Predicting Brand Sentiment on Twitter

- Nick Gigliotti
- ndgigliotti@gmail.com

Business Problem

Apple has asked me to create a strong predictive model for detecting positive, negative, and neutral sentiment in tweets. They are primarily concerned with tweets about their company and products, but also might want to know what people are saying about competitors. They intend to use the model to classify new, never-before-seen, tweets, in order to conduct their research. My goals are:

1. Create an accurate classifier which can classify **novel tweets** as positive, negative, or neutral.
2. Find out what people are saying about Apple (at South by Southwest, 2011).
3. Make some recommendations based on my findings.

Imports

Because there are so many of them, I've created a separate section.

Standard Library and External

In [1]:

```
import re
import string
from functools import partial
from operator import itemgetter, attrgetter
from os.path import normpath
from typing import Callable

import joblib
import matplotlib.pyplot as plt
import nltk
import numpy as np
import pandas as pd
import seaborn as sns
from gensim.parsing.preprocessing import STOPWORDS
from nltk.collocations import BigramAssocMeasures
from nltk.tokenize.destructive import NLTKWordTokenizer
from nltk.tokenize.treebank import TreebankWordTokenizer
from sklearn.base import clone
from sklearn.compose import (
    ColumnTransformer,
    make_column_selector,
    make_column_transformer,
)
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import (
    CountVectorizer,
    HashingVectorizer,
    TfidfTransformer,
    TfidfVectorizer,
)
```

```python
from sklearn.feature_selection import VarianceThreshold
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import (
    LogisticRegression,
    LogisticRegressionCV,
    PassiveAggressiveClassifier,
    RidgeClassifier,
    RidgeClassifierCV,
    SGDClassifier,
)
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    RepeatedStratifiedKFold,
    StratifiedKFold,
    train_test_split,
)
from sklearn.pipeline import FeatureUnion, Pipeline, make_pipeline
from sklearn.preprocessing import (
    Binarizer,
    FunctionTransformer,
    MaxAbsScaler,
    MinMaxScaler,
    Normalizer,
    PowerTransformer,
    QuantileTransformer,
    RobustScaler,
    StandardScaler,
)

# Set Seaborn theme and default palette
sns.set_theme(font_scale=1.25, style="darkgrid")
sns.set_palette("deep", desat=0.85, color_codes=True)

# Turn on inline plotting
%matplotlib inline

# Load Black auto-formatter
%load_ext nb_black

# Enable automatic reloading
%load_ext autoreload
%autoreload 2
```

My `tools` Package

I put a lot of time and energy into developing my own tools for analysis. It's probably my favorite part of this kind of work, and I (admittedly) tend to get carried away with it. I developed a lot in the `tools.language` module for this project in particular.

Caching

Some computationally expensive functions in `tools.language` implement caching, allowing them to save the results of previous calls and reuse them. This **dramatically increases their performance** when being called over and over again as part of a preprocessing pipeline. Essentially, after the function has been called once with certain parameters, every subsequent call with those parameters is fulfilled instantly. This is a highly non-trivial development, which increases the speed of parameter searches (e.g. with `GridSearchCV`) and makes model development more efficient in general.

Polymorphism

I've designed the text processing functions in `tools.language` to be polymorphic: capable of handling both a single string document and various types of iterables of documents. This level of flexibility is arguably overkill for the present task, but it allows the functions to be easily deployed within Scikit-Learn's `FunctionTransformer` (where they take array input) or as the `TfidfVectorizer.preprocessor` (where they take string input). They can also directly handle Pandas `Series` objects.

VaderVectorizer

Another notable development is the `VaderVectorizer`, which extracts VADER (Valence Aware Dictionary and Sentiment Reasoner) polarity scores from documents and turns them into short vectors of shape (n_samples, 4). This is essentially just a fancy wrapper around the VADER tools from NLTK, which integrates them with the Scikit-Learn API. Nevertheless, it proved very useful for the current project.

In [2]:

```
# Import my modules
from tools import cleaning, plotting, language as lang, utils
from tools.modeling.transformers import POSExtractor
from tools.modeling.vectorizers import VaderVectorizer
from tools.modeling.classification import diagnostics as diag
from tools.modeling.selection import sweep, load_results, load_best_params

# Set my default MPL settings
plt.rcParams.update(plotting.MPL_DEFAULTS)

# RandomState for reproducibility
rando = np.random.RandomState(9547)
```

Overview of Dataset

Since Apple is interested in sentiment analysis on Twitter, I've found a Twitter dataset with crowdsourced sentiment labels. It comes from CrowdFlower, which has released other similar datasets.

The tweets are related to South by Southwest, an annual conference and arts festival in Austin, Texas. They are from 2011, when Apple launched the iPad 2.

It has only three features: the tweet text, the brand object of the sentiment, and the sentiment. It has only about 9,100 tweets.

In [3]:

```
df = pd.read_csv(normpath("data/crowdflower_tweets.csv"))
df.head()
```

Out[3]:

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

The dataset contains one text feature and two categorical features, one of which has a lot of null values. The feature names are very long and wordy, presumably to reflect the actual language used by CrowdFlower in crowdsourcing the dataset. I'm going to rename those before I do anything else.

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column                                              Non-Null Count  Dtype
---  ------                                              --------------  -----
 0   tweet_text                                          9092 non-null   object
 1   emotion_in_tweet_is_directed_at                     3291 non-null   object
 2   is_there_an_emotion_directed_at_a_brand_or_product  9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

Cleaning

Renaming

In [5]:
```
# Assign new column names
df.columns = ["text", "object_of_emotion", "emotion"]
df.head()
```

Out[5]:

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

Next, I take a look at the values of the categorical variables. The categories make sense, although the names are longer than necessary. I'm going to shorten some of them as well.

In [6]:
```
cleaning.show_uniques(df)
```

object_of_emotion	emotion
iPhone	Negative emotion
iPad or iPhone App	Positive emotion
iPad	No emotion toward brand or product
Google	I can't tell
Android	
Apple	
Android App	
Other Google product or service	

object_of_emotion	emotion
Other Apple product or service	

First, I convert the categorical columns to `CategoricalDtype`. This will make it easier to rename the categories, and is a convenient way to differentiate the categorical features from the text column.

In [7]:

```
# Convert categorical columns to categorical dtype
cat_cols = ["emotion", "object_of_emotion"]
df[cat_cols] = df.loc[:, cat_cols].astype("category")

# Delete temp variable
del cat_cols

# Display results
display(df["emotion"].head(3), df["object_of_emotion"].head(3))
```

```
0    Negative emotion
1    Positive emotion
2    Positive emotion
Name: emotion, dtype: category
Categories (4, object): ['I can't tell', 'Negative emotion', 'No emotion toward brand or product',
'Positive emotion']
0                 iPhone
1    iPad or iPhone App
2                   iPad
Name: object_of_emotion, dtype: category
Categories (9, object): ['Android', 'Android App', 'Apple', 'Google', ..., 'Other Google product or
service', 'iPad', 'iPad or iPhone App', 'iPhone']
```

Next, I rename the categories for both categorical features.

I use a single `dict` mapping old category names to new ones. I only need one `dict` for both features because the method `Series.cat.rename_categories(...)` ignores irrelevant keys.

In [8]:

```
# Create mapping of old categories to new ones
new_cats = {
    # New 'emotion' categories
    "Negative emotion": "Negative",
    "Positive emotion": "Positive",
    "No emotion toward brand or product": "Neutral",
    "I can't tell": "Uncertain",
    # New 'object_of_emotion' categories
    "iPad or iPhone App": "iOS App",
    "Other Google product or service": "Other Google Product",
    "Other Apple product or service": "Other Apple Product",
}

# Rename categories in-place (ignores irrelevant keys)
df["emotion"].cat.rename_categories(new_cats, inplace=True)
df["object_of_emotion"].cat.rename_categories(new_cats, inplace=True)

# Delete renaming dict
del new_cats

# Show results
cleaning.show_uniques(df)
```

object_of_emotion	emotion
iPhone	Negative
iOS App	Positive

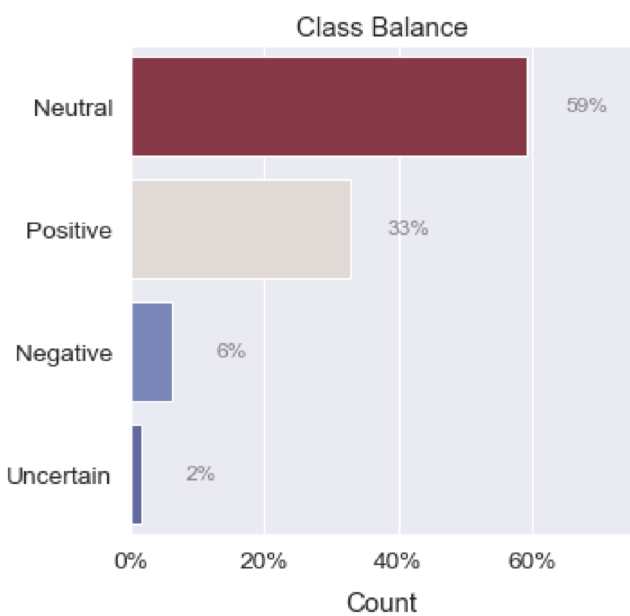
object_of_emotion	emotion
iPad	Neutral
Google	Uncertain
Android	
Apple	
Android App	
Other Google Product	
Other Apple Product	

The 'Neutral' category dominates the distribution, and 'Negative' is very underrepresented. 'Uncertain' is fortunately a very small 2% of the samples. That's good, because it's completely useless to me.

In [9]:

```
ax = plotting.countplot(df["emotion"], normalize=True)
ax.set(title="Class Balance")
ax.set_xlim((0, 0.75))
plotting.save(ax.figure, "images/class_balance.svg")
```

Out[9]: 'images\\class_balance.svg'

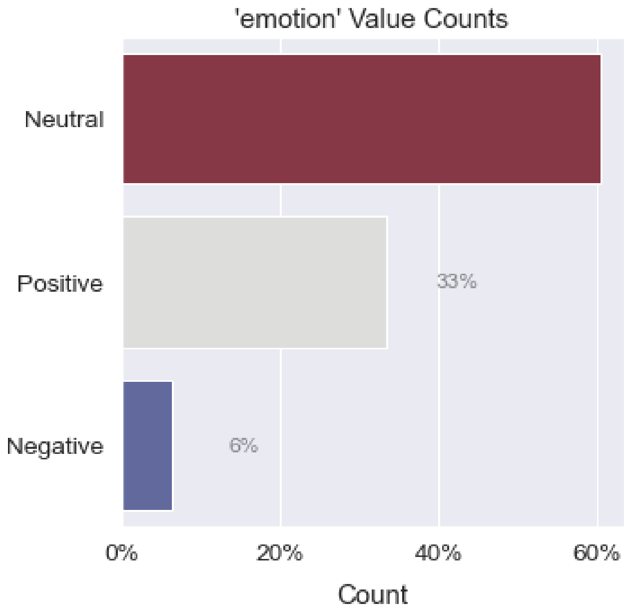


I drop the uncertain category, which doesn't have any clear value. I will have to cope with this imbalance later.

In [10]:

```
# Remove 'Uncertain' category
df.emotion.cat.remove_categories("Uncertain", inplace=True)
plotting.countplot(df.emotion, normalize=True)
```

Out[10]: <AxesSubplot:title={'center':"'emotion' Value Counts"}, xlabel='Count'>



Missing Values

According to the table below, there are a lot of missing values in the 'object_of_emotion' category. I bet, however, that these NaN values correspond to the 'Neutral' category. If a tweet doesn't express a brand-emotion, then there shouldn't be any brand in the 'object_of_emotion' column.

There's also one null 'text' row, and a bunch of null 'emotion' rows where the 'Uncertain' category used to be.

In [11]:

```
cleaning.info(df)
```

Out[11]:

	null	null_%	uniq	uniq_%	dup	dup_%
object_of_emotion	5802	63.81	9	0.10	22	0.24
emotion	156	1.72	3	0.03	22	0.24
text	1	0.01	9065	99.69	22	0.24

I'll go ahead and drop the nulls in the 'text' and 'emotion' columns first.

In [12]:

```
df.dropna(subset=["text", "emotion"], inplace=True)
cleaning.info(df)
```

Out[12]:

	null	null_%	uniq	uniq_%	dup	dup_%
object_of_emotion	5654	63.27	9	0.10	22	0.25
text	0	0.00	8909	99.70	22	0.25
emotion	0	0.00	3	0.03	22	0.25

In [13]:

```
null_rows = cleaning.null_rows(df)
lang.readable_sample(null_rows["text"], random_state=rando)
```

text

	text
5140	RT @mention @mention New iPad Apps For Speech Therapy And Communication Are Showcased At #SXSW Conference {link} #sxswi #hcsm #sxswh
509	Please RT Follow the next big #college social network @mention chance to win an #iPad at 7,000 followers #socialmedia #SXSW
4916	millions of iPhone cases at #SXSW trade show but can any of them double as shuffleboard wax sprinklers? I think not. #fail (CC @mention
6384	RT @mention not launching any products at #SXSW but we're doing plenty else. {link}
790	Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw"
8793	Google giving Social another go? {link} Google Circles, let's see what the guys at #SXSW make of it
8452	@mention The unofficial #SXSW torrents are a great way to hear what you can expect this year {link}
3645	U gotta fight for yr right to party & to privacy ACLU/google #sxsw #partylikeits1986
61	#futuremf @mention {link} spec for recipes on the web, now in google search: {link} #sxsw
4081	Hope people ask the tough questions. RT @mention Reminder: Android and Chrome TTS talk @mention 1 PM today! {link} #sxsw

Looks like some of the NaN values don't line up with the 'Neutral' category. Also, it's important to note that some retweets, e.g. 64, 68, do have sentimental content beyond that of the original tweet.

In [14]:
```
emotion_without_object = null_rows.loc[null_rows.emotion != "Neutral"]

# Delete variable
del null_rows

display(emotion_without_object.head(), emotion_without_object.shape)
```

	text	object_of_emotion	emotion
46	Hand-Held ���Hobo�; Drafthouse launches ���Ho...	NaN	Positive
64	Again? RT @mention Line at the Apple store is ...	NaN	Negative
68	Boooo! RT @mention Flipboard is developing an ...	NaN	Negative
103	Know that "dataviz" translates to &q...	NaN	Negative
112	Spark for #android is up for a #teamandroid aw...	NaN	Positive

```
(357, 3)
```

These are positive tweets which are missing a brand label. Many of them seem positive, some towards a brand and some not. The original features names were 'emotion_in_tweet_is_directed_at' and 'is_there_an_emotion_directed_at_a_brand_or_product', which is not consistent with brandless positivity. But this is data science, and in data science, nothing is consistent.

In [15]:
```
lang.readable_sample(
    emotion_without_object.groupby("emotion").get_group("Positive").text,
    random_state=456,
)
```

	text
6606	RT @mention RT @mention Shiny new @mention @mention @eightbit apps, a new @garyvee book, pop-up iPad 2 stores... #SXSW is Christmas for nerds.

	text
4164	Mad long line for Google party at Maggie Mae's. Hope it's worth it.. but with 80s theme I am very optimistic #sxsw
3020	Apple offers original iPad donation program {link} #entry #friends #house #sxsw
8114	#touchingstories giving us the background to STARTING. Great to hear after yesterday's presos on #uncertainty #iPad and/or #tablet #SXSW
555	I have my golden tickets f 4sq party Day after the real party #Redbullbpm with Felix da Housecat playing on iPad! #SXSW {link}
5501	RT @mention At #sxsw even the cabbies are tech savvy. That's his iPhone streaming twitter. @mention {link}
6676	RT @mention Soundtrckr featured by @mention @mention as a Must-have for #SXSW {link}
157	@mention #SXSW LonelyPlanet Austin guide for #iPhone is free for a limited time {link} #lp #travel
5019	Here he comes ladies! @mention @mention RT @mention I'll be at Austin Convention Center w/ @mention showing my iPhone game. #SXSW
8025	Someone asks Leo about an iPad 2 at #SXSW, he says 'Email me, I'll send you one free'. O.o

Fortunately there aren't very many of them, so not much hangs on my decision to go ahead and fill in the missing brands.

In [16]:

```
# Create regex for finding each brand
re_apple = r"ipad\d?\s*app|ipad\d?|iphone\s*app|iphone|apple"
re_google = r"android\s*app|android|google"

# Find all brand/product name occurrences for each brand
findings = lang.locate_patterns(
    re_apple,
    re_google,
    strings=emotion_without_object["text"],
    exclusive=True,
    flags=re.I,
)

# Convert to lowercase
findings = findings.str.lower()

# View results
display(
    findings.value_counts(),
    findings.size,
)
```

```
google         122
ipad            98
apple           76
iphone          57
ipad2           26
android         19
iphone app       8
ipad app         4
ipad1            1
android app      1
Name: locate_patterns, dtype: int64
412
```

In [17]:

```
# Rename Apple apps to match categories defined previously
findings = findings.str.replace(
    r"ipad\s+app|iphone\s+app", "ios app", case=False, regex=True
)
```

```python
# Fuzzy match with previously defined categories
findings = lang.fuzzy_match(findings, df["object_of_emotion"].cat.categories)

# View results
findings.sort_values("score")
```

Out[17]:

	original	match	score
5401	ipad2	iPad	89
3179	ipad2	iPad	89
8149	ipad2	iPad	89
6309	ipad2	iPad	89
3710	ipad2	iPad	89
...	...	...	...
3224	ipad	iPad	100
3179	ipad	iPad	100
3134	google	Google	100
3055	ipad	iPad	100
9054	ipad	iPad	100

412 rows × 3 columns

In [18]:

```python
# Define sort order, i.e. fill priority
order = [
    "iOS App",
    "Android App",
    "iPhone",
    "iPad",
    "Android",
    "Apple",
    "Google",
]

# Sort values in reverse order
utils.explicit_sort(
    findings,
    order=order,
    by="match",
    ascending=False,
    inplace=True,
)

# Fill in reverse, overwriting lower priority values
for i, brand in findings.match.items():
    df.at[i, "object_of_emotion"] = brand
df.loc[findings.index].sample(10, random_state=rando)
```

Out[18]:

	text	object_of_emotion	emotion
8029	Yeah I wasn't doing it, but I got couldn't res...	iPad	Positive
2753	I love the waves!!!!!! {link} iPad Webber #jap...	iPad	Positive
8973	Google guy at #sxsw talk is explaining how he ...	Google	Negative
1089	���@mention So @mention just spilled the beans...	iPhone	Positive

	text	object_of_emotion	emotion
4674	Apple opening up temporary store in downtown A...	iPad	Positive
4536	Whoa - line for ipad2 is 3blks long!!! #apple ...	iPad	Positive
6078	RT @mention I'm debuting my new iPhone & D...	iPhone	Positive
6710	RT @mention Temporary #apple store is def not ...	Apple	Positive
682	#technews iPad 2 Gets Temporary Apple Store fo...	iPad	Positive
5501	RT @mention At #sxsw even the cabbies are tech...	iPhone	Positive

In [19]:

```
# Get indices which were not filled
emotion_without_object.drop(findings.index, inplace=True)

# Drop unfilled observations
df.drop(emotion_without_object.index, inplace=True)

print(f"{emotion_without_object.shape[0]} observations dropped.")

del emotion_without_object
```

```
24 observations dropped.
```

Here are the tweets which are labeled 'Neutral' but have a brand label, implying that a non-neutral emotion is being expressed towards a brand. Most 'Neutral' tweets do not have a brand label, so these 91 tweets are an anomaly.

In [20]:

```
object_without_emotion = df.loc[
    (df.emotion == "Neutral") & df.object_of_emotion.notnull()
]
display(object_without_emotion.head(), object_without_emotion.shape)
```

	text	object_of_emotion	emotion
63	#Smile RT @mention I think Apple's "pop-u...	Apple	Neutral
265	The #SXSW Apple "pop-up" store was n...	Apple	Neutral
317	I arrived at #sxsw and my @mention issue hasn'...	iOS App	Neutral
558	haha. the google "Party like it's 1986&qu...	Google	Neutral
588	Diller on Google TV: "The first product w...	Other Google Product	Neutral

```
(91, 3)
```

Tweet 6517 seems clearly negative to me, and 7137 seems kind of sardonic. 2666 seems weakly positive. 8647, 5696, 7521, 668, and 265 don't seem to express an emotion toward a brand or product. Since most of them seem neutral to me, and that's consistent with their 'Neutral' label, I'm going to keep them that way.

In [21]:

```
lang.readable_sample(object_without_emotion["text"], random_state=rando)
```

	text
668	#sxsw guy in front of me at this panel has an ipad in an etch-a-sketch case...device of wondeR? #iusxsw
1628	@mention @mention Similarily, Tweetcaster for Android lets you zip tweets w annoying hash tags, like #sxsw
1253	Google vp to speak. The topic: 10 quick steps to owning everything in the world. #sxsw {link}

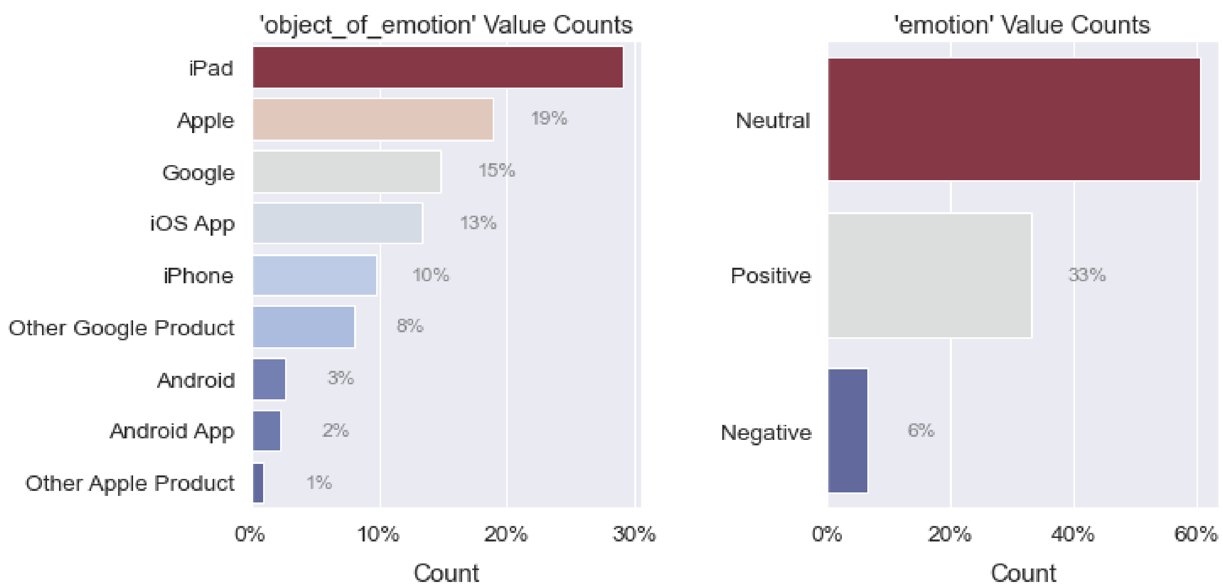
	text
2849	Nice to see the speaker sneak in an irrelevant snarky comment about Apple. Class! #sxsw #authenticationdesign
7658	Score a free imo tshirt outside the SXSW Apple store today at 2:15 PM & check out imo's app for the iPad 2 {link} #sxsw #ipad2
4119	From #Apple to Naomi Campbell: pop-up stores are all the rage: {link} #sxsw
5912	RT @mention Google to launch new social network at SXSW? - CNET News {link} #sxsw
6082	RT @mention I'm not really at #sxsw. Just messing with you. I'm making money instead. // I bet someone left the iPad queue
6491	RT @mention RT @mention "IAVA wants to be the Google of nonprofits." / yes, we do b/c our #vets deserve nothing less! #sxsw #letshookup
8902	@mention Which is to say iPad is going to be ubiquitous a lot faster than anyone expected a year or even 6 mo. ago. #newsapps #sxsw

In [22]:
```
# Set object to null where emotion is neutral
df.loc[object_without_emotion.index, "object_of_emotion"] = np.nan

# Ensure that 'Neutral' rows line up with 'NaN' rows
(df["emotion"] == "Neutral").equals(df["object_of_emotion"].isnull())
```

Out[22]:
```
True
```

Here's a look at the final distributions.

In [23]:
```
fig = plotting.countplot(df.select_dtypes("category"), normalize=1)
```



Duplicates

There are 22 duplicate rows, and even more when only the text is considered. I don't want to get rid of all retweets, but I do want to get rid of those which don't have novel content.

In [24]:
```
cleaning.dup_rows(df.text).sort_values()
```

```
Out[24]: 3962    #SXSW is just starting, #CTIA is around the co...
468        Before It Even Begins, Apple Wins #SXSW {link}
2559    Counting down the days to #sxsw plus strong Ca...
776     Google to Launch Major New Social Network Call...
8483    I just noticed DST is coming this weekend. How...
2232    Marissa Mayer: Google Will Connect the Digital...
8747    Need to buy an iPad2 while I'm in Austin at #s...
4897    Oh. My. God. The #SXSW app for iPad is pure, u...
5882    RT @mention Google to Launch Major New Social ...
5884    RT @mention Google to Launch Major New Social ...
5883    RT @mention Google to Launch Major New Social ...
5881    RT @mention Google to Launch Major New Social ...
5885    RT @mention Google to Launch Major New Social ...
6299    RT @mention Marissa Mayer: Google Will Connect...
6297    RT @mention Marissa Mayer: Google Will Connect...
6295    RT @mention Marissa Mayer: Google Will Connect...
6300    RT @mention Marissa Mayer: Google Will Connect...
6298    RT @mention Marissa Mayer: Google Will Connect...
6294    RT @mention Marissa Mayer: Google Will Connect...
6296    RT @mention Marissa Mayer: Google Will Connect...
6546    RT @mention RT @mention Google to Launch Major...
6576    RT @mention RT @mention It's not a rumor: Appl...
5338    RT @mention ��� GO BEYOND BORDERS! ��_ {link} ...
5341    RT @mention ��� Happy Woman's Day! Make love, ...
3950    Really enjoying the changes in Gowalla 3.0 for...
3814                 Win free iPad 2 from webdoc.com #sxsw RT
3813                 Win free ipad 2 from webdoc.com #sxsw RT
Name: text, dtype: object
```

I filter the text by removing occurrences of 'RT' and then check for duplicates. This should get rid of retweets which are just copies of original tweets in the dataset.

```
In [25]:
dups = df.text.str.replace(r"\s*RT\s*", "", regex=True).duplicated()
df = df.loc[~dups]
dups.sum()
```

```
Out[25]: 33
```

Feature Engineering

I get organized with text-processing functions and stopword sets, and then engineer some features.

Stopword Sets

```
In [26]:
my_stop = {
    "sxsw",
    "quot",
    "link",
    "austin",
    "mention",
    "sxswi",
    "america",
    "southbysouthwest",
}
brand_stop = {
    "apple",
    "applesxsw",
    "google",
    "android",
    "andoid",
    "app",
    "ipad",
    "iphone",
```

```
    "androidsxsw",
}
gensim_stop = set(STOPWORDS)
nltk_stop = set(nltk.corpus.stopwords.words("english"))
pd.Series(list(gensim_stop))
```

Out[26]:

```
0       system
1        along
2         four
3         thru
4          are
        ...
332     anyone
333        not
334      those
335     enough
336   besides
Length: 337, dtype: object
```

In [27]:

```
pd.Series(
    dict(
        my_stop=my_stop,
        brand_stop=brand_stop,
        gensim_stop=gensim_stop,
        nltk_stop=nltk_stop,
    )
).to_json(normpath("data/stopwords.json"))
```

Text-Processing Functions

Most of these functions from my `tools.language` module are simple polymorphic wrappers around functions from `gensim.parsing.preprocessing` or `nltk`.

Probably the most complex is `wordnet_lemmatize`, which must tokenize text, tag parts of speech, translate the tags, lemmatize the tagged tokens, and then finally detokenize.

In [28]:

```
funcs = [
    lang.lowercase,
    lang.strip_short,
    lang.strip_punct,
    lang.strip_multiwhite,
    lang.strip_numeric,
    lang.strip_non_alphanum,
    lang.split_alphanum,
    lang.uni2ascii,
    lang.stem_text,
    lang.strip_handles,
    lang.limit_repeats,
    lang.wordnet_lemmatize,
    lang.stem_text,
    lang.mark_pos,
    lang.mark_negation,
]

func_names = utils.get_func_name(funcs)
funcs = pd.Series(dict(zip(func_names, funcs)))
funcs
```

Out[28]:

```
lowercase                     <function lowercase at 0x0000021557CD1670>
strip_short                 <function strip_short at 0x0000021557CD1700>
strip_punct                 <function strip_punct at 0x000002155D323040>
strip_multiwhite        <function strip_multiwhite at 0x0000021557CD1790>
strip_numeric             <function strip_numeric at 0x0000021557CD1820>
```

```
strip_non_alphanum    <function strip_non_alphanum at 0x0000021557CD...
split_alphanum          <function split_alphanum at 0x0000021557CD1940>
uni2ascii                   <function uni2ascii at 0x0000021557CD1F70>
stem_text                   <function stem_text at 0x0000021557CD1D30>
strip_handles           <function strip_handles at 0x0000021557CD1DC0>
limit_repeats           <function limit_repeats at 0x0000021557CD19D0>
wordnet_lemmatize     <function wordnet_lemmatize at 0x000002155D325...
mark_pos                     <function mark_pos at 0x000002155D325430>
mark_negation           <function mark_negation at 0x000002155D325820>
dtype: object
```

In [29]:

```
filts = [
    "lowercase",
    "strip_handles",
    "uni2ascii",
    "wordnet_lemmatize",
    "strip_punct",
    "strip_numeric",
    "strip_short",
    "limit_repeats",
    "strip_multiwhite",
]

filts = lang.make_preprocessor(funcs.loc[filts].to_list())
filts
```

Out[29]:

```
functools.partial(<function chain_funcs at 0x000002155D323A60>, funcs=[<function lowercase at 0x000
0021557CD1670>, <function strip_handles at 0x0000021557CD1DC0>, <function uni2ascii at 0x0000021557
CD1F70>, <function wordnet_lemmatize at 0x000002155D325C10>, <function strip_punct at 0x000002155D3
23040>, <function strip_numeric at 0x0000021557CD1820>, <function strip_short at 0x0000021557CD1700
>, <function limit_repeats at 0x0000021557CD19D0>, <function strip_multiwhite at 0x0000021557CD1790
>])
```

In [30]:

```
df["clean_text"] = filts(df["text"])
df["clean_text"] = lang.strip_stopwords(df["clean_text"], my_stop)
df.clean_text.head()
```

Out[30]:

```
0    have iphone after tweet rise dead need upgrade...
1    know about awesome ipad iphone app that you li...
2    cannot wait for ipad also they should sale the...
3    hope this year festival ben crashy this year i...
4    great stuff fri marissa mayer google tim reill...
Name: clean_text, dtype: object
```

Brand Terms

I extract brand terms based on the crowdsourced labels using regular expressions. I'm comfortable using these for training the model, since they were extracted algorithmically.

In [31]:

```
re_brand = fr"{re_apple}|{re_google}"
regex_brands = lang.locate_patterns(re_brand, strings=df.clean_text)
regex_brands = utils.implode(regex_brands).reindex_like(df)
df["brand_terms"] = regex_brands
del regex_brands
df["brand_terms"].head()
```

Out[31]:

```
0               [iphone]
1    [ipad, iphone app]
2                 [ipad]
3           [iphone app]
4               [google]
Name: brand_terms, dtype: object
```

Parts of Speech

I use `nltk.casual_tokenize` a.k.a. `nltk.TweetTokenizer` to extract tokens from the raw text for PoS tagging. This tokenizer is able to capture '@mentions' and 'hashtags' without chopping them up.

In [32]:

```
# Tokenize and tag using TweetTokenizer
df["tagged"] = lang.tokenize_tag(df["text"], tokenizer=nltk.casual_tokenize)

# My `lang.tokenize_tag` function tokenizes and tags POS all at once, which
# is often convenient, since tagging POS requires tokenization.

df["tagged"].head()
```

Out[32]:

```
0    [(., .), (@wesley83, NN), (I, PRP), (have, VBP...
1    [(@jessedee, NN), (Know, NNP), (about, IN), (@...
2    [(@swonderlin, NNS), (Can, MD), (not, RB), (wa...
3    [(@sxsw, NN), (I, PRP), (hope, VBP), (this, DT...
4    [(@sxtxstate, JJ), (great, JJ), (stuff, NN), (...
Name: tagged, dtype: object
```

I extract just the tags in hopes of analyzing them like words. They can be easily vectorized with `CountVectorizer` or `TfidfVectorizer` if converted to `str`.

Future work: Create a PoS extractor/vectorizer which is integrated with the Scikit-Learn API and can be tuned alongside other feature-extractors.

In [33]:

```
# Explode the lists of tagged-tokens, get the tag, and then "implode"
df["pos_tags"] = utils.implode(df["tagged"].explode().map(itemgetter(1), "ignore"))

# My `utils.implode` function retracts a long-form Series into one
# with unique indices and lists as values.

df["pos_tags"].head()
```

Out[33]:

```
0    [., NN, PRP, VBP, DT, CD, NN, ., IN, CD, NN, N...
1    [NN, NNP, IN, NNP, ., NNP, NN, NNP, NN, NN, WD...
2    [NNS, MD, RB, VB, IN, JJ, CD, RB, ., PRP, MD, ...
3    [NN, PRP, VBP, DT, NN, NN, NN, RB, JJ, IN, DT,...
4    [JJ, JJ, NN, IN, NNP, NN, :, NNP, NNP, (, NNP,...
Name: pos_tags, dtype: object
```

Simple Counts

I engineer character counts (minus spaces), word counts, and average word lengths. Maybe an interesting pattern will show up.

In [34]:

```
# String length without whitespace
df["n_chars"] = df["text"].str.replace("\s+", "", regex=True).map(len)

# Number of words as parsed by TweetTokenizer
df["n_words"] = df["text"].map(nltk.casual_tokenize).map(len)

# Calculate average word length
df["avg_word_len"] = df["n_chars"] / df["n_words"]

# Show results
df[["n_chars", "n_words", "avg_word_len"]].head()
```

Out[34]:

	n_chars	n_words	avg_word_len
0	104	29	3.586207
1	118	26	4.538462
2	65	17	3.823529
3	68	16	4.250000
4	115	27	4.259259

I engineer exclamation point and question mark counts, which I've discovered have a surprisingly robust connection to sentiment.

In [35]:

```
df["ep_count"] = df["text"].str.count(r"\!")
df["qm_count"] = df["text"].str.count(r"\?")
df[["ep_count", "qm_count"]].head()
```

Out[35]:

	ep_count	qm_count
0	1	0
1	0	1
2	0	0
3	0	0
4	0	0

In [36]:

```
df.to_json(normpath("data/processed_tweets.json"))
```

Modeling

In [37]:

```
df["brand_terms"] = utils.implode(
    df["brand_terms"]
    .explode()
    .str.replace(" ", "_")
    .str.replace(r"([a-zA-Z]+)app", lambda x: f"{x[1]}_app", regex=True)
    .fillna("")
)

df["brand_terms"].head()
```

Out[37]:

```
0               [iphone]
1    [ipad, iphone_app]
2                 [ipad]
3           [iphone_app]
4               [google]
Name: brand_terms, dtype: object
```

In [38]:

```
df.brand_terms.explode().unique()
```

Out[38]:

```
array(['iphone', 'ipad', 'iphone_app', 'google', 'ipad_app', 'android',
       'apple', 'android_app', ''], dtype=object)
```

In [39]:
```
df["brand_terms"] = df["brand_terms"].str.join(" ")
df["brand_terms"].head()
```

Out[39]:
```
0              iphone
1    ipad iphone_app
2                ipad
3          iphone_app
4              google
Name: brand_terms, dtype: object
```

In [40]:
```
df["pos_tags"] = df["pos_tags"].str.join(" ")
df["pos_tags"].head()
```

Out[40]:
```
0    . NN PRP VBP DT CD NN . IN CD NN NN IN NNP , P...
1    NN NNP IN NNP . NNP NN NNP NN NN WDT VBZ JJ NN...
2    NNS MD RB VB IN JJ CD RB . PRP MD NN PRP RP IN...
3     NN PRP VBP DT NN NN NN RB JJ IN DT NN NN NN . NN
4    JJ JJ NN IN NNP NN : NNP NNP ( NNP ) , NNP NNP...
Name: pos_tags, dtype: object
```

Train-Test-Split

In [41]:
```
cols = [
    "text",
    "brand_terms",
    "pos_tags",
    "n_chars",
    "n_words",
    "avg_word_len",
    "ep_count",
    "qm_count",
]
X = df.loc[:, cols].copy()
y = df.emotion.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    random_state=rando,
    stratify=y,
    shuffle=True,
)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[41]:
```
((6659, 8), (6659,), (2220, 8), (2220,))
```

Baseline Dummy Model

I create a `ColumnTransformer` to process the data of different columns and concatenate the results. As you can see, I set `remainder="drop"`, which means that only 'text' and 'brand_terms' are being used. The other engineered features turn out not to have much of an impact, and only make it difficult to get feature names for the model coefficients. The 'ep_count' and 'qm_count' features are essentially created by `TfidfVectorizer`, anyway.

In [42]:
```
col_xform = ColumnTransformer(
    [
        ("txt", TfidfVectorizer(tokenizer=nltk.casual_tokenize), "text"),
        (
            "bra",
            CountVectorizer(tokenizer=lang.space_tokenize, binary=True),
```

```python
            "brand_terms",
        ),
    ],
    remainder="drop",
)
col_xform
```

Out[42]:
```
ColumnTransformer(transformers=[('txt',
                                 TfidfVectorizer(tokenizer=<function casual_tokenize at 0x000002155
3A0FE50>),
                                 'text'),
                                ('bra',
                                 CountVectorizer(binary=True,
                                                 tokenizer=<function space_tokenize at 0x000002155D
323CA0>),
                                 'brand_terms')])
```

In [43]:
```python
main_pipe = Pipeline(
    [
        ("col", col_xform),
        ("cls", DummyClassifier(strategy="stratified", random_state=rando)),
    ]
)
main_pipe
```

Out[43]:
```
Pipeline(steps=[('col',
                 ColumnTransformer(transformers=[('txt',
                                                  TfidfVectorizer(tokenizer=<function casual_tokeni
ze at 0x0000021553A0FE50>),
                                                  'text'),
                                                 ('bra',
                                                  CountVectorizer(binary=True,
                                                                  tokenizer=<function space_tokeniz
e at 0x000002155D323CA0>),
                                                  'brand_terms')])),
                ('cls',
                 DummyClassifier(random_state=RandomState(MT19937) at 0x2155D4CA240,
                                 strategy='stratified'))])
```

In [44]:
```python
vecs = main_pipe[:1].fit_transform(X)
vecs
```

Out[44]:
```
<8879x10595 sparse matrix of type '<class 'numpy.float64'>'
	with 184853 stored elements in Compressed Sparse Row format>
```

In [45]:
```python
vecs.todense()
```

Out[45]:
```
matrix([[0.08833442, 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         1.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ],
        ...,
        [0.        , 0.20527898, 0.        , ..., 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ]])
```

What is `partial`?

You'll see me use `functools.partial` pretty often. It creates a thin wrapper over an existing function with some preset arguments held in place. It's like creating a wrapper with new default parameters, except that `partial` can hold positional arguments in place too. It comes in handy in many situations. In this case, I'm using it to wrap `diagnostics.test_fit` and hold `X_train`, `X_test`, `y_train`, and `y_test` in place.

In [46]:

```python
test_fit = partial(
    diag.test_fit,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
)
test_fit
```

Out[46]:

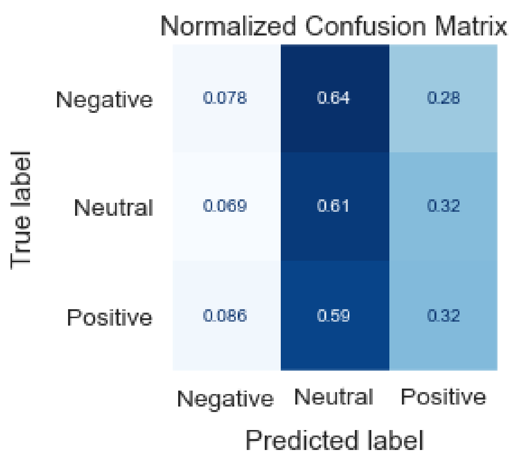
```
functools.partial(<function test_fit at 0x000002155D4ACCA0>, X_train=
text        brand_terms  \
7610  #foursquare and @mention now sees #google as t...            google
4742  Ten percent of the crowd at &quot;Designing iP...         ipad ipad
7077  Apple to Open Pop-Up Shop at SXSW [REPORT]: {l...             apple
8700  We interrupt your regularly-scheduled #sxsw ge...            google
7604  apple store #sxsw line is moving at the front....             apple
...                                                 ...               ...
1249  Expecting to see a flood of shiny new ipad2's ...        ipad apple
4165  First in line this morning at the Apple pop-up...             apple
4052  SXSW panel: Google envisions search without se...            google
4501  line around the corner for #iPad2 at #sxsw, i ...  ipad iphone apple
1962  Holler Gram for iPad on the iTunes App Store -...              ipad

                                               pos_tags  n_chars  n_words  \
7610  NN CC NN RB VBZ NNS IN PRP$ NN RB RB JJ CC NN ...      100       18
4742  CD NN IN DT NN IN NNP NNP NN NNP NNP RB VBP DT...       82       18
7077     NNP TO VB NNP NNP IN NNP NNP NNP NNP ( NN ) NN       47       14
8700  PRP VBP PRP$ JJ JJ NN NN IN JJ NN . ( NN ) IN ...       97       18
7604  NN NN JJ NN VBZ VBG IN DT NN . DT NN NN VB JJR...       75       19
...                                                 ...      ...      ...
1249  VBG TO VB DT NN IN JJ JJ NN CD '' NN NN NNP NN...       88       23
4165  RB IN NN DT NN IN DT NNP NN NN IN NNP , NN VBD...       99       23
4052          JJ NN : NNP NNS NN IN NNS : ( VB ) NNP NN       64       14
4501  NN IN DT NN IN NN IN NNP , NNS VBP VBP IN PRP ...       95       25
1962  NNP NNP IN NN IN DT NNS NNP NNP : ( NN ) ( IN ...       60       17

      avg_word_len  ep_count  qm_count
7610      5.555556         0         0
4742      4.555556         0         0
7077      3.357143         0         0
8700      5.388889         1         0
7604      3.947368         1         0
...            ...       ...       ...
1249      3.826087         0         0
4165      4.304348         0         0
4052      4.571429         0         0
4501      3.800000         0         0
1962      3.529412         0         0

[6659 rows x 8 columns], X_test=                                                text    brand_te
rms  \
3417  @mention has arrived at the #GSDM &amp; #Googl...        google
6112  RT @mention If you're in a room full of people...       android
514   Wish list for tech?  #NTN #SXSW Google Apps he...        google
7198  @mention ouch. Looks like I might be able to g...         apple
8039  @mention so... when is the Apple &quot;pop-up&...         apple
...                                                 ...           ...
2510  I just made the decision: Id like to purchase ...          ipad
1174  Have both the phones in the @mention demo been...  google iphone
1436  Anonymity: Zuckerberg &quot;wrong&quot; says 4...        google
6783  So @mention thinks I may have coined a new ter...    iphone_app
1682  Google's #geosocial Offers platform goes live ...        google
```

```
                                            pos_tags  n_chars  n_words  \
3417              NN VBZ VBN IN DT NNP CC NNP NNP NN NN .       59       12
6112     NNP NN IN VBN IN DT NN JJ IN NNS VBP JJ JJ NN ...      114       33
514      JJ NN IN NN . JJ NN NNP NNP VBZ VB DT NN NN IN...       94       23
7198     NN RB . VBZ IN PRP MD VB JJ TO VB CD IN DT NNP...       67       18
8039     NN RB : WRB VBZ DT NNP NNP JJ NNP NN NN . CC I...      121       29
...                                                    ...      ...      ...
2510     PRP RB VBD DT NN : NN IN TO VB DT NN . PRP MD ...       73       22
1174     VB DT DT NNS IN DT NN NN VBN DT NNP NNS . CC V...       85       23
1436     NN : NNP NNP JJ NNP VBZ CD '' JJ NNP NNP ( NN ...      111       23
6783     RB JJ NNS PRP MD VB VBN DT JJ NN : JJ NN . VB ...      124       29
1682                        NNP JJ NNP NN VBZ JJ IN NNP ( NN )       53       11

      avg_word_len  ep_count  qm_count
3417      4.916667         0         0
6112      3.454545         0         0
514       4.086957         0         1
7198      3.722222         0         0
8039      4.172414         0         2
...            ...       ...       ...
2510      3.318182         0         0
1174      3.695652         0         2
1436      4.826087         0         1
6783      4.275862         0         0
1682      4.818182         0         0

[2220 rows x 8 columns], y_train=array(['Neutral', 'Neutral', 'Neutral', ..., 'Neutral', 'Negativ
e',
       'Positive'], dtype=object), y_test=array(['Neutral', 'Neutral', 'Positive', ..., 'Neutral',
'Neutral',
       'Neutral'], dtype=object))
```

In [47]:

```
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.065	0.609	0.324	0.332	0.480	0.471	0.332
recall	0.071	0.582	0.344	0.332	0.471		
f1-score	0.068	0.595	0.334	0.332	0.475		
support	0.064	0.605	0.332				



Baseline Model

In [48]:

```
logit = LogisticRegression(
    class_weight="balanced",
    multi_class="multinomial",
```

```
    solver="lbfgs",
    max_iter=1e4,
    verbose=0,
    random_state=rando,
)
logit
```

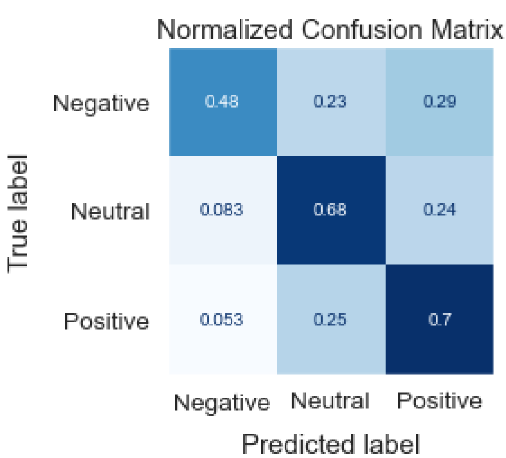
Out[48]:
```
LogisticRegression(class_weight='balanced', max_iter=10000.0,
                   multi_class='multinomial',
                   random_state=RandomState(MT19937) at 0x2155D4CA240)
```

In [49]:
```
main_pipe.set_params(cls=logit)
```

Out[49]:
```
Pipeline(steps=[('col',
                 ColumnTransformer(transformers=[('txt',
                                                  TfidfVectorizer(tokenizer=<function casual_tokeni
ze at 0x0000021553A0FE50>),
                                                  'text'),
                                                 ('bra',
                                                  CountVectorizer(binary=True,
                                                                  tokenizer=<function space_tokeniz
e at 0x000002155D323CA0>),
                                                  'brand_terms')])),
                ('cls',
                 LogisticRegression(class_weight='balanced', max_iter=10000.0,
                                    multi_class='multinomial',
                                    random_state=RandomState(MT19937) at 0x2155D4CA240))])
```

In [50]:
```
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.307	0.810	0.589	0.569	0.705	0.673	0.618
recall	0.475	0.679	0.701	0.618	0.673		
f1-score	0.373	0.739	0.640	0.584	0.683		
support	0.064	0.605	0.332				



Select Tokenizer

In [51]:
```
run_sweep = partial(
    sweep,
    X=X,
    y=y,
```

```python
    scoring="balanced_accuracy",
    n_jobs=-1,
    cv=StratifiedKFold(),
    random_state=46,
)
run_sweep
```

Out[51]:

```
functools.partial(<function sweep at 0x000002155D4BB160>, X=
text      brand_terms  \
0     .@wesley83 I have a 3G iPhone. After 3 hrs twe...           iphone
1     @jessedee Know about @fludapp ? Awesome iPad/i...  ipad iphone_app
2     @swonderlin Can not wait for #iPad 2 also. The...             ipad
3     @sxsw I hope this year's festival isn't as cra...       iphone_app
4     @sxtxstate great stuff on Fri #SXSW: Marissa M...           google
...                                                 ...              ...
9088                      Ipad everywhere. #SXSW {link}             ipad
9089  Wave, buzz... RT @mention We interrupt your re...           google
9090  Google's Zeiger, a physician never reported po...           google
9091  Some Verizon iPhone customers complained their...           iphone
9092  �?������_��υ�??й��������V_���������_���RT @mentio...           google

                                               pos_tags  n_chars  n_words  \
0     . NN PRP VBP DT CD NN . IN CD NN NN IN NNP , P...      104       29
1     NN NNP IN NNP . NNP NN NNP NN NN WDT VBZ JJ NN...      118       26
2     NNS MD RB VB IN JJ CD RB . PRP MD NN PRP RP IN...       65       17
3      NN PRP VBP DT NN NN NN RB JJ IN DT NN NN NN . NN       68       16
4     JJ JJ NN IN NNP NN : NNP NNP ( NNP ) , NNP NNP...      115       27
...                                                 ...      ...      ...
9088                               NNP RB . VB ( NN )       26        7
9089  NNP , NN : NNP NN PRP VBP PRP$ RB VBN JJ NN NN...      107       22
9090  NNP NNP , DT NN RB VBD JJ NNP . CC NNP NNS IN ...      127       27
9091  DT NNP NN NNS VBD PRP$ NN VBD RB DT NN DT NN ....      117       25
9092  JJ NNP NNP NNP NNP NNP NNP NNP NNP NNP NNP NNP...       89       41

      avg_word_len  ep_count  qm_count
0         3.586207         1         0
1         4.538462         0         1
2         3.823529         0         0
3         4.250000         0         0
4         4.259259         0         0
...            ...       ...       ...
9088      3.714286         0         0
9089      4.863636         0         0
9090      4.703704         0         0
9091      4.680000         0         0
9092      2.170732         0         0

[8879 rows x 8 columns], y=array(['Negative', 'Positive', 'Positive', ..., 'Neutral', 'Neutral',
       'Neutral'], dtype=object), scoring='balanced_accuracy', n_jobs=-1, cv=StratifiedKFold(n_spli
ts=5, random_state=None, shuffle=False), random_state=46)
```

In [52]:

```python
tok_grid = pd.Series(
    dict(
        tokenizer=[
            nltk.casual_tokenize,
            nltk.word_tokenize,
            nltk.wordpunct_tokenize,
            lang.space_tokenize,
            TreebankWordTokenizer().tokenize,
            NLTKWordTokenizer().tokenize,
            None,
        ]
    )
)

# run_sweep(
#     main_pipe,
#     tok_grid.add_prefix("col__txt__"),
```

```
#     dst="sweeps/tokenizer",
#     kind="grid",
# )
```

In [53]:

```
results = load_results("sweeps/tokenizer.joblib")
results.tokenizer = results.tokenizer.map(str).map(lang.strip_punct)
results.style.bar("mean_score")
```

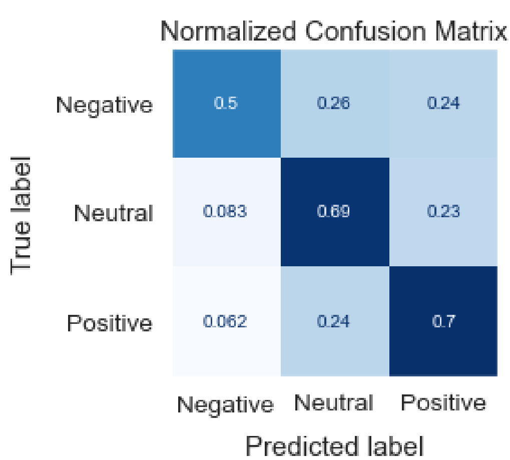
Out[53]:

	tokenizer	mean_score	rank_score
2	bound method RegexpTokenizer tokenize of WordPunctTokenizer pattern w w s gaps False discard empty True flags re UNICODE re MULTILINE re DOTALL	0.636942	1
1	function word tokenize at 0x0000021553AB13A0	0.634967	2
0	function casual tokenize at 0x0000021553A0FE50	0.625265	3
6	None	0.624929	4
4	bound method TreebankWordTokenizer tokenize of nltk tokenize treebank TreebankWordTokenizer object at 0x00000215641B9F40	0.621358	5
5	bound method NLTKWordTokenizer tokenize of nltk tokenize destructive NLTKWordTokenizer object at 0x00000215641B9FD0	0.621224	6
3	function space tokenize at 0x000002155D323CA0	0.613024	7

In [54]:

```
main_pipe.set_params(col__txt__tokenizer=nltk.wordpunct_tokenize)
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.310	0.811	0.597	0.573	0.709	0.677	0.628
recall	0.504	0.685	0.696	0.628	0.677		
f1-score	0.384	0.743	0.643	0.590	0.687		
support	0.064	0.605	0.332				



Select Filters

Create Text Pipeline

In [55]:

```
txt_pipe = Pipeline(
```

```
    [
        ("uni", "passthrough"),
        ("low", "passthrough"),
        ("shrt", "passthrough"),
        ("usr", "passthrough"),
        ("num", "passthrough"),
        ("rep", "passthrough"),
        ("punc", "passthrough"),
        ("wht", "passthrough"),
        ("stem", "passthrough"),
        ("mark", "passthrough"),
        ("vec", clone(col_xform.named_transformers_["txt"])),
    ]
)
txt_pipe
```

Out[55]:
```
Pipeline(steps=[('uni', 'passthrough'), ('low', 'passthrough'),
                ('shrt', 'passthrough'), ('usr', 'passthrough'),
                ('num', 'passthrough'), ('rep', 'passthrough'),
                ('punc', 'passthrough'), ('wht', 'passthrough'),
                ('stem', 'passthrough'), ('mark', 'passthrough'),
                ('vec',
                 TfidfVectorizer(tokenizer=<bound method RegexpTokenizer.tokenize of WordPunctToken
izer(pattern='\\w+|[^\\w\\s]+', gaps=False, discard_empty=True, flags=re.UNICODE|re.MULTILINE|re.DO
TALL)>))])
```

In [56]:
```
col_xform.transformers[0] = ("txt", txt_pipe, "text")
col_xform
```

Out[56]:
```
ColumnTransformer(transformers=[('txt',
                                 Pipeline(steps=[('uni', 'passthrough'),
                                                 ('low', 'passthrough'),
                                                 ('shrt', 'passthrough'),
                                                 ('usr', 'passthrough'),
                                                 ('num', 'passthrough'),
                                                 ('rep', 'passthrough'),
                                                 ('punc', 'passthrough'),
                                                 ('wht', 'passthrough'),
                                                 ('stem', 'passthrough'),
                                                 ('mark', 'passthrough'),
                                                 ('vec',
                                                  TfidfVectorizer(tokenizer=<bound method RegexpTok
enizer.tokenize of WordPunctTokenizer(pattern='\\w+|[^\\w\\s]+', gaps=False, discard_empty=True, fl
ags=re.UNICODE|re.MULTILINE|re.DOTALL)>))]),
                                 'text'),
                                ('bra',
                                 CountVectorizer(binary=True,
                                                 tokenizer=<function space_tokenize at 0x000002155D
323CA0>),
                                 'brand_terms')])
```

Test Punctuation

In [57]:
```
# Create a FunctionTransformer for each symbol in `string.punctuation`
excludes = [FunctionTransformer(lang.strip_punct)]
for x in string.punctuation:
    kw_args = {"exclude": x}
    excludes.append(FunctionTransformer(lang.strip_punct, kw_args=kw_args))

excludes[:10]
```

Out[57]:
```
[FunctionTransformer(func=<function strip_punct at 0x000002155D323040>),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                     kw_args={'exclude': '!'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
```

```
                            kw_args={'exclude': '"'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': '#'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': '$'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': '%'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': '&'}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': "'"}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': '('}),
 FunctionTransformer(func=<function strip_punct at 0x000002155D323040>,
                            kw_args={'exclude': ')'})]
```

In [58]:

```python
grid = {"col__txt__punc": excludes}
# run_sweep(main_pipe, grid, dst="sweeps/punctuation", kind="grid")
```

'!' is the punctuation mark that really stands out. There are many others which are above the baseline (no exclusions), but excluding only exclamation point results in by far the best score.

In [59]:

```python
results = load_results("sweeps/punctuation")
results.head(10).style.bar("mean_score")
```

Out[59]:

	punc	mean_score	rank_score
1	strip_punct(exclude='!')	0.631135	1
21	strip_punct(exclude='?')	0.625472	2
3	strip_punct(exclude='#')	0.625104	3
14	strip_punct(exclude='.')	0.622870	4
4	strip_punct(exclude='$')	0.622077	5
29	strip_punct(exclude='{')	0.621936	6
32	strip_punct(exclude='~')	0.621905	7
31	strip_punct(exclude='}')	0.621874	8
28	strip_punct(exclude='`')	0.621769	9
20	strip_punct(exclude='>')	0.621769	9

In [60]:

```python
funcs = funcs.map(lambda x: FunctionTransformer(x) if isinstance(x, Callable) else x)
funcs["strip_punct"].kw_args = dict(exclude="!")
funcs
```

Out[60]:

```
lowercase            FunctionTransformer(func=<function lowercase a...
strip_short          FunctionTransformer(func=<function strip_short...
strip_punct          FunctionTransformer(func=<function strip_punct...
strip_multiwhite     FunctionTransformer(func=<function strip_multi...
strip_numeric        FunctionTransformer(func=<function strip_numer...
strip_non_alphanum   FunctionTransformer(func=<function strip_non_a...
split_alphanum       FunctionTransformer(func=<function split_alpha...
uni2ascii            FunctionTransformer(func=<function uni2ascii a...
stem_text            FunctionTransformer(func=<function stem_text a...
strip_handles        FunctionTransformer(func=<function strip_handl...
limit_repeats        FunctionTransformer(func=<function limit_repea...
wordnet_lemmatize    FunctionTransformer(func=<function wordnet_lem...
mark_pos             FunctionTransformer(func=<function mark_pos at...
```

```
mark_negation         FunctionTransformer(func=<function mark_negati...
dtype: object
```

Select Filters

In [61]:

```python
filt_grid = pd.Series(txt_pipe[:-3].named_steps).map(lambda x: [x])
filt_grid["uni"] += [funcs.uni2ascii]
filt_grid["low"] += [funcs.lowercase]
filt_grid["shrt"] += [funcs.strip_short]
filt_grid["usr"] += [funcs.strip_handles]
filt_grid["num"] += [funcs.strip_numeric]
filt_grid["rep"] += [funcs.limit_repeats]
filt_grid["punc"] += [funcs.strip_punct]
filt_grid["wht"] += [funcs.strip_multiwhite]

filt_grid
```

Out[61]:

```
uni     [passthrough, FunctionTransformer(func=<functi...
low     [passthrough, FunctionTransformer(func=<functi...
shrt    [passthrough, FunctionTransformer(func=<functi...
usr     [passthrough, FunctionTransformer(func=<functi...
num     [passthrough, FunctionTransformer(func=<functi...
rep     [passthrough, FunctionTransformer(func=<functi...
punc    [passthrough, FunctionTransformer(func=<functi...
wht     [passthrough, FunctionTransformer(func=<functi...
dtype: object
```

In [62]:

```python
col_xform.set_params(txt__vec__lowercase=False)

# run_sweep(
#     main_pipe,
#     filt_grid.add_prefix("col__txt__"),
#     dst="sweeps/txt_filters",
#     kind="grid",
# )
```

Out[62]:

```
ColumnTransformer(transformers=[('txt',
                                 Pipeline(steps=[('uni', 'passthrough'),
                                                 ('low', 'passthrough'),
                                                 ('shrt', 'passthrough'),
                                                 ('usr', 'passthrough'),
                                                 ('num', 'passthrough'),
                                                 ('rep', 'passthrough'),
                                                 ('punc', 'passthrough'),
                                                 ('wht', 'passthrough'),
                                                 ('stem', 'passthrough'),
                                                 ('mark', 'passthrough'),
                                                 ('vec',
                                                  TfidfVectorizer(lowercase=False,
                                                                  tokenizer=<bound method RegexpTok
enizer.tokenize of WordPunctTokenizer(pattern='\\w+|[^\\w\\s]+', gaps=False, discard_empty=True, fl
ags=re.UNICODE|re.MULTILINE|re.DOTALL)>))]),
                                 'text'),
                                ('bra',
                                 CountVectorizer(binary=True,
                                                 tokenizer=<function space_tokenize at 0x000002155D
323CA0>),
                                 'brand_terms')])
```

In [63]:

```python
results = load_results("sweeps/txt_filters")
results.head(10).where(results != "passthrough")
```

Out[63]:

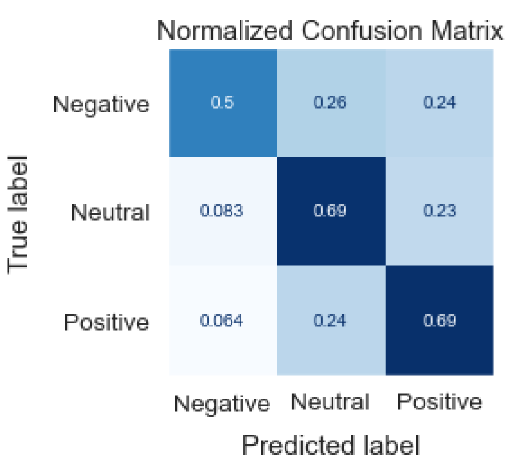
	low	num	punc	rep	shrt	uni	usr	wht	mean_score	rank_score

	low	num	punc	rep	shrt	uni	usr	wht	mean_score	rank_score
145	lowercase()	NaN	NaN	limit_repeats()	NaN	NaN	NaN	strip_multiwhite()	0.638556	1
144	lowercase()	NaN	NaN	limit_repeats()	NaN	NaN	NaN	NaN	0.638556	1
149	lowercase()	NaN	NaN	limit_repeats()	NaN	uni2ascii()	NaN	strip_multiwhite()	0.637429	3
148	lowercase()	NaN	NaN	limit_repeats()	NaN	uni2ascii()	NaN	NaN	0.637429	3
135	lowercase()	NaN	NaN	NaN	NaN	uni2ascii()	strip_handles()	strip_multiwhite()	0.637250	5
134	lowercase()	NaN	NaN	NaN	NaN	uni2ascii()	strip_handles()	NaN	0.637250	5
128	lowercase()	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.636942	7
129	lowercase()	NaN	NaN	NaN	NaN	NaN	NaN	strip_multiwhite()	0.636942	7
133	lowercase()	NaN	NaN	NaN	NaN	uni2ascii()	NaN	strip_multiwhite()	0.636755	9
132	lowercase()	NaN	NaN	NaN	NaN	uni2ascii()	NaN	NaN	0.636755	9

In [64]:

```
main_pipe.set_params(**load_best_params("sweeps/txt_filters"))
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.306	0.811	0.599	0.572	0.708	0.677	0.626
recall	0.496	0.688	0.693	0.626	0.677		
f1-score	0.378	0.744	0.643	0.588	0.687		
support	0.064	0.605	0.332				



Select Stemmer

In [65]:

```
stem_grid = pd.Series(
    {"stem": ["passthrough", funcs.stem_text, funcs.wordnet_lemmatize]}
)
# run_sweep(
#     main_pipe,
#     stem_grid.add_prefix("col__txt__"),
#     dst="sweeps/txt_stem",
#     kind="grid",
# )
```

In [66]:
```
results = load_results("sweeps/txt_stem")
results.style.bar("mean_score")
```

Out[66]:

	stem	mean_score	rank_score
0	passthrough	0.638556	1
2	wordnet_lemmatize()	0.636272	2
1	stem_text()	0.635878	3

Select Marker

In [67]:
```
mark_grid = pd.Series(
    {
        "mark": [
            funcs.mark_pos,
            FunctionTransformer(lang.mark_pos, kw_args={"sep": " "}),
            funcs.mark_negation,
            FunctionTransformer(lang.mark_negation, kw_args={"sep": " "}),
        ]
    }
)

# run_sweep(
#     main_pipe,
#     mark_grid.add_prefix("col__txt__"),
#     dst="sweeps/txt_mark",
#     kind="grid",
# )
```

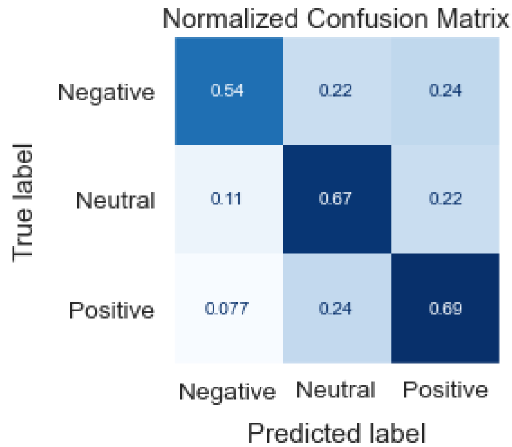
In [68]:
```
results = load_results("sweeps/txt_mark")
results.style.bar("mean_score")
```

Out[68]:

	mark	mean_score	rank_score
3	mark_negation(sep=' ')	0.638587	1
1	mark_pos(sep=' ')	0.631085	2
2	mark_negation()	0.629677	3
0	mark_pos()	0.614449	4

In [69]:
```
main_pipe.set_params(**load_best_params("sweeps/txt_mark"))
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.271	0.816	0.607	0.565	0.712	0.669	0.633
recall	0.539	0.672	0.688	0.633	0.669		
f1-score	0.361	0.737	0.645	0.581	0.683		
support	0.064	0.605	0.332				

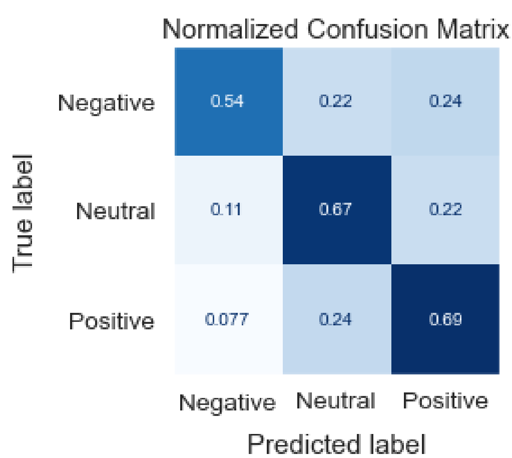


I condense the pipeline into a single function and assign this function to the vectorizer's `preprocessor` parameter. This way, I'll be able to easily get the feature names from `col_xform` later.

In [70]:

```
col_xform.set_params(txt=clone(txt_pipe["vec"]))
text_pp = [lang.lowercase, lang.limit_repeats, partial(lang.mark_negation, sep=" ")]
text_pp = lang.make_preprocessor(text_pp)
col_xform.set_params(txt__preprocessor=text_pp)
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.271	0.816	0.607	0.565	0.712	0.669	0.633
recall	0.539	0.672	0.688	0.633	0.669		
f1-score	0.361	0.737	0.645	0.581	0.683		
support	0.064	0.605	0.332				



Select Stopwords

In [71]:

```
grid = dict(
    col__txt__vec__stop_words=[
        gensim_stop,
        nltk_stop,
        my_stop,
        "english",
        None,
    ]
)
```

```
# run_sweep(
#     main_pipe,
#     grid,
#     dst="sweeps/stop_words",
# )
```

Looks like my stop words are the only ones above the baseline.

In [72]:

```
results = load_results("sweeps/stop_words")
results
```

Out[72]:

	stop_words	mean_score	rank_score
4	None	0.638587	1
1	{shouldn't, own, won, of, are, by, did, aren, ...	0.638381	2
2	{america, mention, southbysouthwest, austin, s...	0.638156	3
0	{four, are, did, five, also, here, except, re,...	0.631351	4
3	english	0.630227	5

Select N-Gram Range

In [73]:

```
ranges = utils.cartesian([1], [1, 2, 3])
grid = pd.Series(dict(ngram_range=ranges.tolist()))
display(grid)

# run_sweep(main_pipe, grid.add_prefix("col__txt__vec__"), dst="sweeps/txt_ngrams")
```

```
ngram_range    [[1, 1], [1, 2], [1, 3]]
dtype: object
```

In [74]:

```
load_results("sweeps/txt_ngrams").style.bar("mean_score")
```

Out[74]:

	ngram_range	mean_score	rank_score
0	[1, 1]	0.638587	1
1	[1, 2]	0.620556	2
2	[1, 3]	0.615984	3

In [75]:

```
df_grid = pd.Series(
    dict(
        min_df=[1, 10, 100],
        max_df=[0.2, 0.4, 0.6, 0.8, 1.0],
    )
)
display(df_grid)

# run_sweep(
#     main_pipe,
#     df_grid.add_prefix("col__txt__vec__"),
#     dst="sweeps/txt_df",
#     kind="grid",
# )
```

```
min_df                  [1, 10, 100]
max_df    [0.2, 0.4, 0.6, 0.8, 1.0]
dtype: object
```

Configure TF*IDF

In [76]:

```
load_results("sweeps/txt_df").head()
```

Out[76]:

	max_df	min_df	mean_score	rank_score
12	1.0	1	0.638587	1
6	0.6	1	0.636253	2
9	0.8	1	0.636253	2
3	0.4	1	0.636031	4
0	0.2	1	0.632922	5

In [77]:

```
tfidf_grid = pd.Series(
    dict(
        binary=[True, False],
        norm=["l2", "l1", None],
        smooth_idf=[True, False],
        sublinear_tf=[True, False],
        use_idf=[True, False],
    )
)
display(tfidf_grid)

# run_sweep(
#     main_pipe,
#     tfidf_grid.add_prefix("col__txt__vec__"),
#     dst="sweeps/txt_tfidf",
#     kind="grid",
# )
```

```
binary              [True, False]
norm               [l2, l1, None]
smooth_idf          [True, False]
sublinear_tf        [True, False]
use_idf             [True, False]
dtype: object
```

In [78]:

```
load_results("sweeps/txt_tfidf").head(10).style.bar("mean_score")
```

Out[78]:

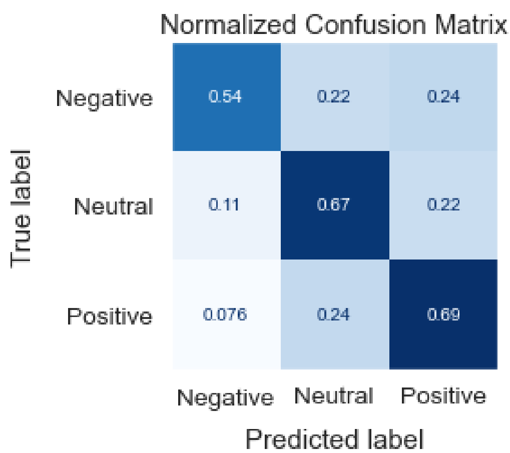
	binary	norm	smooth_idf	sublinear_tf	use_idf	mean_score	rank_score
30	False	l2	False	False	True	0.641701	1
26	False	l2	True	False	True	0.641641	2
0	True	l2	True	True	True	0.638876	3
2	True	l2	True	False	True	0.638876	3
4	True	l2	False	True	True	0.638428	5
6	True	l2	False	False	True	0.638428	5
28	False	l2	False	True	True	0.638184	7

	binary	norm	smooth_idf	sublinear_tf	use_idf	mean_score	rank_score
24	False	l2	True	True	True	0.637555	8
25	False	l2	True	True	False	0.624522	9
29	False	l2	False	True	False	0.624522	9

In [79]:

```
main_pipe.set_params(col__txt__smooth_idf=False, col__txt__sublinear_tf=False)
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.271	0.816	0.608	0.565	0.712	0.669	0.633
recall	0.539	0.672	0.689	0.633	0.669		
f1-score	0.361	0.737	0.646	0.581	0.683		
support	0.064	0.605	0.332				



Add VADER Vectorizer

In [80]:

```
col_xform.transformers.append(("vad", VaderVectorizer(), "text"))
col_xform
```

Out[80]:

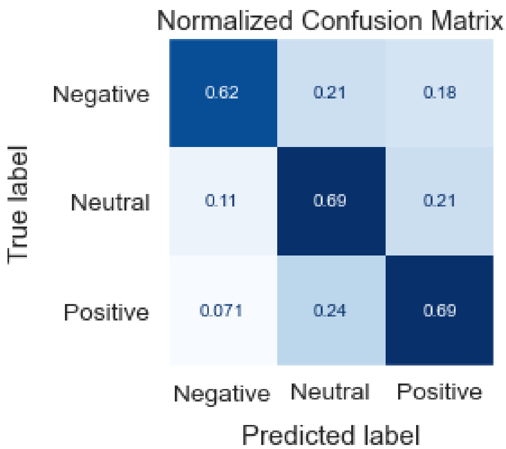
```
ColumnTransformer(transformers=[('txt',
                                 TfidfVectorizer(lowercase=False,
                                                 preprocessor=functools.partial(<function chain_fun
cs at 0x000002155D323A60>, funcs=[<function lowercase at 0x0000021557CD1670>, <function limit_repea
ts at 0x0000021557CD19D0>, functools.partial(<function mark_negation at 0x000002155D325820>, sep='
')]),
                                                 smooth_idf=False,
                                                 tokenizer=<bound method RegexpTokenizer.tokenize o
f WordPunctTokenizer(pattern='\\w+|[^\\w\\s]+', gaps=False, discard_empty=True, flags=re.UNICODE|r
e.MULTILINE|re.DOTALL)>),
                                 'text'),
                                ('bra',
                                 CountVectorizer(binary=True,
                                                 tokenizer=<function space_tokenize at 0x000002155D
323CA0>),
                                 'brand_terms'),
                                ('vad', VaderVectorizer(), 'text')])
```

Looks like a sizable increase in balanced accuracy by just adding 4 new features.

In [81]:

```
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.310	0.816	0.626	0.584	0.721	0.683	0.664
recall	0.617	0.688	0.686	0.664	0.683		
f1-score	0.412	0.747	0.655	0.605	0.695		
support	0.064	0.605	0.332				



In [82]:

```
vader_grid = [
    dict(
        col__vad__trinarize=[True, False],
        col__vad__category_only=[True, False],
        col__vad__norm=["l1", "l2", None],
    ),
    dict(
        col__vad__trinarize=[True, False],
        col__vad__compound_only=[True, False],
        col__vad__norm=["l1", "l2", None],
    ),
]

# run_sweep(
#     main_pipe,
#     vader_grid,
#     dst="sweeps/vader_switches",
# )
```

In [83]:

```
load_results("sweeps/vader_switches").head(10).style.bar("mean_score")
```

Out[83]:

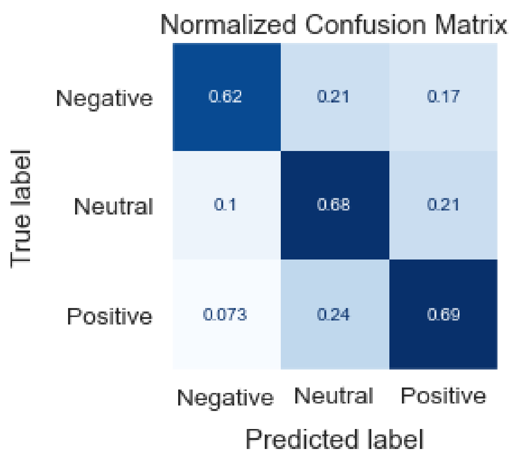
	category_only	compound_only	norm	trinarize	mean_score	rank_score
19	nan	False	l1	False	0.659612	1
7	False	nan	l1	False	0.659612	1
11	False	nan	None	False	0.657781	3
23	nan	False	None	False	0.657781	3
10	False	nan	None	True	0.657339	5
22	nan	False	None	True	0.657339	5
6	False	nan	l1	True	0.657329	7

	category_only	compound_only	norm	trinarize	mean_score	rank_score
18	nan	False	l1	True	0.657329	7
5	True	nan	None	False	0.656781	9
9	False	nan	l2	False	0.656620	10

In [84]:

```
main_pipe.set_params(**load_best_params("sweeps/vader_switches"))
test_fit(main_pipe)
```

	Negative	Neutral	Positive	macro avg	weighted avg	accuracy	bal accuracy
precision	0.312	0.818	0.621	0.584	0.720	0.681	0.665
recall	0.624	0.683	0.689	0.665	0.681		
f1-score	0.416	0.744	0.653	0.604	0.693		
support	0.064	0.605	0.332				



Tune L2 Regularization Strength

In [85]:

```
grid = dict(cls__C=np.geomspace(1e-4, 1e4, 9))

# run_sweep(main_pipe, grid, dst="sweeps/penalty")
```

Looks like the default value of 1.0 is actually the best.

In [86]:

```
load_results("sweeps/penalty").style.bar("mean_score")
```

Out[86]:

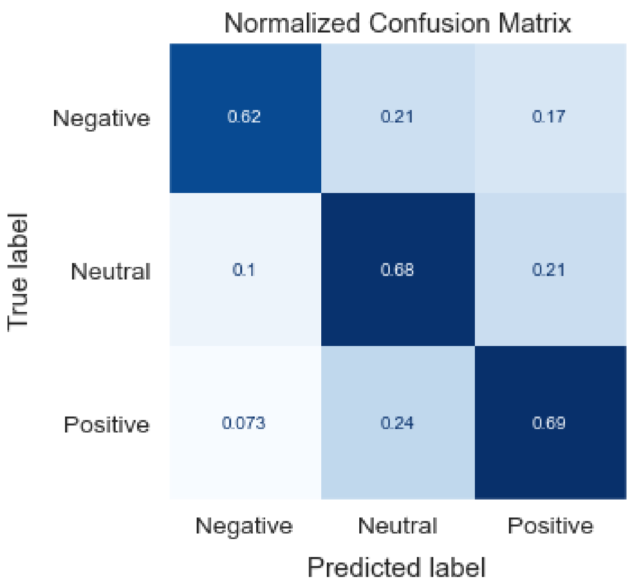
	C	mean_score	rank_score
4	1.000000	0.659612	1
3	0.100000	0.623537	2
5	10.000000	0.622728	3
6	100.000000	0.586719	4
2	0.010000	0.584548	5
7	1000.000000	0.567452	6

	C	mean_score	rank_score
8	10000.000000	0.549594	7
1	0.001000	0.542401	8
0	0.000100	0.519919	9

Save final confusion matrix for the presentation.

In [87]:

```python
fig = diag.classification_plots(
    main_pipe.fit(X_train, y_train), X_test, y_test, pos_label="Positive"
)
fig.savefig(normpath("images/final_confusion_matrix.svg"))
```



Refit with Final Parameters

In [88]:

```python
main_pipe.fit(X, y)
```

Out[88]:

```
Pipeline(steps=[('col',
                 ColumnTransformer(transformers=[('txt',
                                                  TfidfVectorizer(lowercase=False,
                                                                  preprocessor=functools.partial(<f
unction chain_funcs at 0x000002155D323A60>, funcs=[<function lowercase at 0x0000021557CD1670>, <fun
ction limit_repeats at 0x0000021557CD19D0>, functools.partial(<function mark_negation at 0x00000215
5D325820>, sep=' ')]),
                                                                  smooth_idf=False,
                                                                  tok...ue, flags=re.UNICODE|re.MUL
TILINE|re.DOTALL)>),
                                                  'text'),
                                                 ('bra',
                                                  CountVectorizer(binary=True,
                                                                  tokenizer=<function space_tokeniz
e at 0x000002155D323CA0>),
                                                  'brand_terms'),
                                                 ('vad',
                                                  VaderVectorizer(norm='l1'),
                                                  'text')])),
                ('cls',
                 LogisticRegression(class_weight='balanced', max_iter=10000.0,
                                    multi_class='multinomial',
                                    random_state=RandomState(MT19937) at 0x2155D4CA240))])
```

Interpretation

The first order of business is to label the coefficients.

In [89]:

```
feat_names = col_xform.get_feature_names()
classes = main_pipe["cls"].classes_
coef = pd.DataFrame(main_pipe["cls"].coef_, columns=feat_names, index=classes).T
coef.rename({"bra__": "bra__none"}, inplace=True)
coef.sort_values("Negative", ascending=False)
```

Out[89]:

	Negative	Neutral	Positive
txt__headaches	3.370846	-1.580462	-1.790384
txt__fail	3.139523	-1.687693	-1.451830
txt__deleting	2.734694	-1.042144	-1.692549
txt__long	2.469644	-1.181006	-1.288638
txt__fades	2.323666	-1.180612	-1.143054
...	...	...	...
txt__at	-1.592673	1.048761	0.543912
txt__block	-1.597308	1.148435	0.448873
txt__from	-1.605380	0.834986	0.770394
txt__}	-1.682028	0.999499	0.682528
bra__none	-1.922529	2.514256	-0.591727

9877 rows × 3 columns

Top 25 Overall

Then I examine the 25 coefficients with the largest magnitude.

In [90]:

```
top25 = coef.abs().max(axis=1).sort_values().tail(25).index
top25
```

Out[90]:

```
Index(['txt__hm', 'txt__because', 'txt__battery', 'txt__why', 'txt__money',
       'txt__seems', 'vad__neg', 'txt__great', 'txt__needs', 'txt__ridic',
       'txt__suck', 'txt__design', 'txt__sucks', 'txt__again', 'txt__apps',
       'txt__?', 'txt__hate', 'txt__fades', 'txt__long', 'bra__none',
       'txt__deleting', 'txt__cool', 'txt__!', 'txt__fail', 'txt__headaches'],
      dtype='object')
```

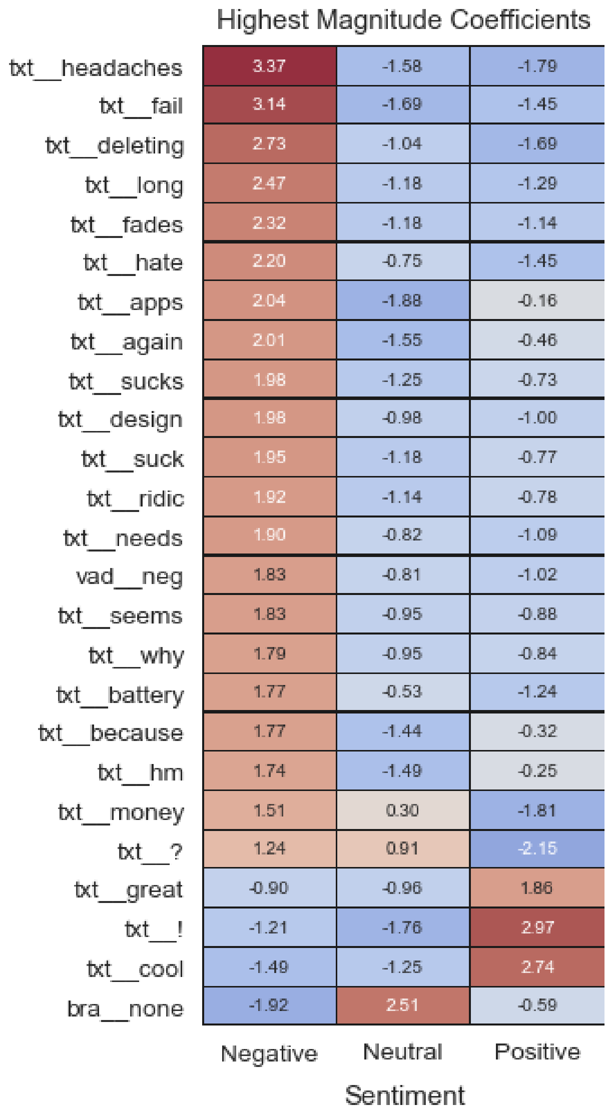
Most of the top 25 coefficients are from the TF*IDF word vectors, unsurprisingly. As predicted, '!' and '?' show up. VADER 'neg' score is quite strong for 'Negative'. The empty brand term category is very strongly related to 'Neutral'.

In [91]:

```
fig, ax = plt.subplots(figsize=(4, 10))
hm_style = dict(plotting.HEATMAP_STYLE)
del hm_style["square"]

sns.heatmap(
    coef.loc[top25].sort_values("Negative", ascending=False),
    ax=ax,
```

```
    square=False,
    **hm_style,
)

ax.set(xlabel="Sentiment")
ax.set_title("Highest Magnitude Coefficients", pad=10)
plotting.save(fig, "images/top25_coef.svg")
```

Out[91]: 'images\\top25_coef.svg'

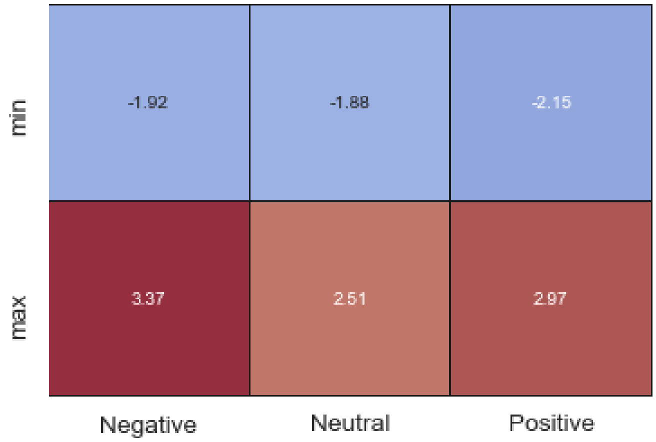


Interesting that the largest overall coefficient is for 'Negative'. Also the maxima are greater in magnitude than the minima.

In [92]:

```
sns.heatmap(
    coef.agg(["min", "max"]),
    square=False,
    **hm_style,
)
```

Out[92]: <AxesSubplot:>



I create a function for grabbing and formatting subsets of the coefficients.

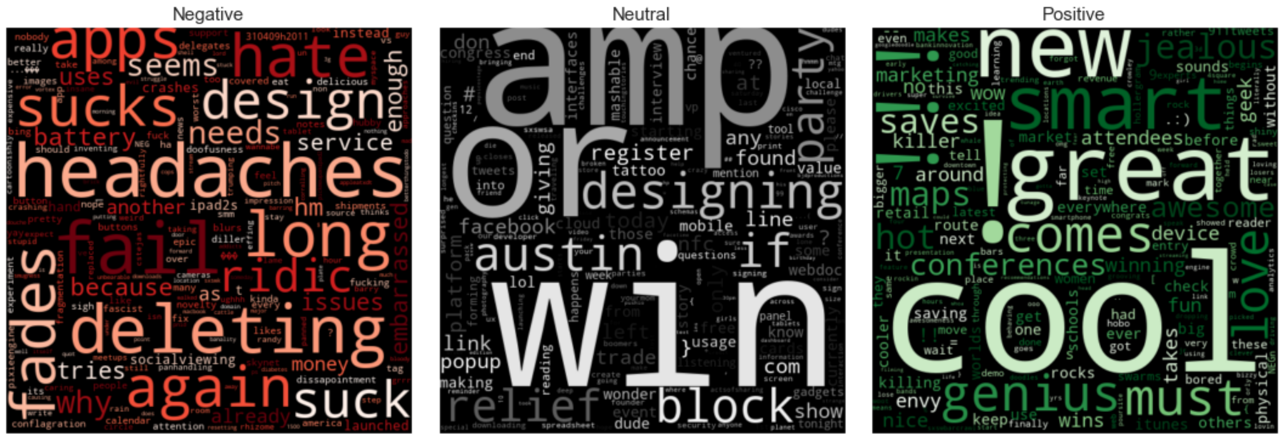
In [93]:
```
def get_coefs(
    prefix,
    index_name,
    coef=coef,
    titlecase=True,
    icase=False,
):
    data = coef.filter(like=f"{prefix}__", axis=0)

    # Remove prefix
    data.index = data.index.str.replace("\w+__", "", regex=True)

    # Make snake_case titlecase
    data.index.name = index_name
    if titlecase:
        data = utils.title_mode(data)
        if icase:
            data.index = data.index.str.replace("Ip", "iP")
    return data.sort_values("Positive")
```

TF*IDF Words

The TF*IDF features were the most influential overall, but they are not the most useful for brand-related insights. See my EDA notebook (exploratory.ipynb) for a deeper brand-related examination of TF*IDF keywords.

The terms in both the 'Positive' and 'Negative' wordclouds make very good sense, and many of them show up in the EDA wordclouds.

In [94]:
```
text_coef = get_coefs("txt", "Text", titlecase=False)

fig = plotting.wordcloud(
    text_coef,
    cmap=dict(Negative="Reds", Neutral="Greys", Positive="Greens"),
    size=(5, 5),
    random_state=rando,
)
fig.savefig(normpath("images/txt_coef_wordclouds.svg"), bbox_inches="tight")
```



Brand Terms

Here are all the brand term coefficients. Remember that these are not TF*IDF features. I extracted these with regular expressions and encoded their presence or absence in each tweet using `CountVectorizer(binary=True)`. These can be thought of like one-hot-encoded categorical variables.

In [95]:

```
brand_coef = get_coefs("bra", "Brand", icase=True)
brand_coef.index = ["None"] + brand_coef.index.to_list()[1:]
if "iPhoneapp" in brand_coef.index:
    brand_coef.drop("iPhoneapp", inplace=True)
brand_coef
```

Out[95]:

	Negative	Neutral	Positive
None	-1.922529	2.514256	-0.591727
iPhone	0.327189	-0.138161	-0.189028
Google	-0.091510	0.121018	-0.029509
Android	-0.315659	0.246953	0.068705
Android App	0.073547	-0.180877	0.107330
iPhone App	0.122709	-0.299128	0.176419
Apple	-0.049582	-0.153267	0.202849
iPad	-0.117353	-0.161968	0.279322
iPad App	-0.453842	0.082832	0.371010

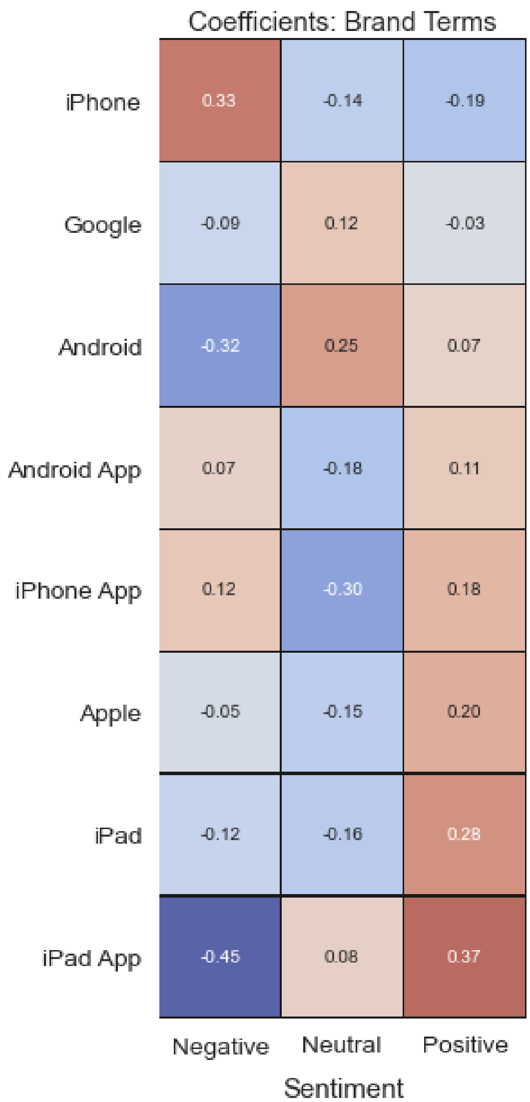
A heatmap is one good way to visualize these coefficients. Unfortunately, it doesn't always bring out the see-saw-like patterns, where a brand term has a positive relationship with one class and a negative relationship with the opposite class.

In [96]:

```
fig, ax = plt.subplots(figsize=(4, 10))
hm_style = dict(plotting.HEATMAP_STYLE)
del hm_style["square"]

if "None" in brand_coef.index:
    brand_coef.drop("None", inplace=True)

sns.heatmap(brand_coef.sort_values("Positive"), ax=ax, square=True, **hm_style)
```

```
ax.set(xlabel="Sentiment", title="Coefficients: Brand Terms")
plotting.save(fig, "images/brand_term_coef.svg")
```

Out[96]: 'images\\brand_term_coef.svg'



In [97]:

```
emo_pal = dict(Negative="r", Neutral="gray", Positive="g")
emo_pal
```

Out[97]: {'Negative': 'r', 'Neutral': 'gray', 'Positive': 'g'}

I create a useful function for making positive vs. negative coefficient plots.

In [98]:

```
def pos_neg_catplot(
    coefs,
    name=None,
    drop_neutral=True,
    palette=emo_pal,
    col_wrap=4,
    sup_y=1.05,
    annot_dist=0.15,
    annot_pad=0.025,
    height=3,
):
    if drop_neutral:
```

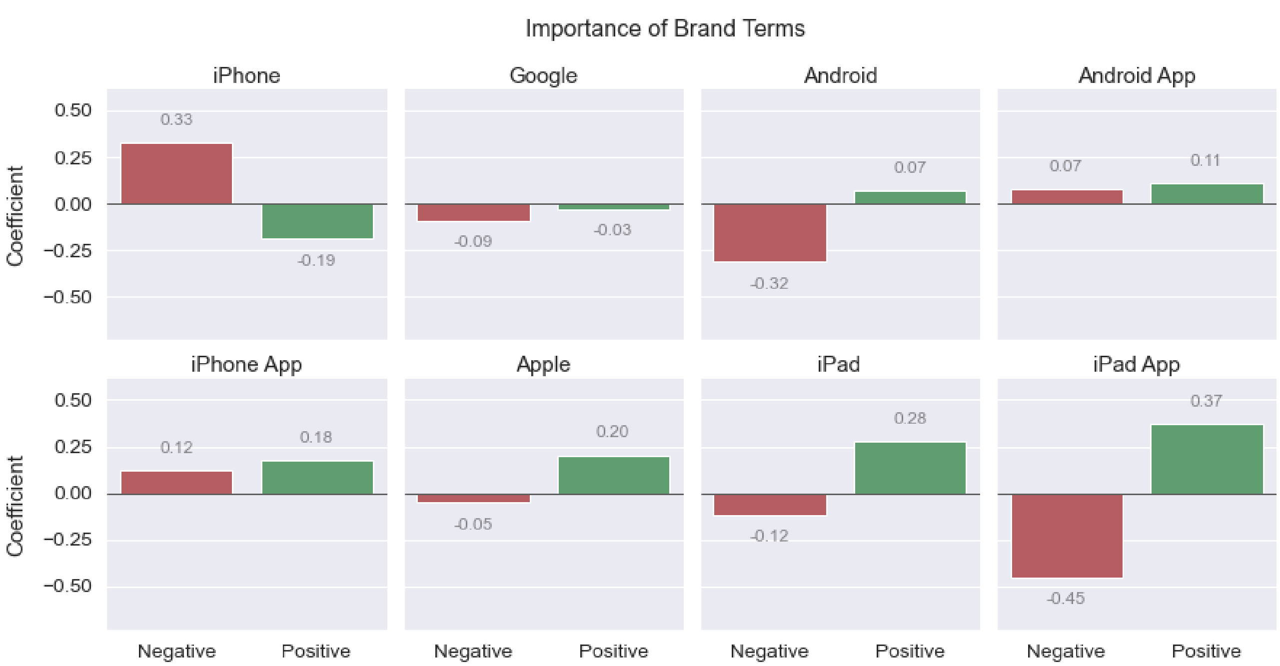
```
        coefs = coefs.drop("Neutral", axis=1)

    # Plot bars on FacetGrid
    g = sns.catplot(
        data=coefs.reset_index(),
        col="index",
        col_wrap=col_wrap,
        kind="bar",
        palette=palette,
        height=height,
    )

    # Annotate
    plotting.annot_bars(g.axes, orient="v", dist=annot_dist, pad=annot_pad)

    # Add horizontal y=0 line
    for ax in g.axes:
        ax.axhline(0, color="k", lw=1, alpha=0.7)

    # Set Axes titles and ylabels
    g.set_titles("{col_name}")
    g.set_ylabels("Coefficient")

    # Create overall title
    if name is None:
        title = "Feature Importances"
    else:
        title = f"Importance of {name}"
    g.fig.suptitle(title, y=sup_y, fontsize=16)
    return g
```

Here is a simplified set of barplots with the 'Neutral' category dropped. Looks like 'iPhone', 'iPad App', and 'Android' have some of the largest coefficients.

In [99]:

```
pos_neg_catplot(brand_coef, name="Brand Terms")
```

Out[99]: `<seaborn.axisgrid.FacetGrid at 0x2156635e9a0>`



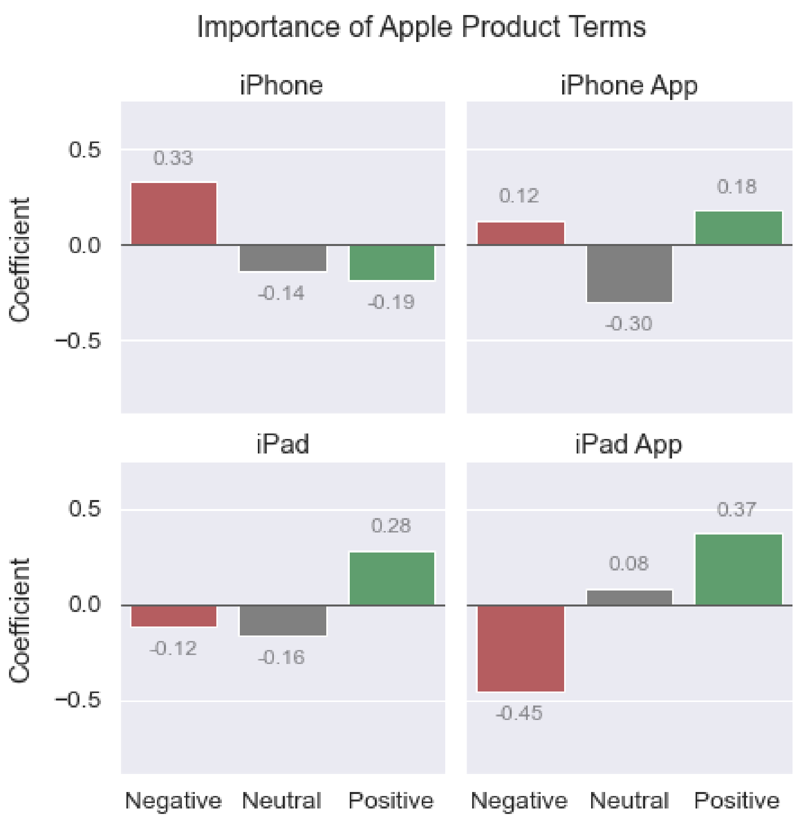
Interestingly, 'iPhone' has "negatively sloped" bars (if you imagine a line connecting them), whereas iPad has positively sloped bars. 'iPad App' is even more dramatically positive, and 'iPhone App' has a weak relationship with both 'Positive' and 'Negative'. 'iPhone App' does have a negative relationship with 'Neutral', however, indicating some amount of controversy.

In [100…

```
g = pos_neg_catplot(
    brand_coef.loc[["iPhone", "iPhone App", "iPad", "iPad App"]],
    name="Apple Product Terms",
    col_wrap=2,
    annot_pad=0.05,
    sup_y=1.05,
    drop_neutral=False,
)
plotting.save(g.fig, "images/apple_prod_term_coef.svg")
```

Out[100…

```
'images\\apple_prod_term_coef.svg'
```



VADER Valence

Here are the VADER coefficients. They are relatively large, as expected. Adding VADER vectors to the mix proved to be a good idea.

In [101…

```
vad_coef = coef.filter(like="vad__", axis=0)
vad_coef.index = vad_coef.index.str.replace("vad__", "")
vad_coef = utils.title_mode(vad_coef)
vad_coef
```

Out[101…

	Negative	Neutral	Positive
Neg	1.832594	-0.810800	-1.021795
Neu	-0.503304	0.812233	-0.308930
Pos	0.960299	-1.410587	0.450288

	Negative	Neutral	Positive
Compound	-1.408255	0.242419	1.165835

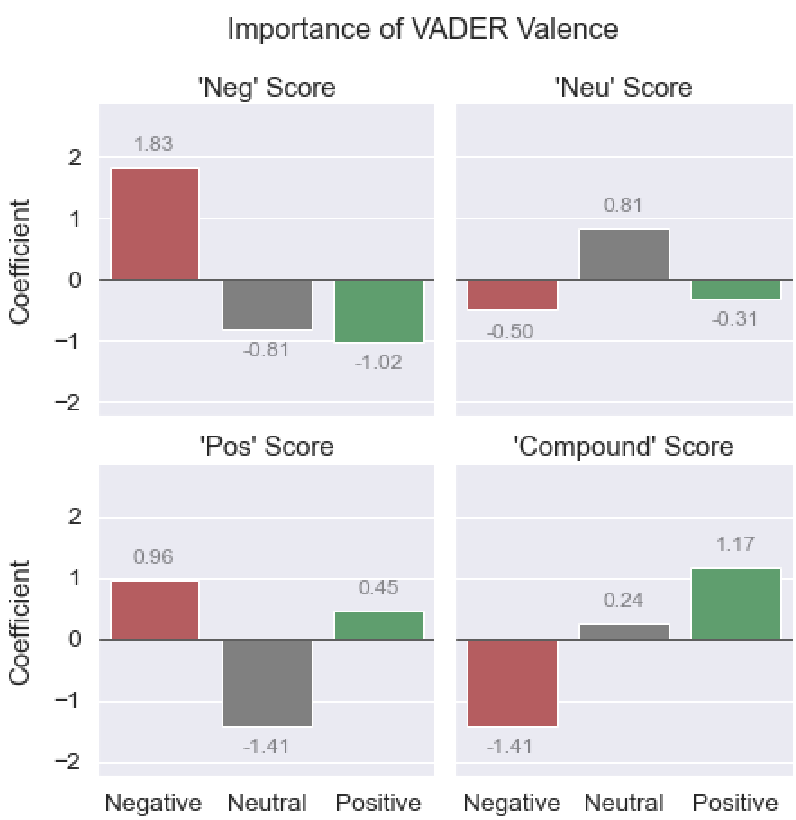
The 'Neg' VADER polarity scores have a strong relationship with 'Negative', unsurprisingly. What is surprising is that 'Pos' has a stronger relationship with 'Negative' than 'Positive'. 'Pos' has a very strong negative relationship with the 'Neutral' class, at least. The 'Compound' score is "positively sloped", as one might have predicted.

In [102…

```
g = pos_neg_catplot(
    vad_coef,
    name="VADER Valence",
    col_wrap=2,
    annot_dist=0.45,
    annot_pad=0.03,
    drop_neutral=False,
)

g.set_titles("'{col_name}' Score")
```

Out[102…

```
<seaborn.axisgrid.FacetGrid at 0x2156638b9d0>
```



Recommendations

1. Try to shake your authoritarian image by ostensibly allowing end-users more freedom.

People like that Apple products just work out of the box, but they find your paternalistic approach to managing your products off-putting. **Send the message** that when you buy an Apple product, you are free to do what you want with it. Keep control over the most important things, but relinquish control over the less important things. Make people feel like they have the freedom to customize your products as they see fit. Make some concessions to placate the majority, while allowing the elite techno-snobs to continue complaining on the fringe.

2. Do something to improve the iPhone's battery life and turn it into a marketing campaign.

There were a lot complaints about the iPhone's battery life. One user suggested that their Blackberry was doing much better. There were also complaints about #batterykiller apps which use geolocation in the background. If you made a big publicized effort to increase the iPhone's battery life, that would get people excited.

3. Expand the App Store offerings for iPad.

There is a lot of enthusiasm about iPad apps, and less about iPhone apps and iPhone in general. Focus more energy on both developing iPad apps and nurturing the iPad development community.

Future Work

Stacking Classifiers

After experimenting a little with Scikit-Learn's `StackingClassifier`, it's become clear that I could use it to develop a more accurate final model. The `StackingClassifier` trains several classifiers on the data and then trains a final classifier on the concatenated output of those classifiers. It also allows you to pass the training data to your final estimator, so the final estimator is trained both on prior data and the predictions of the classifier ensemble.

Sophisticated Vectorization

I experimented some with Doc2Vec, a sophisticated unsupervised document vectorization algorithm, but didn't find it to offer any advantage over `TfidfVectorizer` when trained on this small dataset. It proved to be slower, much more complicated, and much less interpretable. However, if trained on a large corpus of tweets, and then used to predict vectors for the present dataset, it could prove to be better than TF*IDF vectorization. Even if the Doc2Vec vectors didn't turn out to be better than the TF*IDF vectors, they could potentially augment them. A Doc2Vec model trained on a large corpus would probably contribute **novel information**.

Conclusion

I created a reasonably accurate model, at around 0.68 accuracy and 0.67 balanced accuracy. However, I'm confident that I can raise increase the accuracy even more by stacking classifiers. I'd also like to try alternative methods of vectorization, but I'm not as confident that it will improve the model.

Through interpreting my model and conducting a brief exploratory analysis in exploratory.ipynb, I arrived at three recommendations. First, you should relinquish a small (ceremonial) amount of control over your products in a public manner, to sent the message that you're not tyrants. Second, you should improve the iPhone's battery life and turn that into a rallying point for a marketing campaign. People are really concerned about their battery life. Third, you should invest money and resources into expanding iPad app development. People are excited about the iPad and its potential, and somewhat jaded about the iPhone.

In [ ]:

```
```