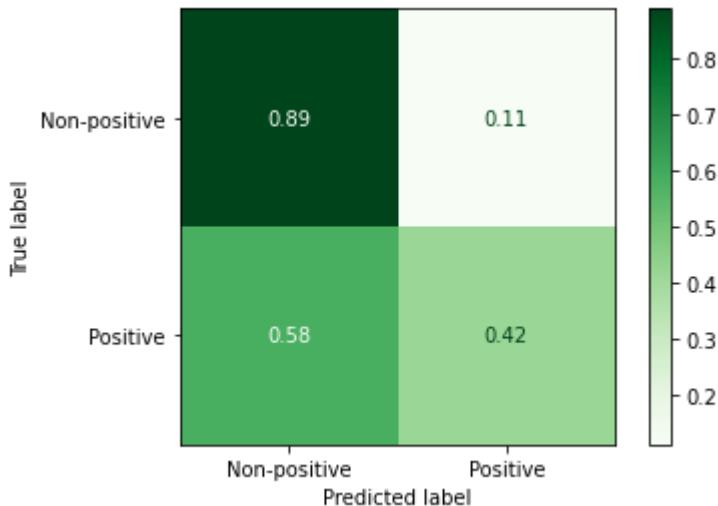
Random Forest

Rough model

```
In [77]: # random forest

# create pipeline
pipeline_rf = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', RandomForestClassifier(random_state=SEED))
])
print('Random Forest')
# create model from pipeline and display results
model_results(pipeline_rf)
```

Random Forest
Training Score: 0.9653
Test Score: 0.7317



This is the highest accuracy score we've seen so far. However, these results are not impressive in terms of solving the overfitting issue or the lopsided confusion matrix issue. Let's try tuning.

```
In [78]: if do_grids == True:

    param_grid = {'clf_criterion': ['gini', 'entropy'],
                  'clf_max_depth': [10, 20, None],
                  'clf_min_samples_leaf': [1, 2, 3]
                 }

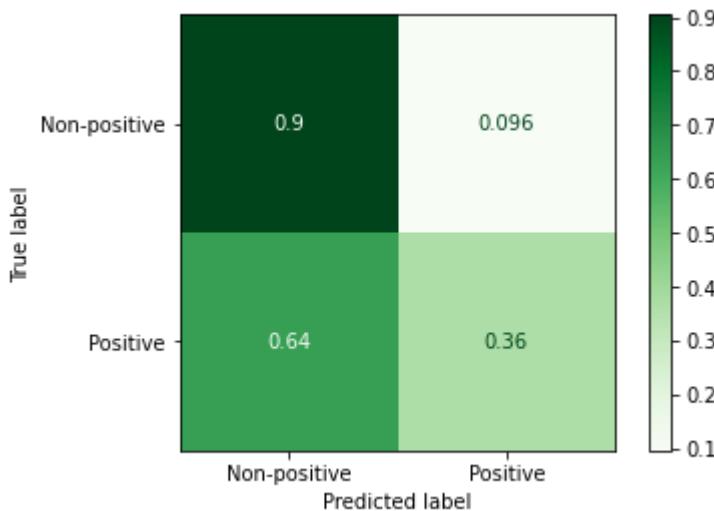
    gridsearch = GridSearchCV(estimator=pipeline_rf, param_grid = param_grid)
    gridsearch.fit(X_train, y_train)
    gridsearch.best_params_
```

Tuned Random Forest

```
In [79]: # tuned random forest

# create pipeline
pipeline_rf = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', RandomForestClassifier(criterion='entropy',
                                   max_depth=None,
                                   min_samples_leaf=2,
                                   random_state=SEED))
])
print('Random Forest')
# create model from pipeline and display results
model_results(pipeline_rf)
```

Random Forest
Training Score: 0.864
Test Score: 0.7245



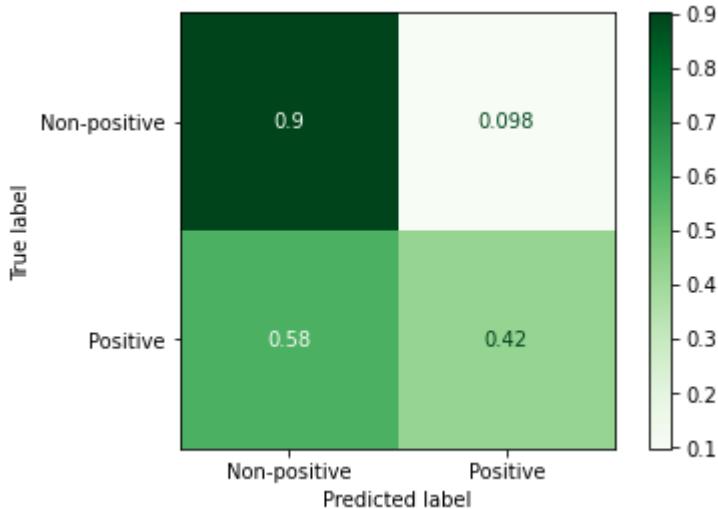
Tuning this model seems only to have worsened things.

Support Vector Machine

```
In [80]: # support vector machine

# create pipeline
pipeline_svm = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', SVC(random_state=SEED))
])
print('Support Vector Machine')
# create model from pipeline and display results
model_results(pipeline_svm)
```

Support Vector Machine
Training Score: 0.926
Test Score: 0.7433



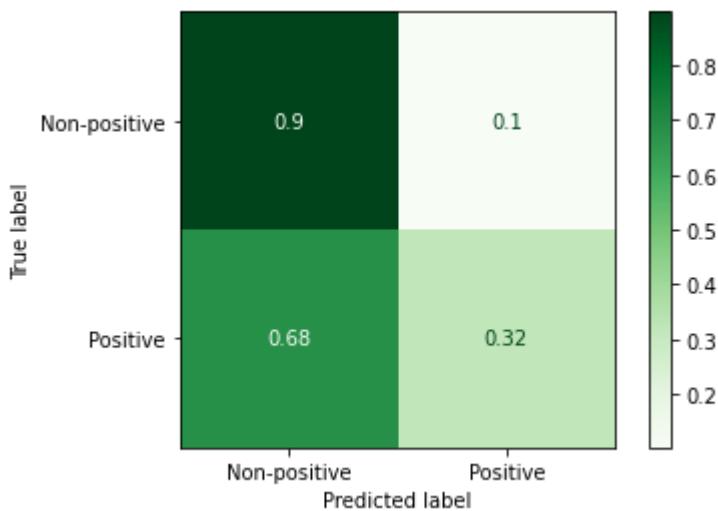
This gives an even higher accuracy score, but clearly overfits the training data and shows a lopsided confusion matrix.

AdaBoost

```
In [81]: # adaboost

# create pipeline
pipeline_ab = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', AdaBoostClassifier(random_state=SEED))
])
print('AdaBoost')
# create model from pipeline and display results
model_results(pipeline_ab)
```

```
AdaBoost
Training Score: 0.7408
Test Score:      0.7063
```



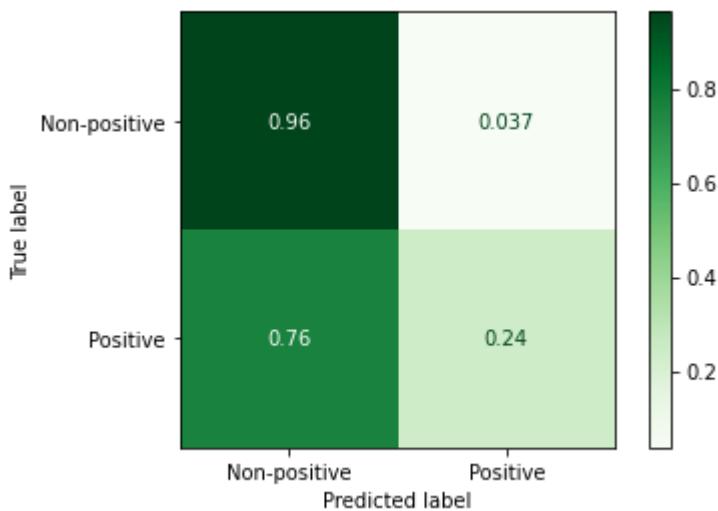
This is decent. It doesn't seem to overfit the data by too much. The true positives are rather low still.

Gradient Boost

```
In [82]: # gradient boost

# create pipeline
pipeline_gb = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', GradientBoostingClassifier(random_state=SEED))
])
print('Gradient Boost')
# create model from pipeline and display results
model_results(pipeline_gb)
```

```
Gradient Boost
Training Score: 0.7494
Test Score:      0.7229
```



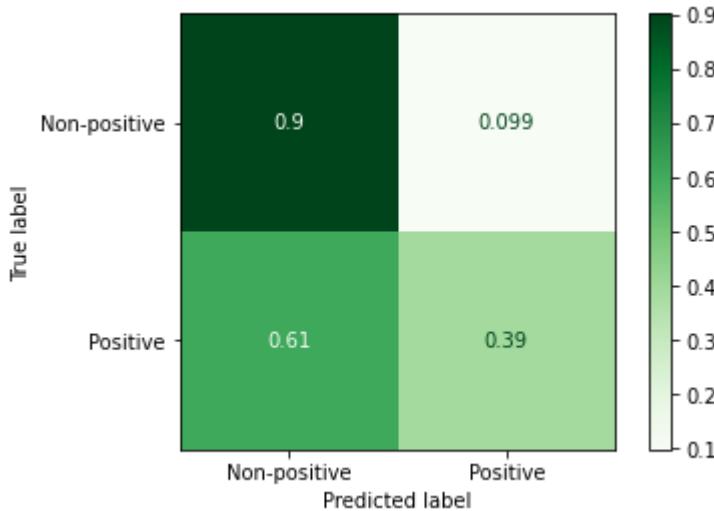
This is very impressive in terms of not overfitting and giving a high accuracy score, the best combination yet. The recall for positive tweets (24%) is not great, and could be worth revisiting in further inquiry.

XGBoost

```
In [83]: # XGB

# create pipeline
pipeline_xgb = Pipeline([
    # set vectorizer
    ('vectorizer', TfidfVectorizer(
        tokenizer = custom_tokenize,
        stop_words = stop_list
    )),
    # set classifier
    ('clf', XGBClassifier(random_state=SEED))
])
print('Gradient Boost')
# create model from pipeline and display results
model_results(pipeline_xgb)
```

```
Gradient Boost
Training Score: 0.8408
Test Score:      0.7328
```



This model gives another good accuracy score but unfortunately overfits the data.

Evaluation

We attempted to oversample the data in order to deal with the class imbalance issue. This only gave worse results each time we tried it.

None of our efforts at tuning the models made a positive impact either.

Most of the models seem to overfit the training data, as evidenced by the large gap between training accuracy and test accuracy. The only models where the two accuracy scores seemed reasonably close were Bagged Trees, AdaBoost, and Gradient Boost. Of these models, Gradient Boost gave the best test accuracy, so we chose Gradient Boost as our final model.

Recommendations

1. Evidence suggests the pop-up store was very popular. This was an effective way to get people excited about the product at a time when they could share their excitement with others around them. This event should be repeated if possible.
2. Apple should consider addressing battery life and design issues with some of their products. These topics didn't fully dominate the discussion by any means, but they were the most significant of Apple's negative topics of any substance.
3. The party Google hosted was clearly very popular and appeared to drive a lot of what buzz they enjoyed at the festival. Apple should consider hosting parties at festivals in a similar manner.
4. A Gradient Boost model will be the most effective model in distinguishing positive from non-positive sentiments in tweets about tech brands and products. Apple can make use of this model to continue future analysis of public response to its operations.

Further Inquiry

More sophisticated modeling techniques might be able to better analyze either a direct positive v. negative comparison or even a multi-class analysis (positive, negative, and neutral). The class imbalances make this difficult.

More direct analysis could be done with the tweets that mentioned *both* Apple and Google brands. Perhaps these tweets feature direct comparisons that could be very illuminating.

With more time we would have liked to explore feature importances of the various models.

We would also like to have explored *why* the models were overfitting the training data so consistently and what aspects could have been changed to prevent this.

We would have liked to investigate other features, such as tweet length (counting both characters and words), to see if that added anything to the models.

It may also be worth rethinking our evaluation metric. It could be of more value to prioritize true positives (recall) than just focusing on accuracy.