How Do You Measure Word of Mouth?

Tweet Sentiment Analysis Using NLP

The objective of this project is to explore the numerous Natural Language Processing tools currently available to see what performs effectively in attempting to do two things: 1) Extracting essential pieces of information from tweets and 2) determine the sentiment c tweets. The business applications for these tools are endless, but the main focus of the project is to build a pipeline for processing l amounts of text data given a reasonable training dataset. Using these models, we can measure the impact of marketing campaigns much people are discussing certain products on social media and how they feel about them. While some people say all press is goc press, negative word of mouth can be far more damaging than no market penetration at all. Also, tools like this can be used for gath feedback on public perception of products and companies.

The Data

The dataset is a collection of 9000 tweets all sent around the time of the SXSW festival regarding different products from Apple and Android, namely iPhones, iPads, Google Apps, and Apple Apps. Each tweet is tagged with the products the tweet concerns as well 'Positive Emotion', 'Negative Emotion', 'No emotion', or 'I can't tell'. The tweets include non-ASCII characters as well as the usual c social media communication. The data is extremely noisy because Twitter is a chaotic place. Some of the tweets aren't even in Engl which is difficult to parse out without an effective language detector. Also, about half of the tweets have not been labeled with their product, so that data will not be included in the training/testing data for the entity recognition model.

Approach

One of the challenges of NLP is deciding on both a vectorization method and how to clean the data before vectorization. There are s tools and methodologies available to approach both of these tasks. The core of Natural Language Processing is turning the text intc that can then be vectorized. To do this, we can either use basic REGEX extraction to find essential information, lemmatization, or we use more advanced libraries of word embeddings such as SpaCy. For this project I will be using three vectorization methods:

```
1) Bag of Words Count Vectorization
2) Term Frequency - Inverse Document Frequency (TF-IDF) Vectorization
3) SpaCy Doc pipeline vectorization
```

I expect there to be a large difference in the quality of these vectorization methods. Each tweet is, by definition, quite short and peo tend to use punctuation and full sentences at random, which could make the SpaCy doc vectorization extremely noisy. For TF-IDF, I that to be more successful at identifying products, as it would make sense an iPad tweet would contain the word 'iPad' and not the 'Google' more often than not.

Preprocessing

However, before we vectorize, we must clean the data. Since these tweets are very noisy with special characters, websites, handles image links, those will have to be removed. The two major choices after that would be concerning stop words (common words with inherent entropic value) and hashtags.

In order to test the various models I created a TextSet class that will allow various preprocessing options. Hashtags are an interestin linguistic element of social media because they can be frustrating noise or they can be topic tags, meaning they are extremely usefu extremely confounding. Take the fake tweet:

```
This iPad's battery is great #justkidding #itsucks
```

Versus:

```
This iPad's battery is great
```

In this case, removing the hashtags completely changes the meaning of the tweet. And so, for each of the vectorization methods, I v create one dataset without Hashtags and one with. The other major processing choice concerns stop words. The SpaCy vectorizati method takes into account sentence structures and punctuation, so I will create two SpaCy datasets, one with hashtags and one wi while I will remove stop words from both the TF-IDF dataset and the Count Vectorized Dataset to measure their performance agains another.

Model Comparison

I will be using several different multi-class classification models to solve these objects. In order to measure the accuracy of each mo against all 4 datasets, I created the ModelComparison class that can produce key accuracy, precision, recall metrics for non-neural models as well as validation and training loss charts for neural network machine learning models. I will be exploring the effectivenes following models on the various datasets:

- Gridsearch Support Vector Machine using 'rbf' and 'sigmoid' kernels
- Gridsearch polynomial kernal for a Support Vector Machine
- Gridsearch Random Forest Classifier
- Complement Naive Bayes (only on TF-IDF and Count vectorized models)
- Multinomial Naive Bayes (only on TF-IDF and Count vectorized models)
- Tensorflow Sequential Neural Network

So, let's take a look at the data and define baselines for model validation.

Part 1: Load and Clean Data

In [1]:
```
import pandas as pd
import numpy as np
import nltk
import string
import re
from nltk.corpus import stopwords
from nltk import FreqDist
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from nltk import word_tokenize, FreqDist
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_s
from sklearn import svm
import matplotlib.pyplot as plt
import spacy
from keras.models import Sequential
from keras import regularizers
from keras.layers import Dense, Dropout, Layer
import time
import warnings
from utils import TextSet, ModelComparison, process_text

warnings.filterwarnings('ignore')
%matplotlib inline
np.random.seed(16)
```

In [3]:
```
df = pd.read_csv('judge-1377884607_tweet_product_company.csv', encoding='unicode_escape')
df.head()
```

Out[3]:

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

As we can already see in the first few rows, keeping hashtags will probably be the better way to approach the problem, but we can s
how impactful it will be in the long run.

In [4]:
```
df.shape
```

Out[4]:
```
(9093, 3)
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
tweet_text                                            1
emotion_in_tweet_is_directed_at                    5802
is_there_an_emotion_directed_at_a_brand_or_product    0
dtype: int64
```

That is a lot of nulls in the 'product' column. While we will be assessing sentiment first, we will need to split the known product rows
the unknown, severely limiting our dataset, which could cause issues ultimately. Also we do not have a 'No Product' tag which will r
hard to generalize this model to the larger population of tweets. We won't be concerned with that for the purposes of this project, bu
major consideration for further iterations.

In [6]:
```
unknown_df = df[df['emotion_in_tweet_is_directed_at'].isna()==True]
known_df = df[df['emotion_in_tweet_is_directed_at'].isna()==False]
```

In [7]:
```
len(known_df)
```

Out[7]:
```
3291
```

In [8]:
```
len(unknown_df)
```

Out[8]:
```
5802
```

Now, let's examine the distribution of these products to determine the baseline accuracy values our models will have to beat.

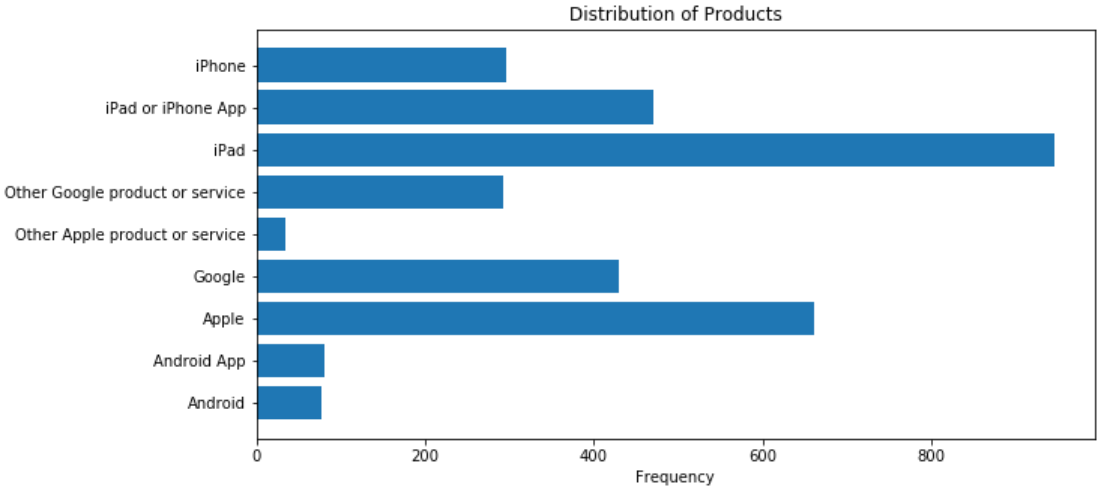
In [9]:
```
df['emotion_in_tweet_is_directed_at'].value_counts(normalize=True)
```

Out[9]:
```
iPad                               0.287451
Apple                              0.200851
iPad or iPhone App                 0.142814
Google                             0.130659
iPhone                             0.090246
Other Google product or service    0.089031
Android App                        0.024613
Android                            0.023701
Other Apple product or service     0.010635
Name: emotion_in_tweet_is_directed_at, dtype: float64
```

Product Entity Recognition: Baseline = 28.8% Accuracy

If we blindly guess iPad, we will be right 29% of the time.

In [10]:
```
products = df.groupby('emotion_in_tweet_is_directed_at').count()['tweet_text']
p_labels = list(products.index)
plt.figure(figsize=(10,5))
plt.barh(y=p_labels, width=products)
plt.title('Distribution of Products')
plt.xlabel('Frequency')
plt.show()
```



Sentiment Analysis: Baseline = 60.9% Accuracy

Below you can see the break down of each category into both number of entries in the dataset as well as relative frequencies.

In [11]:
```
df['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts(normalize=True)
```

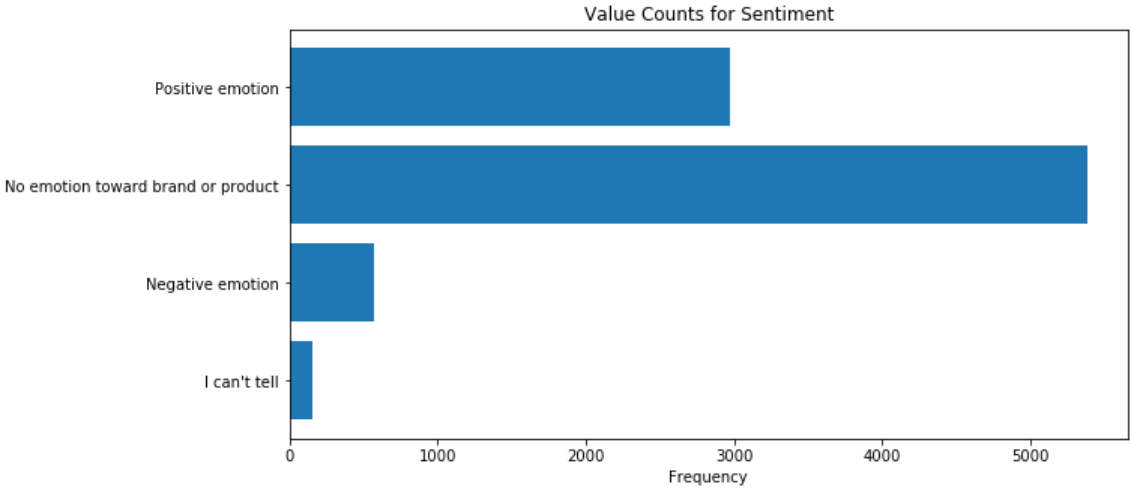
Out[11]:
```
No emotion toward brand or product    0.592654
Positive emotion                      0.327505
Negative emotion                      0.062686
I can't tell                          0.017156
Name: is_there_an_emotion_directed_at_a_brand_or_product, dtype: float64
```

In [12]:
```
df.groupby('is_there_an_emotion_directed_at_a_brand_or_product').count()
```

Out[12]:

is_there_an_emotion_directed_at_a_brand_or_product	tweet_text	emotion_in_tweet_is_directed_at
I can't tell	156	9
Negative emotion	570	519
No emotion toward brand or product	5388	91
Positive emotion	2978	2672

In [13]:
```
emotions = df.groupby('is_there_an_emotion_directed_at_a_brand_or_product').count()['tweet_tex
e_labels = list(emotions.index)
plt.figure(figsize=(10,5))
plt.barh(y=e_labels, width=emotions)
plt.title('Value Counts for Sentiment')
plt.xlabel('Frequency')
plt.show()
```



As you can see there are so few 'I can't tell' tweets that we might as well include those in the 'No Emotion toward Brand or Product also don't want to include duplicates of possibly bot-produced tweets in the dataset either.

In [14]:
```
df.dropna(subset=['tweet_text'],inplace=True)
```

In [15]:
```
df.drop_duplicates(inplace=True)
```

Let's take a quick snapshot of the first 15 tweets to get a sense of what kind of preprocessing we have to do to remove noise:

In [16]:
```
for n in range(15):
    print(df[df['is_there_an_emotion_directed_at_a_brand_or_product'] == "I can't tell"]['twee
```

```
Thanks to @mention for publishing the news of @mention new medical Apps at the #sxswi conf. blog {
k} #sxsw #sxswh
ÛÏ@mention &quot;Apple has opened a pop-up store in Austin so the nerds in town for #SXSW can get
r new iPads. {link} #wow
Just what America needs. RT @mention Google to Launch Major New Social Network Called Circles, Pos
y Today {link} #sxsw
The queue at the Apple Store in Austin is FOUR blocks long. Crazy stuff! #sxsw
Hope it's better than wave RT @mention Buzz is: Google's previewing a social networking platform a
XSW: {link}
SYD #SXSW crew your iPhone extra juice pods have been procured.
Why Barry Diller thinks iPad only content is nuts @mention #SXSW {link}
Gave into extreme temptation at #SXSW and bought an iPad 2... #impulse
Catch 22ÛÒ I mean iPad 2 at #SXSW : {link}
Forgot my iPhone for #sxsw. Android only. Knife to a gun fight
Kawasaki: key to enchantment = trustworthiness of Zappos + likeability of Richard Branson + produc
Apple #sxsw #mccannsxsw #mrmworldwide
Google no lanzara ningun producto en South by SouthWest #sxsw 2011 {link}
Comprando mi iPad 2 en el #SXSW (@mention Apple Store, SXSW w/ 62 others) {link}
I can now say that Google got me drunk #sxsw #h4ckers
ipad is a slow resting heartrate  #tapworthy #sxsw #gsdm about leisure vs quick results on iphone
```

As you can see we have plenty of strange symbols, non-unicode characters, website links, hashtags and even some Spanish tweets start processing these tweets into our 4 datasets.

Part 2: Preprocessing and Vectorization

As stated above, I will be comparing the performance of four different datasets on each of our possible classifiers.

- TF-IDF Vectorization with Hastags included
- SpaCy Doc2Vec Vectorization with Hashtags included
- SpaCy Doc2Vec Vectorization with Hashtags excluded
- Count Vectorization with Hashtags excluded

Since the Sentiment Analysis dataset is larger, we will tackle that objective first. As mentioned, we will combine both the 'I can't tell' 'No Emotion toward brand or product' to give us a majority class of **60.9%**. This is the number for us to beat.

In [17]:
```
df['is_there_an_emotion_directed_at_a_brand_or_product'].replace("I can't tell", "No emotion t
```

In [18]:
```
df['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts(normalize=True)
```

Out[18]:
```
No emotion toward brand or product    0.609813
Positive emotion                      0.327453
Negative emotion                      0.062734
Name: is_there_an_emotion_directed_at_a_brand_or_product, dtype: float64
```

In [19]:
```
df[df['emotion_in_tweet_is_directed_at'].isna()==True]
```

Out[19]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_p
5	@teachntech00 New iPad Apps For #SpeechTherapy...	NaN	No emotion toward brand or p
16	Holler Gram for iPad on the iTunes App Store -...	NaN	No emotion toward brand or p
32	Attn: All #SXSW frineds, @mention Register fo...	NaN	No emotion toward brand or p
33	Anyone at #sxsw want to sell their old iPad?	NaN	No emotion toward brand or p
34	Anyone at #SXSW who bought the new iPad want ...	NaN	No emotion toward brand or p
...	...	...	
9087	@mention Yup, but I don't have a third app yet...	NaN	No emotion toward brand or p
9089	Wave, buzz... RT @mention We interrupt your re...	NaN	No emotion toward brand or p
9090	Google's Zeiger, a physician never reported po...	NaN	No emotion toward brand or p
9091	Some Verizon iPhone customers complained their...	NaN	No emotion toward brand or p
9092	¢Ïį ÏàŠü_‹ Ê‹ Î‹ Ò‹ £‹ Á‹ââ‹ _‹ £‹ ‹â_‹ÛâRT @...	NaN	No emotion toward brand or p

5788 rows × 3 columns

Process_Text Function

Below you will see the function used to process the texts. It is included in the TextSet class, but it is useful for showing how the text processed. Before anything we will define stopwords using the NLTK stop word library and we will be using the SpaCy basic pipelin with the small English vocabulary loaded:

In [20]:
```
nlp = spacy.load("en_core_web_sm")

stopwords_list = stopwords.words('english') + [' ', '  ', '   ', 'w/']
```

In [21]:

```
def process_text(text, is_spacy=False, keep_stopwords=True, keep_links=False, keep_hashtags =

    '''
    Processes, cleans, and tokenizes text with multiple options on how to do so. Returns eithe

    Parameters
    ----------
    text - (str) - the text to be processed
    is_spacy - (bool) - whether or not the return is a SpaCy doc with stop words included. If
    keep_stopwords - (bool) - whether or not to remove stopwords using the nltk stopwords libr
    keep_links - (bool) - whether or not to turn a handle and website link into a blank word s
    keep_hashtags - (bool) - whether or not to simply remove the # symbol from a hashtag or re

    Returns
    -------
    Processed text - doc, if OR list

    '''

    if keep_links == True: # Removes all {links} as well as urls and @handles which sometimes
        text = re.sub('\.?(@[A-Za-z0-9_]+)', 'HANDLE', text)
        text = re.sub('{([A-Za-z]+)}', 'WEBSITE', text)
        text = re.sub('(http://[A-Za-z./0-9?=-]+)', 'WEBSITE', text)
    else:
        text = re.sub('\.?(@[A-Za-z0-9_]+)', '', text)
        text = re.sub('{([A-Za-z]+)}', '', text)
        text = re.sub('(http://[A-Za-z./0-9?=-]+)', '', text)

    if keep_hashtags == False: # Either removes the complete hashtag or just the '#' symbol
        text = re.sub('(#[A-Za-z0-9_]+)', '', text)
    else:
        text = text.replace('#', '')

    text = re.sub('&([A-Za-z])+;', '', text) # Removes special html characters such as &quot
    text = re.sub('(\s){2,}', ' ', text) # Turns double/triple spaces into single spaces
    text = text.replace("'", '') # Removes apostrophes
    text = text.replace('RT', '') # Removes if the tweets have been retweeted
    text = text.replace('\x89Û÷', '') # Removes special characters
    text = text.replace('\x89Ûª', '')
    text = text.replace('\x89ÛÏ', '')
    text = text.replace('\x9d', '')
    text = text.replace('\x89Û', '')
    text = text.replace('\x89ÛÒ', '')
    text = text.replace('[pic]', '') # Removes picture links
    text = text.replace('(\\x89)(\S)*', '')
    doc = nlp(text) # Turns the resulting string into a SpaCy doc object

    if is_spacy == True:
        return doc

    text = text.replace("'", '') # Removes apostrophes

    if keep_stopwords == False: # Makes all tokens lower case, removes punctuation and stopwor
        return [x.lower() for x in list([token.text for token in doc if token.is_punct == Fals
    else: # Does the same thing, but keeps stopwords in
        return [x.lower() for x in list([token.text for token in doc if token.is_punct == Fals
```

Below you will see the loading and processing of the following datasets.

- TF-IDF Vectorization with Hastags included
- SpaCy Doc2Vec Vectorization with Hashtags included
- SpaCy Doc2Vec Vectorization with Hashtags excluded
- Count Vectorization with Hashtags excluded

The TextSet class already splits the data into a holdout, train and test set and provides two processing methods for our y values to r them compatible with all methods of Classification, label encoded for SVM and Naive Bayes, one hot encoded matrices for Neural Networks.

In [22]:
```
hashtags = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='Hashtags',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 56.80351996421814 seconds ---
```

In [23]:
```
spacy_1 = TextSet(df['tweet_text'],
                  df['is_there_an_emotion_directed_at_a_brand_or_product'],
                  name = 'SpaCy No Hash',
                  is_spacy=True,
                  keep_stopwords=True,
                  random_seed=42,
                  split=.3)
```

```
--- 42.942533016204834 seconds ---
```

In [24]:
```
spacy_2 = TextSet(df['tweet_text'],
                  df['is_there_an_emotion_directed_at_a_brand_or_product'],
                  name = 'SpaCy w/ Hash',
                  is_spacy=True,
                  keep_stopwords=True,
                  keep_hashtags=True,
                  random_seed=42,
                  split=.3)
```

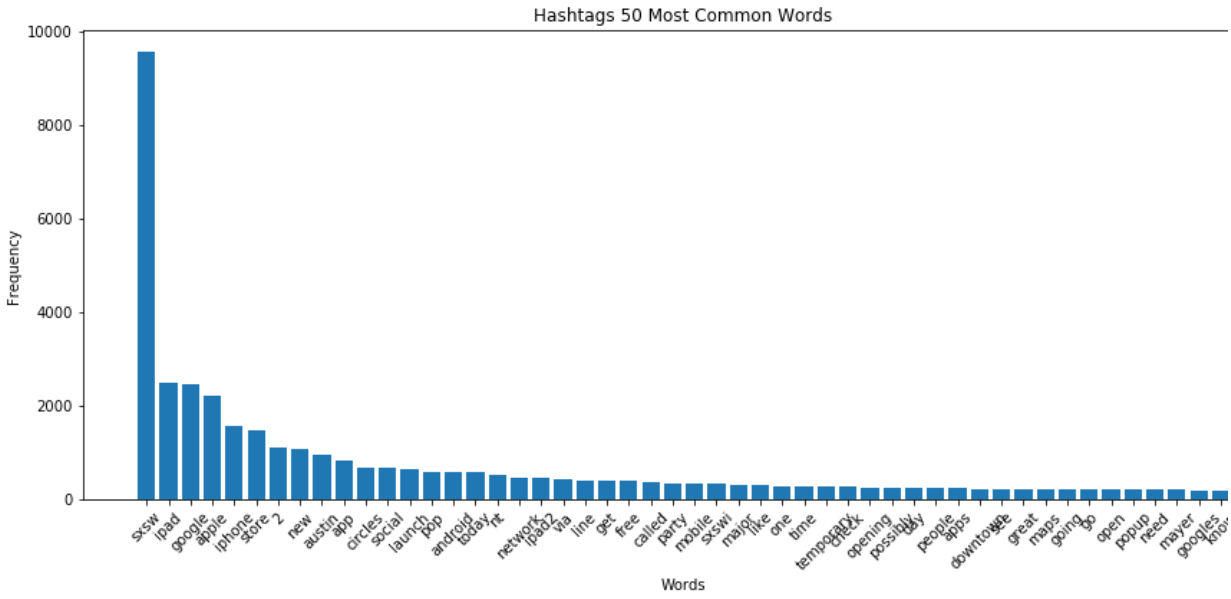
```
--- 49.31569314002991 seconds ---
```

In [25]:
```
no_hash = TextSet(df['tweet_text'],
                  df['is_there_an_emotion_directed_at_a_brand_or_product'],
                  name='No Hashtags',
                  random_seed=42,
                  split=.3)
```

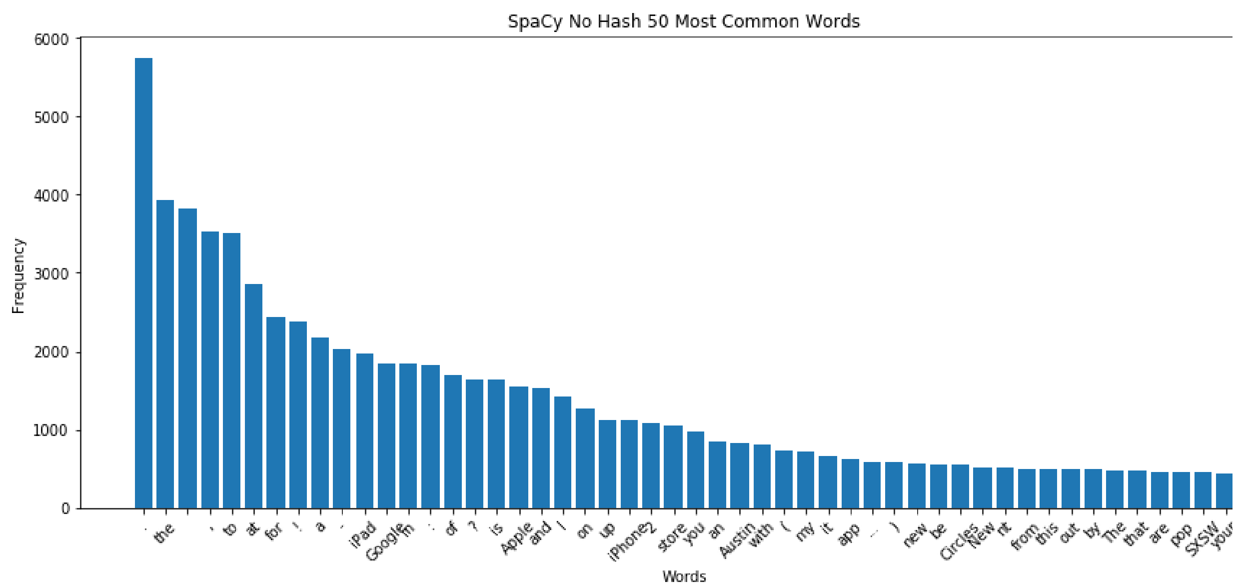
```
--- 42.582802057266235 seconds ---
```

In [26]:
```
datasets = [hashtags, spacy_1, spacy_2, no_hash]
no_spacy = [hashtags, no_hash]
```

In [27]:
```
for data in datasets:
    data.word_cloud()
    data.plot_frequency()
```

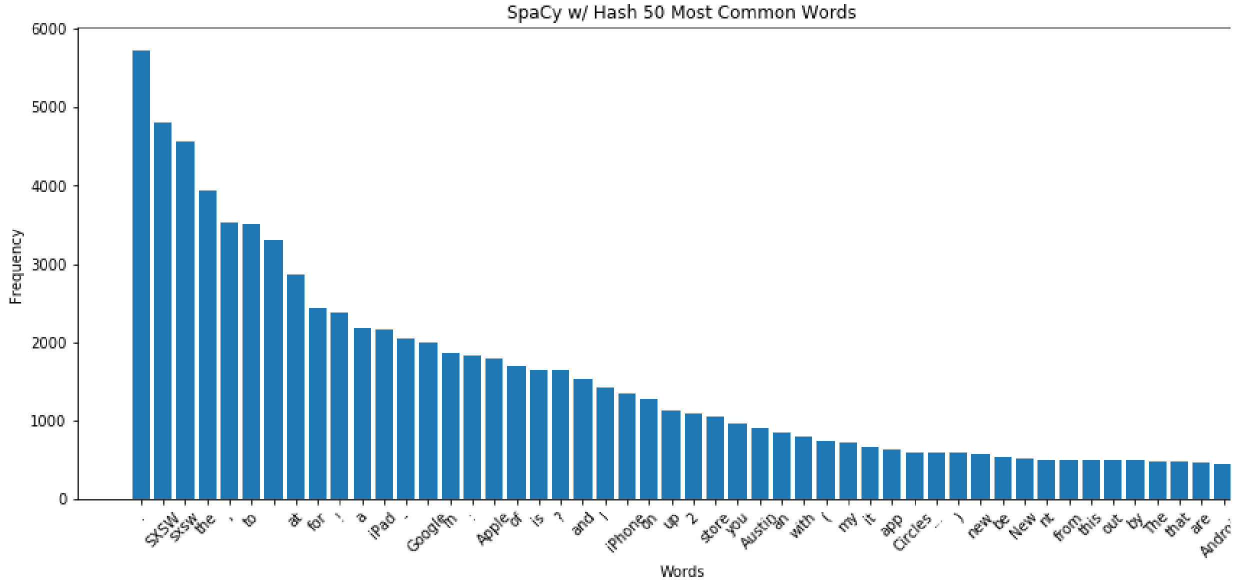



Word Cloud For SpaCy No Hash
Social Network
Google
iPad
store
Apple Store
Social
line
SXSW
Network Called
Austin
Called Circle
Apple
now
New
iPhone
app
pop
one
will
via
Major New
Launch Major
Possibly Today
Circles Possibly
iPhone app
Marissa Mayer
temporary store
downtown Austin


SpaCy No Hash 50 Most Common Words
Frequency
Words
0
1000
2000
3000
4000
5000
6000
the
to
at
for
a
iPad
Google
rt
of
is
Apple
and
on
up
iPhone2
store
you
an
Austin
with
my
it
app
new
be
Circles
New
nt
from
this
out
by
The
that
are
pop
SXSW
your

Word Cloud For SpaCy w/ Hash

SpaCy w/ Hash 50 Most Common Words



Word Cloud For No Hashtags

Observations

As you can see the frequency distributions and word clouds can differ greatly between datasets. You'll notice with the SpaCy datas there are multi-word phrases such as Apple Store. Also the hashtag #SXSW is used more than any other word, so removing hashta massively alters the word distributions.

Now, we have to vectorize each dataset. The default method for non-SpaCy datasets is TF-IDF with 300 features, though that can b altered as needed. The SpaCy datasets default to the Doc2Vec method in the SpaCy doc object.

In [28]:
```
hashtags.vectorize()
```

In [29]:
```
spacy_1.vectorize()
```

In [30]:
```
spacy_2.vectorize()
```

In [31]:
```
no_hash.vectorize(method='count')
```

Part 3: Sentiment Analysis

Now that we have our datasets, we can feed them into our various models to compare how they do. First I will be comparing only te scores and then I will test holdout scores between the best models. Since the datatypes required for neural networks are fairly differ from these other classifiers, I will compare those after tuning the non-neural networks to maximize accuracy.

Non Neural Network Models

Here are the models we will be testing. Keep in mind, some of these cells take a lot time to run. I recommend skipping the random f cells.

- Support Vector Machine Gridsearch
- Polynomial Support Vector Machine Gridsearch
- Random Forest
- Complement Naive Bayes
- Multinomial Naive Bayes

In [32]:
```
svm_pipe = Pipeline([('scaler', StandardScaler()), ('rbf', svm.SVC())])

grid = {
    'rbf__kernel': ['rbf', 'sigmoid'],
    'rbf__gamma': ['scale'],
    'rbf__C': [1, 1e12],
    'rbf__decision_function_shape': ['ovo','ovr']
}

svm_grid = GridSearchCV(svm_pipe, param_grid=grid, scoring='accuracy', cv=5)
svm_compare = ModelComparison(pipeline=svm_grid, data_list=datasets, y_format='label', name='s
```

In [33]:
```
poly_pipe = Pipeline([('scaler', StandardScaler()), ('poly', svm.SVC())])

grid = {
    'poly__kernel': ['poly'],
    'poly__degree': [3, 4, 5],
    'poly__gamma': ['auto'],
    'poly__decision_function_shape': ['ovo','ovr']
}

poly_grid = GridSearchCV(poly_pipe, param_grid=grid, scoring='accuracy', cv=5)
poly_compare = ModelComparison(pipeline=poly_grid, data_list=[hashtags, spacy_1], y_format='la
```

In [34]:
```
cnb_pipe = Pipeline([('scaler', MinMaxScaler()), ('cnb', ComplementNB())])
cnb_compare = ModelComparison(pipeline=cnb_pipe, data_list=no_spacy, y_format='label', name='c
```

In [35]:
```
mnb_pipe = Pipeline([('scaler', MinMaxScaler()), ('mnb', MultinomialNB())])
mnb_compare = ModelComparison(pipeline=mnb_pipe, data_list=no_spacy, y_format='label', name='m
```

In [36]:
```
rf_pipe = Pipeline([('scaler', StandardScaler()), ('rf', RandomForestClassifier())])

grid = {
    'rf__n_estimators': [25, 50, 100],
    'rf__criterion': ['gini','entropy'],
    'rf__max_depth': [5, 7, 9],
    'rf__max_features': [5, 8, None],
    'rf__min_samples_leaf': [3, 5, 10],
    'rf__random_state': [42]
}

rf_grid = GridSearchCV(rf_pipe, param_grid=grid, scoring='accuracy', cv=5)
rf_compare = ModelComparison(pipeline=rf_grid, data_list=datasets, y_format='label', name='rf'
```

In [37]:
```
svm_compare.fit_models()
svm_compare.score_comparison
```

```
--- 549.7049558162689 seconds to process ---
```

Out[37]:

	Hashtags Test	SpaCy No Hash Test	SpaCy w/ Hash Test	No Hashtags Test
Accuracy	0.670478	0.644345	0.643528	0.668028
Precision (Macro)	0.720947	0.408309	0.405924	0.661666
Recall (Macro)	0.427755	0.397422	0.396976	0.430052
F1 (Macro)	0.426585	0.38206	0.381482	0.431314

In [38]:
```
svm_compare.calc_scores(data_type='holdout')
```

In [39]:
```
svm_compare.score_comparison
```

Out[39]:

	Hashtags Holdout	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout	No Hashtags Holdout
Accuracy	0.69129	0.67806	0.689085	0.68688
Precision (Macro)	0.780238	0.430235	0.776801	0.611137
Recall (Macro)	0.438065	0.418686	0.431013	0.434494
F1 (Macro)	0.442035	0.409917	0.429046	0.437919

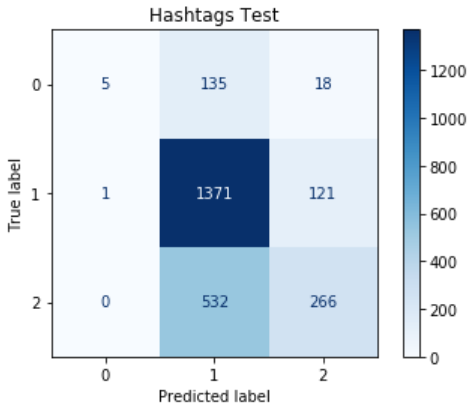
In [40]:
```
poly_compare.fit_models()
poly_compare.score_comparison
```

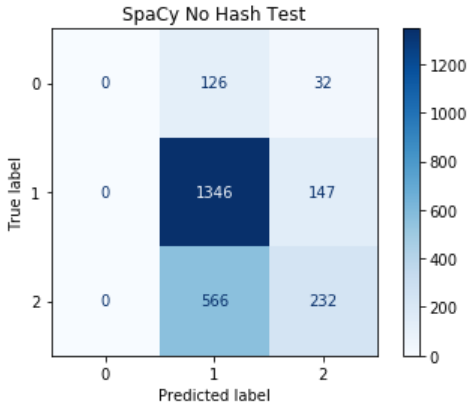
```
--- 219.18670511245728 seconds to process ---
```

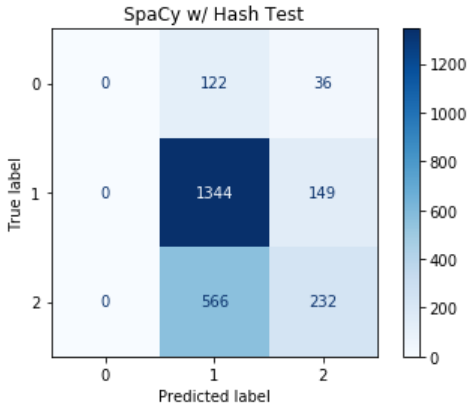
Out[40]:

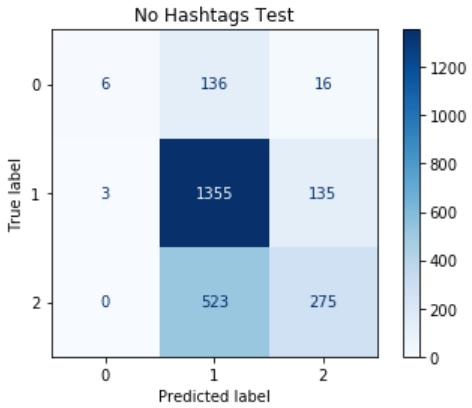
	Hashtags Test	SpaCy No Hash Test
Accuracy	0.632095	0.622703
Precision (Macro)	0.586477	0.44075
Recall (Macro)	0.410854	0.367584
F1 (Macro)	0.415803	0.338963

In [41]:
```
1 svm_compare.compare_confusion()
```

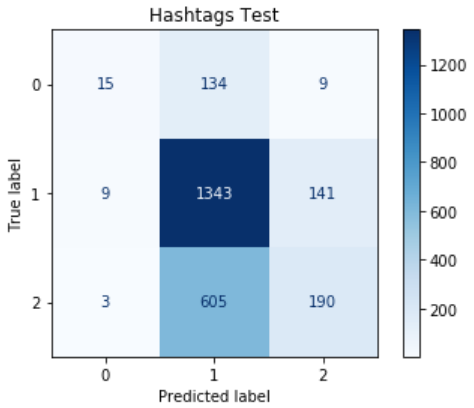


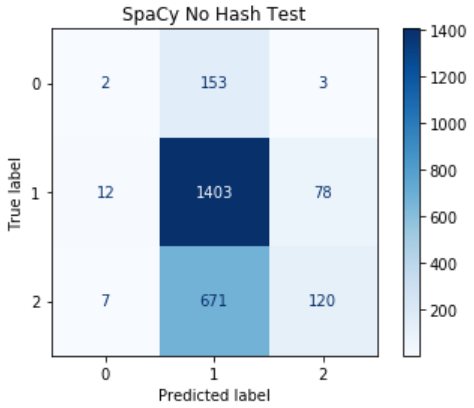






In [42]:
```
1 poly_compare.compare_confusion()
```





In [43]:
```
1 cnb_compare.fit_models()
2 cnb_compare.score_comparison
```

```
--- 0.12322688102722168 seconds to process ---
```

Out[43]:

	Hashtags Test	No Hashtags Test
Accuracy	0.546345	0.532871
Precision (Macro)	0.459053	0.447856
Recall (Macro)	0.495874	0.479016
F1 (Macro)	0.45676	0.441092

In [44]:
```
1 mnb_compare.fit_models()
2 mnb_compare.score_comparison
```

```
--- 0.1182107925415039 seconds to process ---
```

Out[44]:

	Hashtags Test	No Hashtags Test
Accuracy	0.651286	0.643528
Precision (Macro)	0.51229	0.523021
Recall (Macro)	0.411019	0.427527
F1 (Macro)	0.402957	0.434974

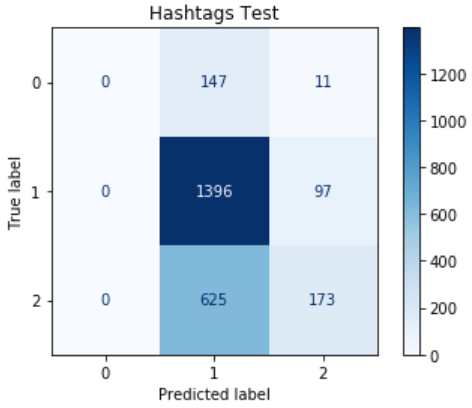
In [45]:
```
1 rf_compare.fit_models()
2 rf_compare.score_comparison
```

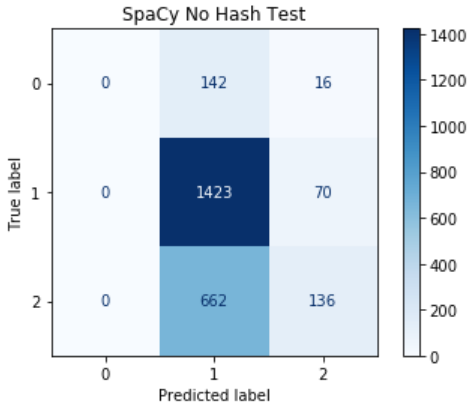
```
--- 7611.418187856674 seconds to process ---
```

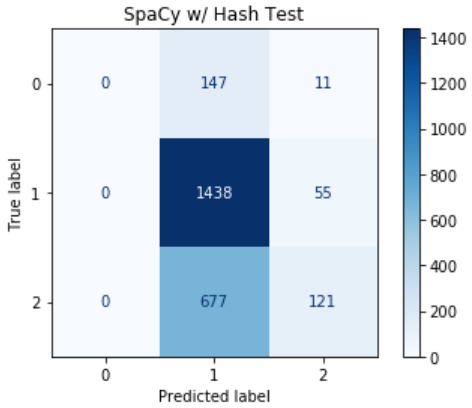
Out[45]:

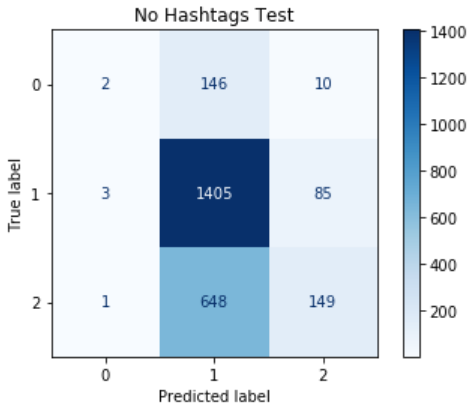
	Hashtags Test	SpaCy No Hash Test	SpaCy w/ Hash Test	No Hashtags Test
Accuracy	0.64067	0.636586	0.636586	0.635361
Precision (Macro)	0.419857	0.417196	0.427593	0.527639
Recall (Macro)	0.383941	0.374514	0.371597	0.380144
F1 (Macro)	0.3611	0.343907	0.337199	0.357161

In [46]:
```
rf_compare.compare_confusion()
rf_compare.calc_scores(data_type='holdout')
rf_compare.score_comparison
```









Out[46]:

	Hashtags Holdout	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout	No Hashtags Holdout
Accuracy	0.679162	0.665932	0.658214	0.684675

	Hashtags Holdout	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout	No Hashtags Holdout
Precision (Macro)	0.451056	0.44306	0.43118	0.621701
Recall (Macro)	0.400619	0.388044	0.378377	0.411281
F1 (Macro)	0.384856	0.368671	0.353923	0.402904

Observations

Out of these first attempts, the SVM is the best model by far but takes a long time to process. The Complement Naive Bayes does f
well on the TF-IDF with hashtags included and processes much faster. Thats said, it's clear that the TF-IDF vectorization method is
more accurate than the SpaCy Doc2Vec.

Tuning TF-IDF and Count Vectorizers

With both TF-IDF and Count vectorizors we can tune them to possibly improve the models. We have two tuning options:

- Play with number of features included (300 is default, try 200 and 400)
- Play with ngram size - Apple, Store vs. Apple Store

In order to test the effectiveness of these vectorization methods, we'll need to process our text again into new instances of the Text
class.

In [47]:
```
hashtags_200 = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='Hash200',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 43.44861316680908 seconds ---
```

In [48]:
```
hashtags_400 = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='Hash400',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 43.85856604576111 seconds ---
```

In [49]:
```
hashtags_2 = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='Hash2n',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 43.44369411468506 seconds ---
```

In [50]:
```
hashtags_3 = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='Hash3n',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 43.63354802131653 seconds ---
```

In [51]:
```
hashtags_final = TextSet(df['tweet_text'],
                   df['is_there_an_emotion_directed_at_a_brand_or_product'],
                   name='HashFinal',
                   keep_hashtags=True,
                   random_seed=42,
                   split=.3)
```

```
--- 43.42817401885986 seconds ---
```

In [52]:
```
hashtags_200.vectorize(max_features=200)
hashtags_400.vectorize(max_features=400)
hashtags_2.vectorize(ngram_range=(1,2))
hashtags_3.vectorize(ngram_range=(1,3))
```

In [53]:
```
hash_test = [hashtags_200, hashtags_400, hashtags_2, hashtags_3]
```

In [54]:
```
cnb_pipe = Pipeline([('scaler', MinMaxScaler()), ('cnb', ComplementNB())])
cnb_compare_2 = ModelComparison(pipeline=cnb_pipe, data_list=hash_test, y_format='label', name
```

In [55]:
```
cnb_compare_2.fit_models()
```

```
--- 0.17105388641357422 seconds to process ---
```

In [56]:
```
cnb_compare_2.score_comparison
```

Out[56]:

	Hash200 Test	Hash400 Test	Hash2n Test	Hash3n Test
Accuracy	0.538587	0.551245	0.516537	0.504287
Precision (Macro)	0.447028	0.461759	0.44972	0.447944
Recall (Macro)	0.474071	0.499137	0.485546	0.476552
F1 (Macro)	0.442801	0.460697	0.438042	0.426779

In [57]:
```
mnb_pipe = Pipeline([('scaler', MinMaxScaler()), ('mnb', MultinomialNB())])
mnb_compare_2 = ModelComparison(pipeline=mnb_pipe, data_list=hash_test, y_format='label', name
```

In [58]:
```
mnb_compare_2.fit_models()
mnb_compare_2.score_comparison
```

```
--- 0.2067122459411621 seconds to process ---
```

Out[58]:

	Hash200 Test	Hash400 Test	Hash2n Test	Hash3n Test
Accuracy	0.642303	0.649245	0.633728	0.614537
Precision (Macro)	0.491855	0.59684	0.465025	0.442663
Recall (Macro)	0.397998	0.430049	0.424267	0.416807
F1 (Macro)	0.385102	0.437085	0.425717	0.414438

In [59]:
```
hashtags_final.vectorize(max_features=500, ngram_range=(1,2))
```

In [60]:
```
mnb_pipe = Pipeline([('scaler', MinMaxScaler()), ('mnb', MultinomialNB())])
mnb_compare_3 = ModelComparison(pipeline=mnb_pipe, data_list=[hashtags_final], y_format='label
```

In [61]:
```
mnb_compare_3.fit_models()
mnb_compare_3.all_scores
```

```
--- 0.07824325561523438 seconds to process ---
```

Out[61]:

	HashFinal Train	HashFinal Test
Accuracy	0.677984	0.642303
Precision (Macro)	0.578405	0.477687
Recall (Macro)	0.478691	0.432729
F1 (Macro)	0.495919	0.436771

In [62]:
```
mnb_compare_3.calc_scores(data_type='holdout')
mnb_compare_3.all_scores
```

Out[62]:

	HashFinal Train	HashFinal Holdout
Accuracy	0.677984	0.647189
Precision (Macro)	0.578405	0.507965
Recall (Macro)	0.478691	0.446428
F1 (Macro)	0.495919	0.457767

In [63]:
```
hashtags_200.vectorize(method='count', max_features=200)
hashtags_400.vectorize(method='count', max_features=400)
hashtags_2.vectorize(method='count', ngram_range=(1,2))
hashtags_3.vectorize(method='count', ngram_range=(1,3))
```

In [64]:
```
count_list = [hashtags_200, hashtags_400, hashtags_2, hashtags_3]
```

In [65]:
```
mnb_compare_4 = ModelComparison(pipeline=mnb_pipe, data_list=count_list, y_format='label', nam
mnb_compare_4.fit_models()
mnb_compare_4.score_comparison
```

```
--- 0.17790913581848145 seconds to process ---
```

Out[65]:

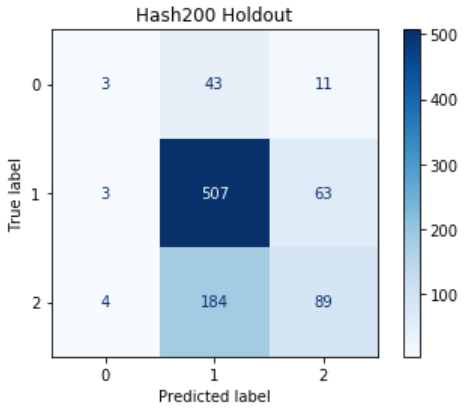
	Hash200 Test	Hash400 Test	Hash2n Test	Hash3n Test
Accuracy	0.64312	0.643936	0.646795	0.646795
Precision (Macro)	0.520981	0.527562	0.537196	0.537196
Recall (Macro)	0.41165	0.453284	0.441468	0.441468
F1 (Macro)	0.410784	0.467934	0.454056	0.454056

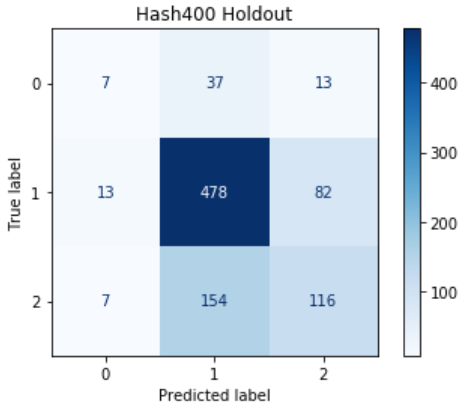
In [66]:
```
mnb_compare_4.calc_scores(data_type='holdout')
mnb_compare_4.score_comparison
```

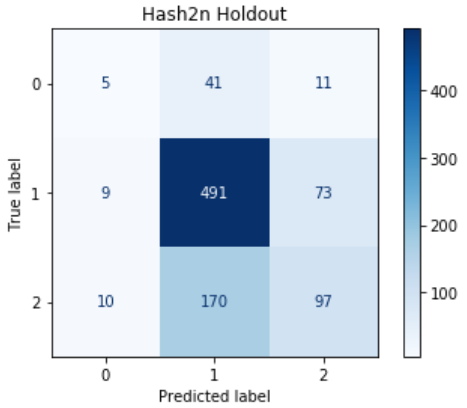
Out[66]:

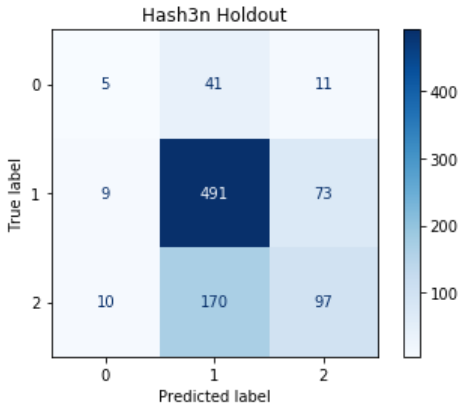
	Hash200 Holdout	Hash400 Holdout	Hash2n Holdout	Hash3n Holdout
Accuracy	0.660419	0.662624	0.653804	0.653804
Precision (Macro)	0.512249	0.507841	0.481225	0.481225
Recall (Macro)	0.419583	0.458595	0.431598	0.431598
F1 (Macro)	0.423307	0.470601	0.439078	0.439078

In [67]:
```
1 mnb_compare_4.compare_confusion(data_type='holdout')
```









Observations

In the end, tuning the number of features didn't increase the score by much. Let's see if we can improve our accuracy with the Sequ
Neural Network model from Keras/TensorFlow.

Neural Network Models

First, we create a baseline model with 3 layers, each using the 'relu' activation method with a final output using the softmax activatio
sort the values into 3 classes. Because the TF-IDF and Count vectorized datasets have an input shape of (300,) and SpaCy vectoriz
has (96,) we will need to analyze those pairs of datasets separately.

In [68]:
```
hashtags_nn = TextSet(df['tweet_text'],
                      df['is_there_an_emotion_directed_at_a_brand_or_product'],
                      name='Hashtags',
                      keep_hashtags=True,
                      random_seed=42,
                      split=.3)
```

```
--- 43.560139894485474 seconds ---
```

In [69]:
```
spacy_1_nn = TextSet(df['tweet_text'],
                     df['is_there_an_emotion_directed_at_a_brand_or_product'],
                     name = 'SpaCy No Hash',
                     is_spacy=True,
                     keep_stopwords=True,
                     random_seed=42,
                     split=.3)
```

```
--- 42.46915888786316 seconds ---
```

In [70]:
```
spacy_2_nn = TextSet(df['tweet_text'],
                     df['is_there_an_emotion_directed_at_a_brand_or_product'],
                     name = 'SpaCy w/ Hash',
                     is_spacy=True,
                     keep_stopwords=True,
                     keep_hashtags=True,
                     random_seed=42,
                     split=.3)
```

```
--- 44.283653259277344 seconds ---
```

In [71]:
```
no_hash_nn = TextSet(df['tweet_text'],
                     df['is_there_an_emotion_directed_at_a_brand_or_product'],
                     name='No Hashtags',
                     random_seed=42,
                     split=.3)
```

```
--- 42.08311414718628 seconds ---
```

In [72]:
```
model_spacy = Sequential()

model_spacy.add(Dense(75, activation='relu', input_shape=(96,)))
model_spacy.add(Dense(50, activation='relu'))
model_spacy.add(Dense(3, activation='softmax'))

model_spacy.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc']
)
```

In [73]:
```
model_tfidf = Sequential()

model_tfidf.add(Dense(75, activation='relu', input_shape=(300,)))
model_tfidf.add(Dense(50, activation='relu'))
model_tfidf.add(Dense(3, activation='softmax'))

model_tfidf.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc']
)
```

Since we are not using the Pipeline object to standardize the vectors, we will use the StandardScaler built into the TextSet Class to regularize the vectors to improve Neural Net performance.

In [74]:
```
spacy_1_nn.vectorize()
spacy_1_nn.regularize()

spacy_2_nn.vectorize()
spacy_2_nn.regularize()
```

In [75]:
```
hashtags_nn.vectorize()
hashtags_nn.regularize()

no_hash_nn.vectorize()
no_hash_nn.regularize()
```

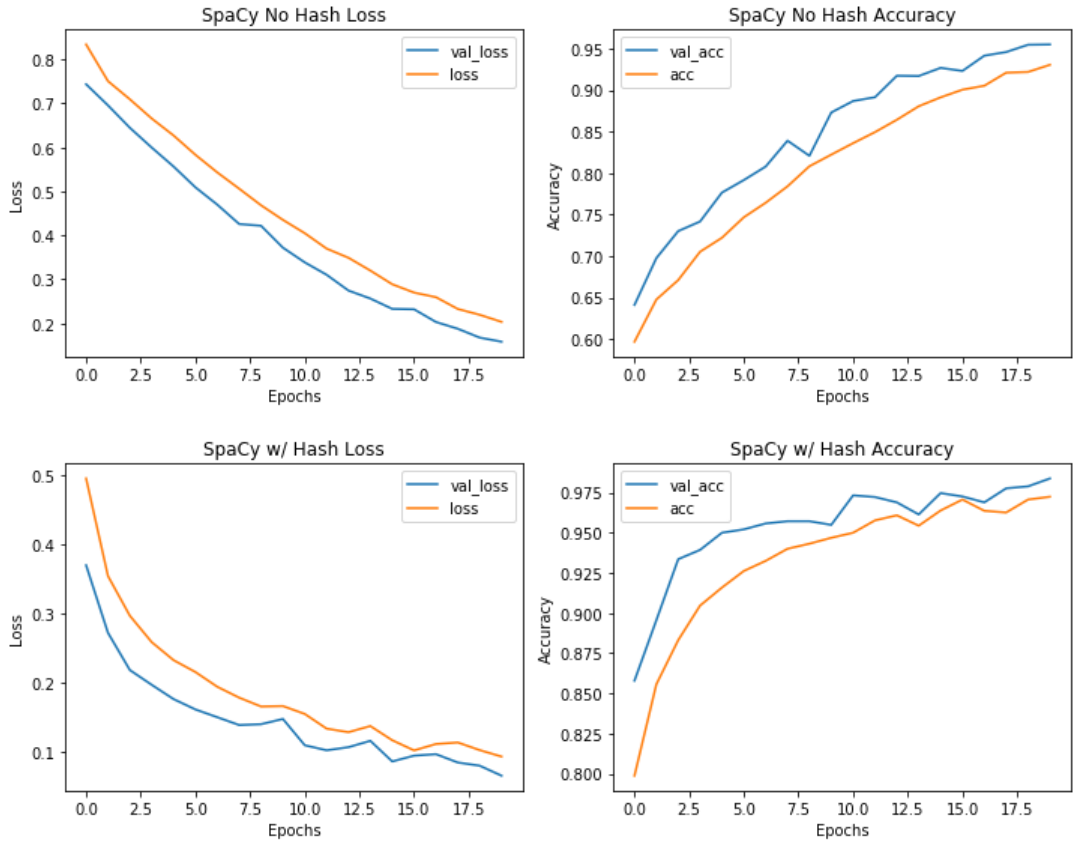
In [76]:
```
spacy_nn = [spacy_1_nn, spacy_2_nn]
tfidf_nn = [hashtags_nn, no_hash_nn]
```

In [77]:
```
nn_compare_s = ModelComparison(pipeline=model_spacy, data_list=spacy_nn, y_format='ohe', name=
nn_compare_tf = ModelComparison(pipeline=model_tfidf, data_list=tfidf_nn, y_format='ohe', name=
```

In [78]:
```
nn_compare_s.fit_models()
```

```
Epoch 1/20
381/381 [==============================] - 16s 3ms/step - loss: 0.8831 - acc: 0.5655 - val_loss: 0
6 - val_acc: 0.6412
Epoch 2/20
381/381 [==============================] - 1s 2ms/step - loss: 0.7505 - acc: 0.6482 - val_loss: 0.
- val_acc: 0.6976
Epoch 3/20
381/381 [==============================] - 1s 2ms/step - loss: 0.7039 - acc: 0.6819 - val_loss: 0.
- val_acc: 0.7301
Epoch 4/20
381/381 [==============================] - 1s 2ms/step - loss: 0.6583 - acc: 0.7177 - val_loss: 0.
- val_acc: 0.7417
Epoch 5/20
381/381 [==============================] - 1s 2ms/step - loss: 0.6176 - acc: 0.7254 - val_loss: 0.
- val_acc: 0.7765
Epoch 6/20
381/381 [==============================] - 1s 2ms/step - loss: 0.5716 - acc: 0.7537 - val_loss: 0.
- val_acc: 0.7916
Epoch 7/20
```

In [79]:
```
1 nn_compare_s.plot_val_history()
```



In [80]:
```
1 nn_compare_s.score_comparison
```

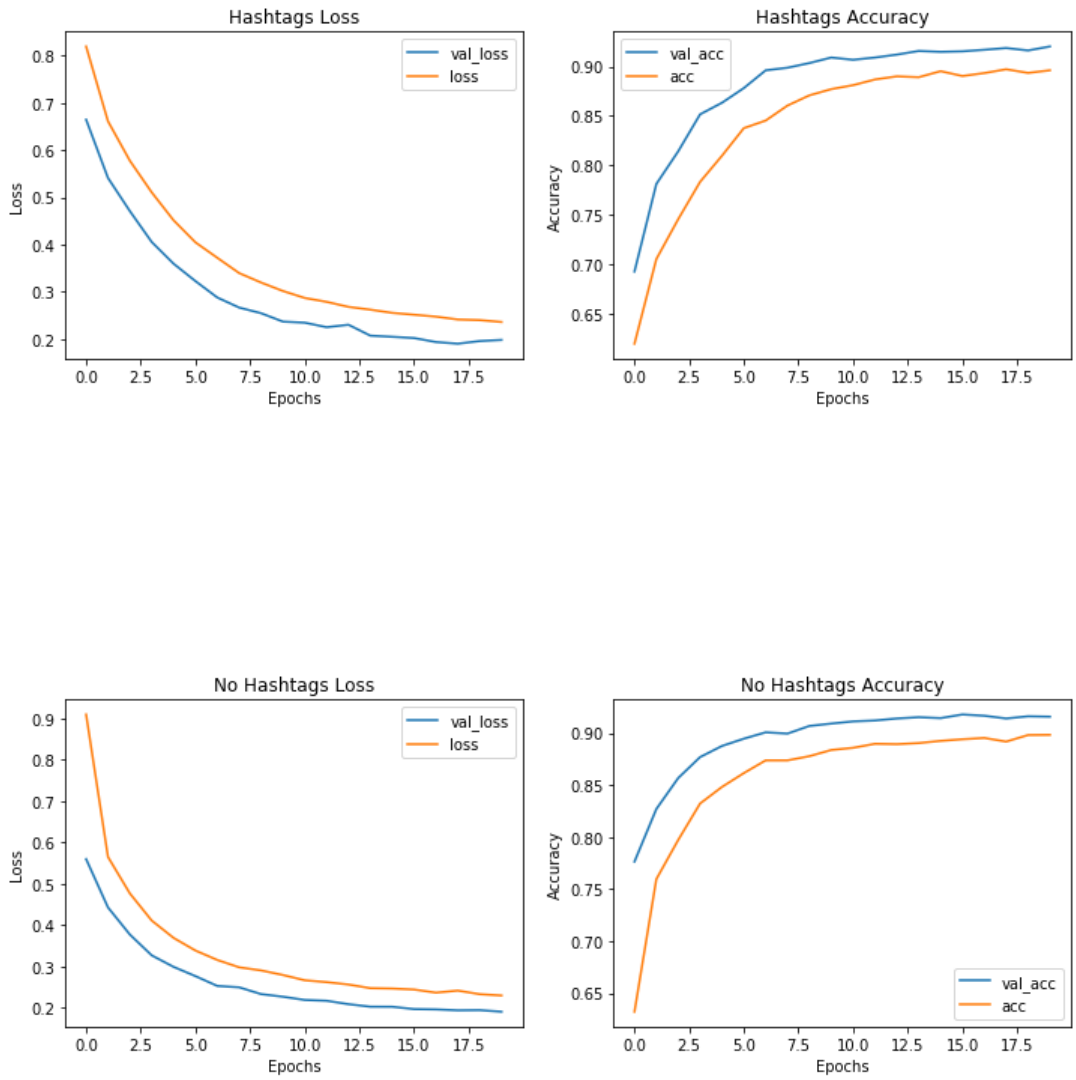
Out[80]:

	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	1.303893	1.819867
Accuracy	0.624035	0.638368

In [81]:
```
1 nn_compare_tf.fit_models()
```

```
Epoch 1/20
381/381 [==============================] - 2s 2ms/step - loss: 0.8707 - acc: 0.5926 - val_loss: 0.
- val_acc: 0.6923
Epoch 2/20
381/381 [==============================] - 1s 2ms/step - loss: 0.6542 - acc: 0.7076 - val_loss: 0.
- val_acc: 0.7811
Epoch 3/20
381/381 [==============================] - 1s 2ms/step - loss: 0.5493 - acc: 0.7664 - val_loss: 0.
- val_acc: 0.8141
Epoch 4/20
381/381 [==============================] - 1s 2ms/step - loss: 0.4918 - acc: 0.7998 - val_loss: 0.
- val_acc: 0.8516
Epoch 5/20
381/381 [==============================] - 1s 2ms/step - loss: 0.4317 - acc: 0.8226 - val_loss: 0.
- val_acc: 0.8633
Epoch 6/20
381/381 [==============================] - 1s 2ms/step - loss: 0.3839 - acc: 0.8513 - val_loss: 0.
- val_acc: 0.8782
Epoch 7/20
```

In [82]:
```
1 nn_compare_tf.plot_val_history()
```



In [83]:
```
1 nn_compare_tf.score_comparison
```

Out[83]:

	Hashtags Holdout	No Hashtags Holdout
Loss	1.165687	1.277036
Accuracy	0.671444	0.644983

Immediately we can see the same trend as our other models, namely that the validation data has higher accuracy than the training d
This is surprising but is most likely due to the noise in the training data more than anything. Let's tune our neural nets

Tuning Neural Networks

There are countless combinations of layers and methods that can be used to incrementally increase model accuracy. I will only be f
on three methods to see how the output is affected:

- Add drop out layer(s)
- Increase nodes per layer

- Increase number of layers

In [84]:
```
model_sb_1 = Sequential()

model_sb_1.add(Dense(100, activation='relu', input_shape=(96,)))
model_sb_1.add(Dropout(.3))
model_sb_1.add(Dense(75, activation='relu'))
model_sb_1.add(Dropout(.3))
model_sb_1.add(Dense(3, activation='softmax'))

model_sb_1.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

In [85]:
```
model_sb_2 = Sequential()

model_sb_2.add(Dense(150, activation='relu', input_shape=(96,)))
model_sb_2.add(Dense(50, activation='relu'))
model_sb_2.add(Dropout(.3))
model_sb_2.add(Dense(75, activation='relu'))
model_sb_2.add(Dense(3, activation='softmax'))

model_sb_2.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

In [86]:
```
model_tfb_1 = Sequential()

model_tfb_1.add(Dense(100, activation='relu', input_shape=(300,)))
model_tfb_1.add(Dropout(.3))
model_tfb_1.add(Dense(75, activation='relu'))
model_tfb_1.add(Dropout(.3))
model_tfb_1.add(Dense(3, activation='softmax'))

model_tfb_1.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

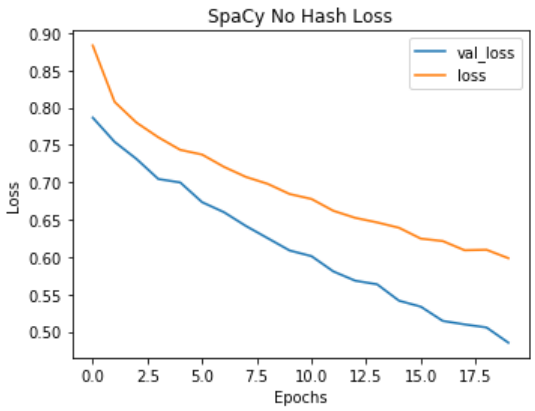
In [87]:
```
model_tfb_2 = Sequential()

model_tfb_2.add(Dense(150, activation='relu', input_shape=(300,)))
model_tfb_2.add(Dense(50, activation='relu'))
model_tfb_2.add(Dropout(.3))
model_tfb_2.add(Dense(75, activation='relu'))
model_tfb_2.add(Dense(3, activation='softmax'))

model_tfb_2.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

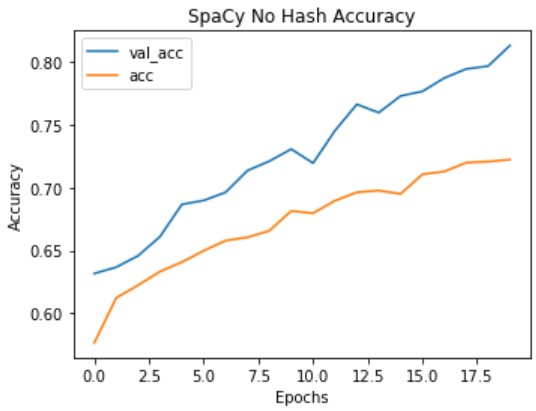
In [88]:
```
nn_sb_1 = ModelComparison(pipeline=model_sb_1, data_list=spacy_nn, y_format='ohe', name='nn_sb
nn_sb_2 = ModelComparison(pipeline=model_sb_2, data_list=spacy_nn, y_format='ohe', name='nn_sb
nn_tfb_1 = ModelComparison(pipeline=model_tfb_1, data_list=tfidf_nn, y_format='ohe', name='nn_
nn_tfb_2 = ModelComparison(pipeline=model_tfb_2, data_list=tfidf_nn, y_format='ohe', name='nn_
```

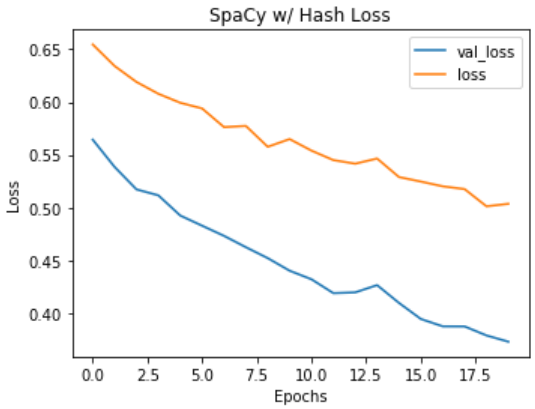
In [89]:
```
nn_sb_1.fit_models()
```

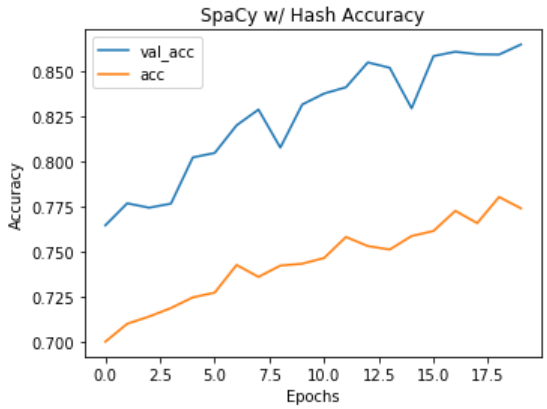
...

In [172]:
```
1 nn_sb_1.plot_val_history()
2 nn_sb_1.score_comparison
```









Out[172]:

	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	0.771601	0.817499
Accuracy	0.653804	0.667034

In [90]:
```
1 nn_sb_1.score_comparison
```

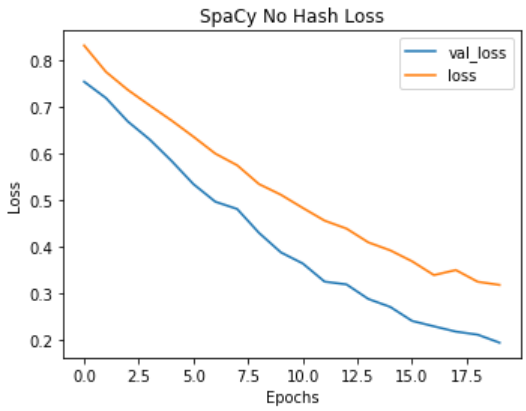
Out[90]:

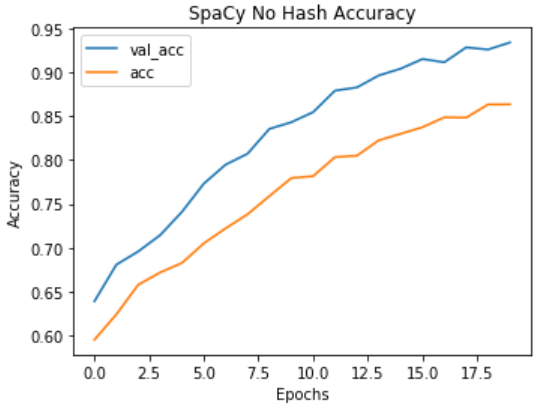
	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	0.771601	0.817499
Accuracy	0.653804	0.667034

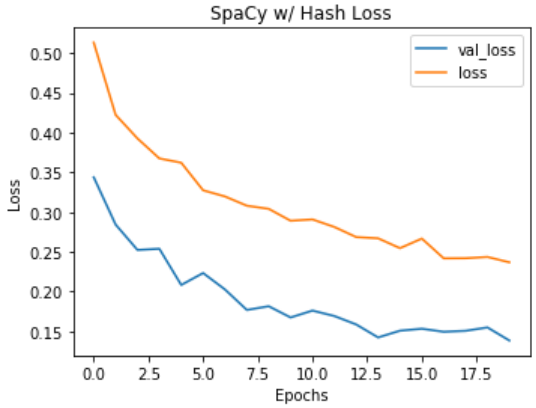
In [91]:
```
1 nn_sb_2.fit_models()
```

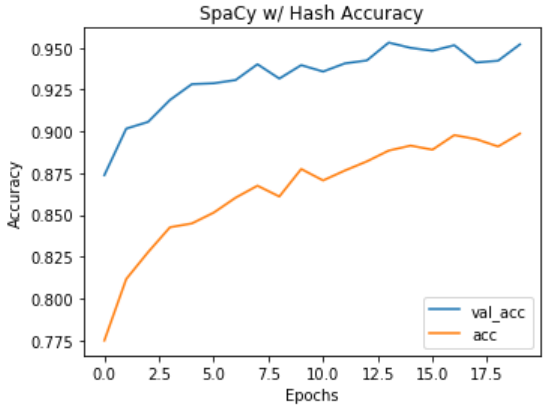
...

In [171]:
```
1 nn_sb_2.plot_val_history()
2 nn_sb_2.score_comparison
```









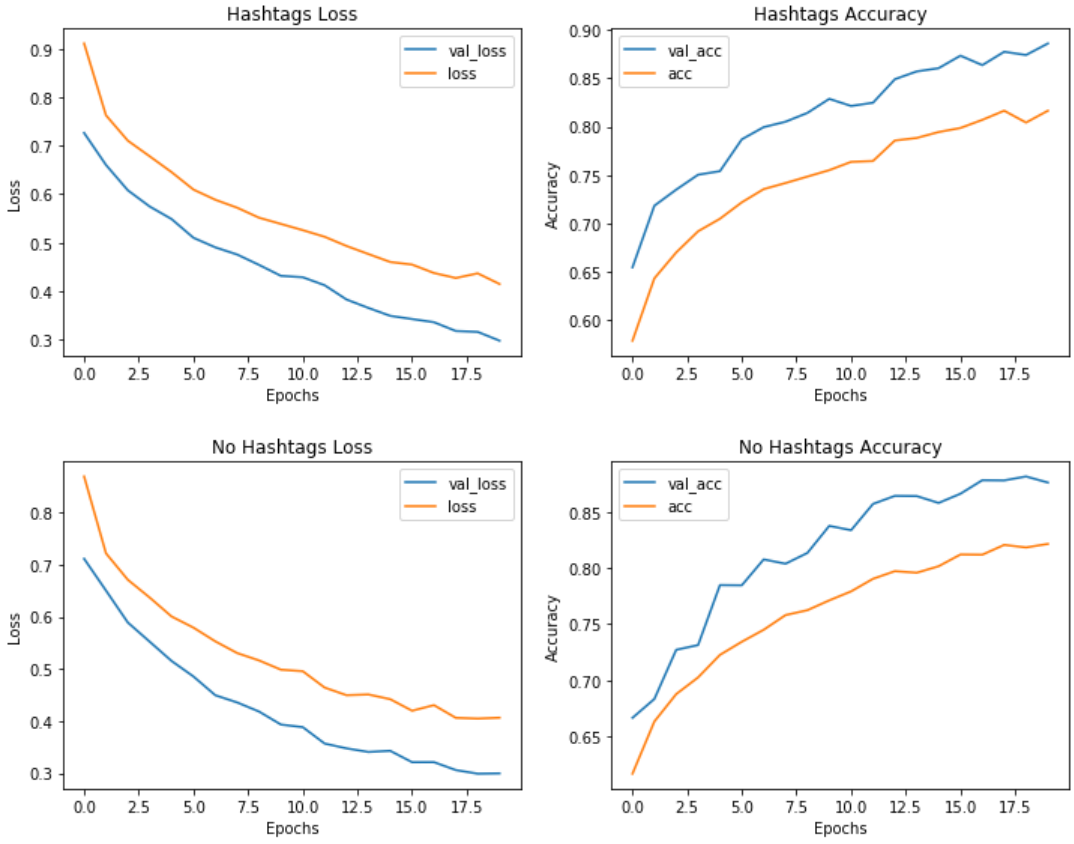
Out[171]:

	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	1.267259	1.635173
Accuracy	0.615215	0.630651

In [92]:
```
1 nn_tfb_1.fit_models()
```

...

In [170]:
```
1 nn_tfb_1.plot_val_history()
2 nn_tfb_1.score_comparison
```



Out[170]:

	Hashtags Holdout	No Hashtags Holdout
Loss	0.863775	0.870073
Accuracy	0.672547	0.690187

In [93]:
```
1 nn_tfb_2.fit_models()
```

...

In [169]:
```
1 nn_tfb_2.plot_val_history()
2 nn_tfb_2.score_comparison
```

...

Observations

Once again, the TF-IDF with Hashtags data set reigns supreme. Notably, the model with the most layers and only a single dropout la nn_tfb_1, has the best accuracy compared to the other models. We have one more step to try to increase accuracy.

Regularization

Once again, there are multiple methods we can use to regularize our vectors, L1, L2 and combination of both. I will see how basic regularization affects the output. For this set, I will also be increasing the number of epochs and the batch size of the data to see if t affects anything as well.

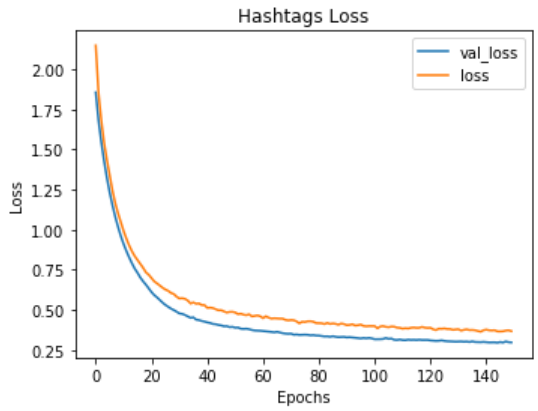
In [94]:
```
model_tfb_3 = Sequential()

model_tfb_3.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.005), input
model_tfb_3.add(Dropout(.3))
model_tfb_3.add(Dense(75,kernel_regularizer=regularizers.l2(0.005), activation='relu'))
model_tfb_3.add(Dropout(.3))
model_tfb_3.add(Dense(3, activation='softmax'))

model_tfb_3.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

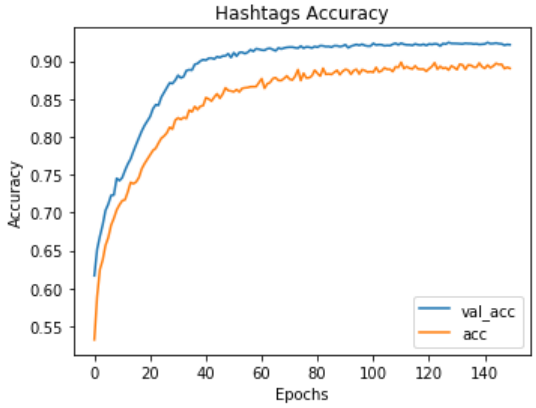
In [95]:
```
model_tfb_4 = Sequential()

model_tfb_4.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l1(0.005), input
model_tfb_4.add(Dropout(.3))
model_tfb_4.add(Dense(75, activation='relu', kernel_regularizer=regularizers.l1(0.005)))
model_tfb_4.add(Dropout(.3))
model_tfb_4.add(Dense(3, activation='softmax'))

model_tfb_4.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

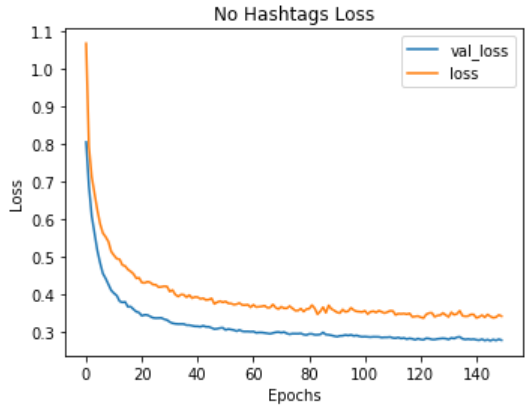
In [96]:
```
nn_tfb_3 = ModelComparison(pipeline=model_tfb_3, data_list=tfidf_nn, y_format='ohe', name='nn_
nn_tfb_3.fit_models(batch_size=256, epochs=150)
```

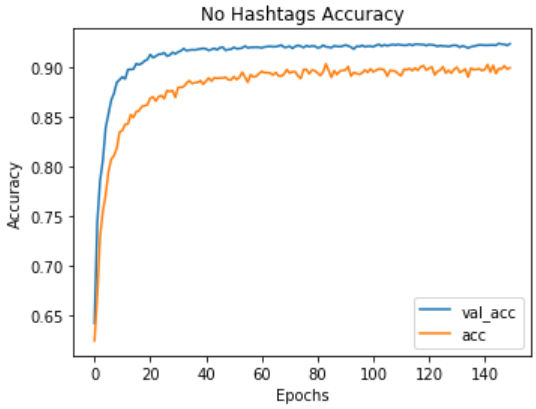
...

In [168]:
```
nn_tfb_3.plot_val_history()
nn_tfb_3.score_comparison
```









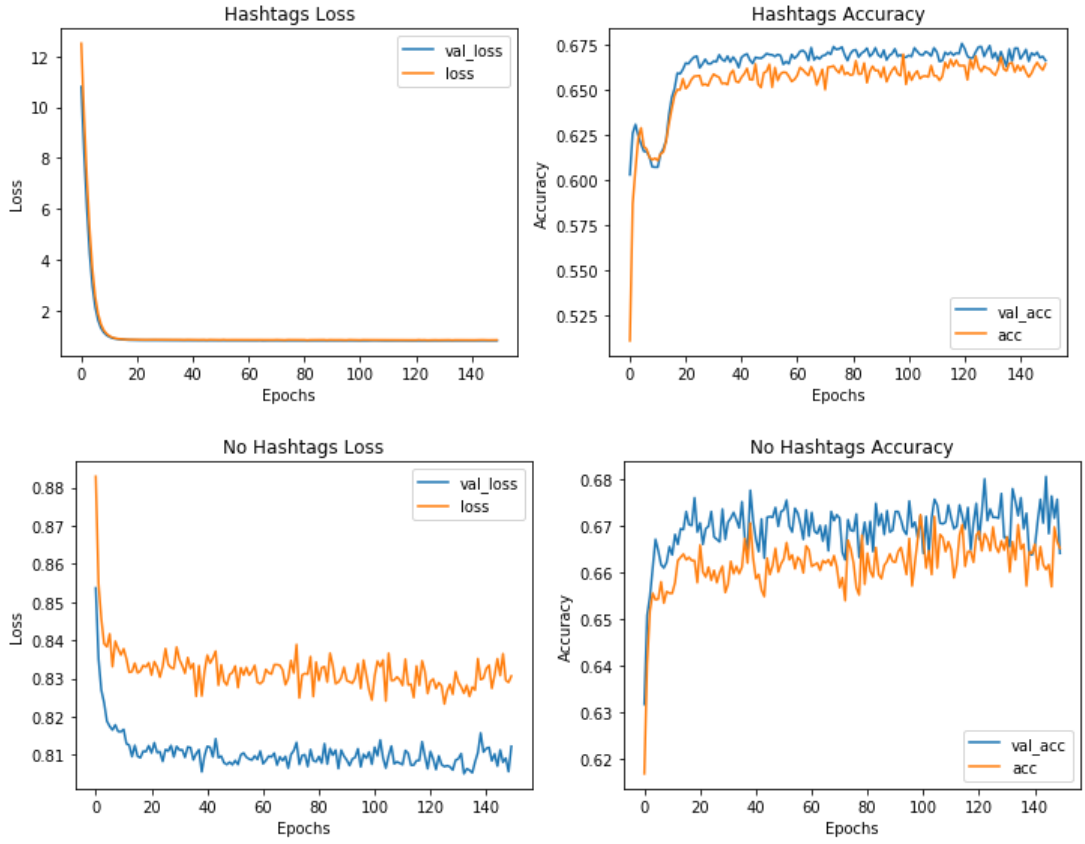
Out[168]:

	Hashtags Holdout	No Hashtags Holdout
Loss	1.076800	1.184689
Accuracy	0.679162	0.686880

In [97]:
```
nn_tfb_4 = ModelComparison(pipeline=model_tfb_4, data_list=tfidf_nn, y_format='ohe', name='nn_
nn_tfb_4.fit_models(batch_size=256, epochs=150)
```

...

In [167]:
```
nn_tfb_4.plot_val_history()
nn_tfb_4.score_comparison
```



Out[167]:

	Hashtags Holdout	No Hashtags Holdout
Loss	0.822676	0.823290
Accuracy	0.672547	0.671444

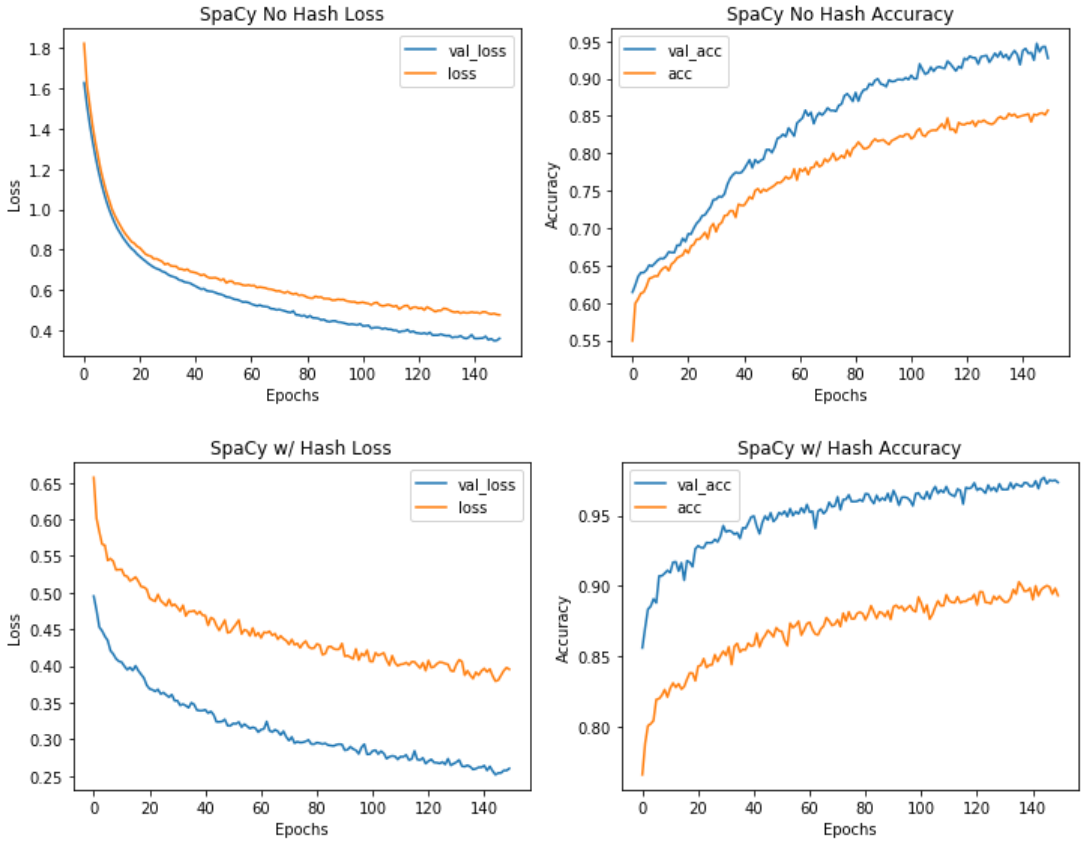
In [98]:
```
model_sb_3 = Sequential()

model_sb_3.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.005), input_
model_sb_3.add(Dropout(.3))
model_sb_3.add(Dense(75,kernel_regularizer=regularizers.l2(0.005), activation='relu'))
model_sb_3.add(Dropout(.3))
model_sb_3.add(Dense(3, activation='softmax'))

model_sb_3.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

In [99]:
```
model_sb_4 = Sequential()

model_sb_4.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l1(0.005), input_
model_sb_4.add(Dropout(.3))
model_sb_4.add(Dense(75, activation='relu', kernel_regularizer=regularizers.l1(0.005)))
model_sb_4.add(Dropout(.3))
model_sb_4.add(Dense(3, activation='softmax'))

model_sb_4.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

In [100]:
```
nn_sb_3 = ModelComparison(pipeline=model_sb_3, data_list=spacy_nn, y_format='ohe', name='nn_sb
nn_sb_3.fit_models(batch_size=256, epochs=150)
nn_sb_3.plot_val_history()
nn_sb_3.score_comparison
```

...

In [165]:
```
nn_sb_3.plot_val_history()
nn_sb_3.score_comparison
```



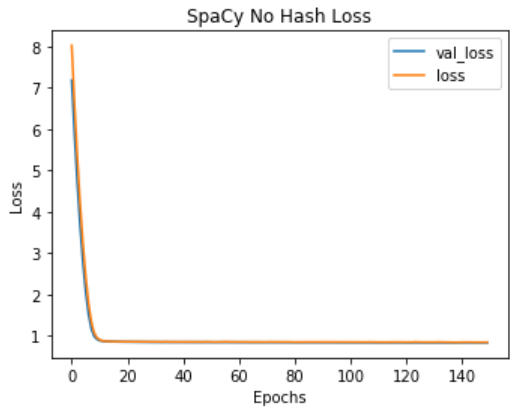
Out[165]:

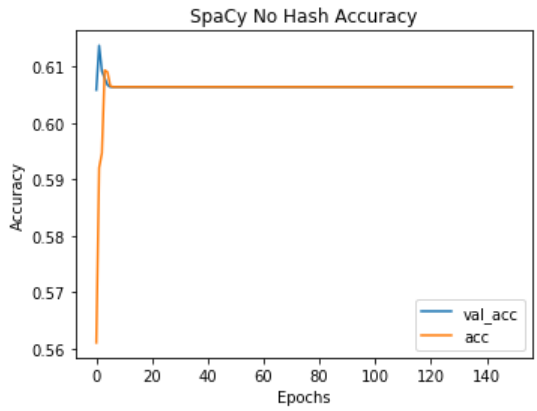
	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	1.011510	1.195369
Accuracy	0.672547	0.644983

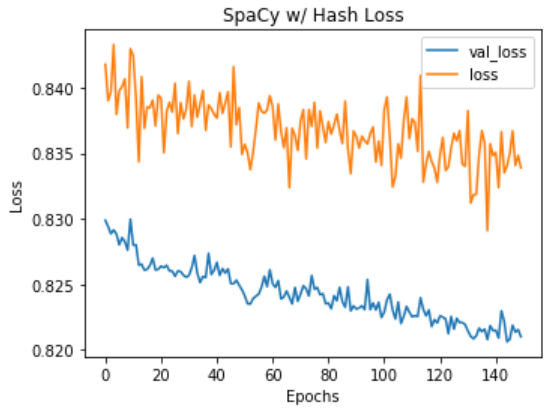
In [101]:
```
nn_sb_4 = ModelComparison(pipeline=model_sb_4, data_list=spacy_nn, y_format='ohe', name='nn_sb
nn_sb_4.fit_models(batch_size=256, epochs=150)

```

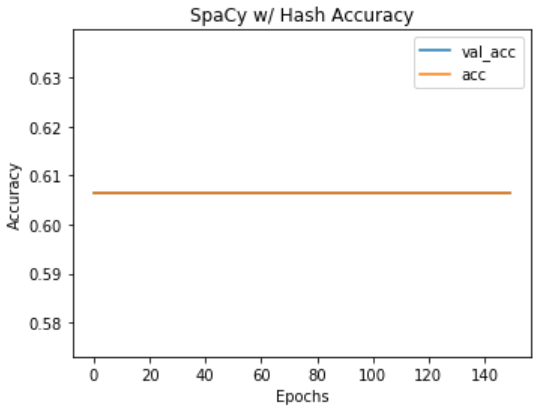
...

In [166]:
```
1 nn_sb_4.plot_val_history()
2 nn_sb_4.score_comparison
```









Out[166]:

	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	0.822630	0.806852
Accuracy	0.631753	0.631753

Observations

We can see that the L1 regularization might be viable, though it seems to only improve the model incrementally. the L2 regularizatior the other hand, had somewhat unpredictable effects, especially on the SpaCy datasets.

Best Models:

Ultimately, with some basic tuning, we managed to reach a holdout accuracy of about 70%, about a 10% increase over the baseline accuracy. This isn't ideal, but it is 10% better than a blind guess. The two models that had the best scores were:

- Neural Network with TF-IDF Plus Hashtags
- Support Vector Machine with TF-IDF Plus Hashtags

In [102]:
```
1 nn_tfb_1.score_comparison #Best model!
```

Out[102]:

	Hashtags Holdout	No Hashtags Holdout
Loss	0.863775	0.870073
Accuracy	0.672547	0.690187

In [103]:
```
1 svm_compare.score_comparison # Second Best Model!
```

Out[103]:

	Hashtags Holdout	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout	No Hashtags Holdout
Accuracy	0.69129	0.67806	0.689085	0.68688
Precision (Macro)	0.780238	0.430235	0.776801	0.611137
Recall (Macro)	0.438065	0.418686	0.431013	0.434494
F1 (Macro)	0.442035	0.409917	0.429046	0.437919

Part 4: Detecting the Products

In order to detect the products mentioned in the tweets, we will use the exact same methodology and process. There are some key differences:

- The training dataset is almost a third of the size of the Sentiment Analysis training data
- There is no category for 'No Product Mentioned'
- We can attempt to use string methods as well

I split the dataframe into labeled products and non-labeled products. Ultimately, we want to predict the products detected in the 50 tweets that have not been labeled yet.

In [104]:
```
1 known_df['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts()
```

Out[104]:
```
Positive emotion                       2672
Negative emotion                        519
No emotion toward brand or product       91
I can't tell                              9
Name: is_there_an_emotion_directed_at_a_brand_or_product, dtype: int64
```

In [105]:
```
1 unknown_df['is_there_an_emotion_directed_at_a_brand_or_product'].value_counts()
```

Out[105]:
```
No emotion toward brand or product    5298
Positive emotion                       306
I can't tell                           147
Negative emotion                        51
Name: is_there_an_emotion_directed_at_a_brand_or_product, dtype: int64
```

In [106]:
```
known_df
```

Out[106]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_pr
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative em
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive em
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive em
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative em
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive em
...	...	...	
9077	@mention your PR guy just convinced me to swit...	iPhone	Positive em
9079	"papyrus...sort of like the ipad" - ...	iPad	Positive em
9080	Diller says Google TV "might be run over ...	Other Google product or service	Negative em
9085	I've always used Camera+ for my iPhone b/c it ...	iPad or iPhone App	Positive em
9088	Ipad everywhere. #SXSW {link}	iPad	Positive em

In [107]:
```
known_df['emotion_in_tweet_is_directed_at'].value_counts(normalize=True)
```

Out[107]:
```
iPad                               0.287451
Apple                              0.200851
iPad or iPhone App                 0.142814
Google                             0.130659
iPhone                             0.090246
Other Google product or service    0.089031
Android App                        0.024613
Android                            0.023701
Other Apple product or service     0.010635
Name: emotion_in_tweet_is_directed_at, dtype: float64
```

In [108]:
```
prod_hash = TextSet(known_df['tweet_text'],
                    known_df['emotion_in_tweet_is_directed_at'],
                    name='Hashtags',
                    keep_hashtags=True,
                    random_seed=42,
                    split=.3)
```

```
--- 16.44481611251831 seconds ---
```

In [109]:
```
prod_sp_1 = TextSet(known_df['tweet_text'],
                    known_df['emotion_in_tweet_is_directed_at'],
                    name = 'SpaCy No Hash',
                    is_spacy=True,
                    keep_stopwords=True,
                    random_seed=42,
                    split=.3)
```

```
--- 16.101919174194336 seconds ---
```

In [110]:
```
prod_sp_2 = TextSet(known_df['tweet_text'],
                    known_df['emotion_in_tweet_is_directed_at'],
                    name = 'SpaCy w/ Hash',
                    is_spacy=True,
                    keep_stopwords=True,
                    keep_hashtags=True,
                    random_seed=42,
                    split=.3)
```

```
--- 16.12209987640381 seconds ---
```

In [111]:
```
prod_no_hash = TextSet(known_df['tweet_text'],
                    known_df['emotion_in_tweet_is_directed_at'],
                    name='No Hashtags',
                    random_seed=42,
                    split=.3)
```

```
--- 15.754395961761475 seconds ---
```

In [112]:
```
X = known_df['tweet_text']
y = known_df['emotion_in_tweet_is_directed_at']
```

In [113]:
```
labeler = LabelEncoder()
y_labels = labeler.fit_transform(y)
```

In [114]:
```
pd.DataFrame([y_labels, known_df['emotion_in_tweet_is_directed_at']]).T.rename(columns={0:'Y_I
```

Out[114]:

	Y_Label	Product
0	8	iPhone
1	7	iPad or iPhone App
2	6	iPad
3	7	iPad or iPhone App
4	3	Google
...	...	...
3286	8	iPhone
3287	6	iPad
3288	5	Other Google product or service
3289	7	iPad or iPhone App
3290	6	iPad

3291 rows × 2 columns

Approach 1: String Methods

There are a number of ways of extracting information using RegEx and none of them use machine learning. Below is a very basic de tree that could be used to identify products in a tweet. Notice that it defaults to 'Other Google Product or Service" if none of the oth options are detected. This means this function would be essentially useless if tweets with zero products mentioned are included in t datatset

In [115]:
```
def find_product(tweet):

    '''
    Identifies products in tweets

    Parameters
    ----------
    tweet : String
        The text to be analyzed

    Returns
    -------
    answer : String
        The name of the product identified in the tweet
    '''

    answer = ''

    if 'apple' in tweet.lower():
        answer = 'Apple'
    elif 'app' in tweet.lower() and ('iphone' in tweet.lower() or 'ipad' in tweet.lower()):
        answer = 'iPad or iPhone App'
    elif 'iphone' in tweet.lower():
        answer = 'iPhone'
    elif 'ipad' in tweet.lower():
        answer = 'iPad'
    elif 'app' in tweet.lower() and 'android':
        answer = 'Android App'
    elif 'google' in tweet.lower():
        answer = 'Google'
    elif 'android' or 'samsung' or 'galaxy' in tweet.lower():
        answer = 'Android'
    else:
        answer = 'Other Google product or service'

    return answer
```

In [116]:
```
known_df['test'] = known_df['tweet_text'].map(lambda x: find_product(x))
```

In [117]:
```
y_preds = known_df['test']
y_preds_labels = labeler.transform(y_preds)
```

In [118]:
```
accuracy_score(y_labels, y_preds_labels)
```

Out[118]:
```
0.7441507140686722
```

This is interesting. Our baseline accuracy on this product problem is technically 29% (blindly guessing the majority class, which is 'i
However, with even a basic decision tree function, I achieved 75% accuracy. Therefore, the goal of modeling this problem is that be

Approach 2: Modeling

We will use exactly the same vectorization methods and models to complete this half of the problem.

In [119]:
```
prod_data = [prod_hash, prod_no_hash, prod_sp_1, prod_sp_2]
prod_tfidf = [prod_hash, prod_no_hash]
```

In [120]:
```
prod_hash.vectorize(max_features=400)
prod_no_hash.vectorize(method="count")
prod_sp_1.vectorize()
prod_sp_2.vectorize()
```

In [142]:
```
svm_pipe = Pipeline([('scaler', StandardScaler()), ('rbf', svm.SVC())])

grid = {
    'rbf__kernel': ['rbf', 'sigmoid'],
    'rbf__gamma': ['scale', 'auto'],
    'rbf__C': [1, 100, 1e12],
    'rbf__decision_function_shape': ['ovo','ovr']
}

svm_grid_2 = GridSearchCV(svm_pipe, param_grid=grid, scoring='accuracy', cv=5)
svm_products = ModelComparison(pipeline=svm_grid, data_list=prod_data, y_format='label', name=
```

In [143]:
```
svm_products.fit_models()
svm_products.score_comparison
```

```
--- 76.68152976036072 seconds to process ---
```

Out[143]:

	Hashtags Test	No Hashtags Test	SpaCy No Hash Test	SpaCy w/ Hash Test
Accuracy	0.773903	0.773903	0.570304	0.5973
Precision (Macro)	0.711705	0.666167	0.632671	0.646521
Recall (Macro)	0.655901	0.613408	0.445267	0.457446
F1 (Macro)	0.67572	0.629462	0.486185	0.497012

In [123]:
```
mnb_products = ModelComparison(pipeline=mnb_pipe, data_list=prod_tfidf, y_format='label', name
mnb_products.fit_models()
mnb_products.score_comparison
```

```
--- 0.06243085861206055 seconds to process ---
```

Out[123]:

	Hashtags Test	No Hashtags Test
Accuracy	0.706412	0.701912
Precision (Macro)	0.674084	0.655906
Recall (Macro)	0.521732	0.509875
F1 (Macro)	0.538482	0.519875

In [144]:
```
svm_products.calc_scores(data_type='holdout')
svm_products.score_comparison
```

Out[144]:

	Hashtags Holdout	No Hashtags Holdout	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Accuracy	0.760606	0.778788	0.581818	0.581818
Precision (Macro)	0.723936	0.562171	0.582822	0.514096
Recall (Macro)	0.660593	0.579111	0.45049	0.451775
F1 (Macro)	0.682159	0.568495	0.469033	0.466525

Observations

Yet again, the TF-IDF vectorization with Hashtags included outperformed everything else on the SVM model. We're not done just ye however.

Neural Networks

This time, instead of building a baseline model, I will simply explore the accuracy of the model that was best for Sentiment Analysis see how it performs with a different target variable.

In [125]:
```
prod_hash.regularize()
prod_no_hash.regularize()
prod_sp_1.regularize()
prod_sp_2.regularize()
```

In [126]:
```
prod_sp_nn = [prod_sp_1, prod_sp_2]
prod_hash_nn = [prod_hash, prod_no_hash]
```

In [127]:
```
model_tfprod_1 = Sequential()

model_tfprod_1.add(Dense(100, activation='relu', input_shape=(400,)))
model_tfprod_1.add(Dropout(.3))
model_tfprod_1.add(Dense(75, activation='relu'))
model_tfprod_1.add(Dropout(.3))
model_tfprod_1.add(Dense(9, activation='softmax'))

model_tfprod_1.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

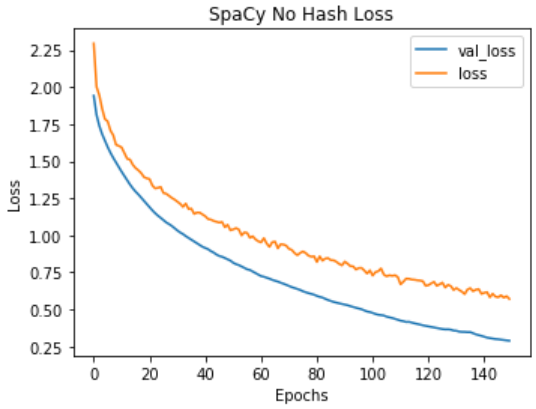
In [128]:
```
model_spprod_1 = Sequential()

model_spprod_1.add(Dense(100, activation='relu', input_shape=(96,)))
model_spprod_1.add(Dropout(.3))
model_spprod_1.add(Dense(75, activation='relu'))
model_spprod_1.add(Dropout(.3))
model_spprod_1.add(Dense(9, activation='softmax'))

model_spprod_1.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

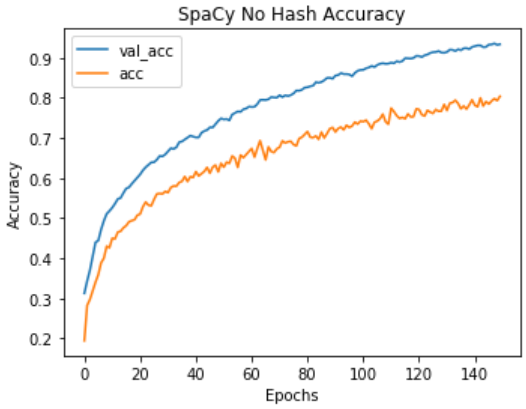
In [129]:
```
model_tfprod_2 = Sequential()

model_tfprod_2.add(Dense(100, activation='relu', input_shape=(300,)))
model_tfprod_2.add(Dropout(.3))
model_tfprod_2.add(Dense(75, activation='relu'))
model_tfprod_2.add(Dropout(.3))
model_tfprod_2.add(Dense(9, activation='softmax'))

model_tfprod_2.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['acc','mse']
)
```

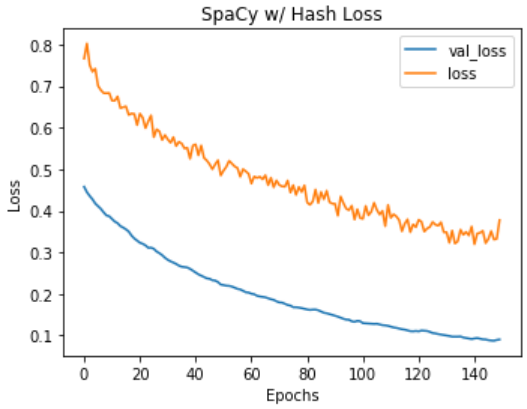
In [130]:
```
nn_sp_prod = ModelComparison(pipeline=model_spprod_1, data_list=prod_sp_nn, y_format='ohe', na
nn_sp_prod.fit_models(batch_size=256, epochs=150)

```

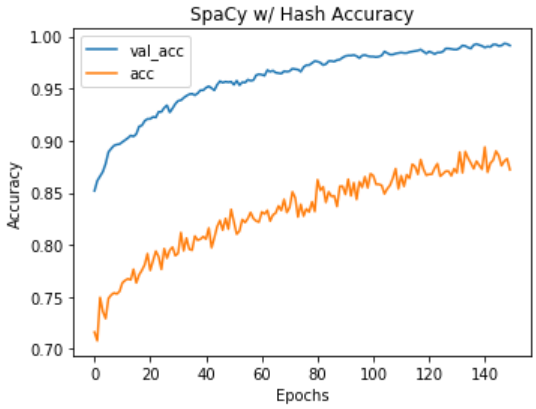
...

In [159]:
```
nn_sp_prod.plot_val_history()
nn_sp_prod.score_comparison
```









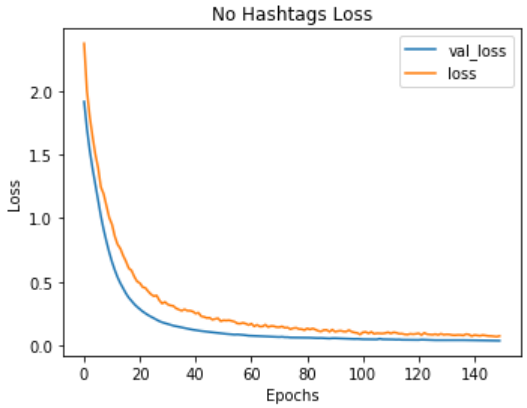
Out[159]:

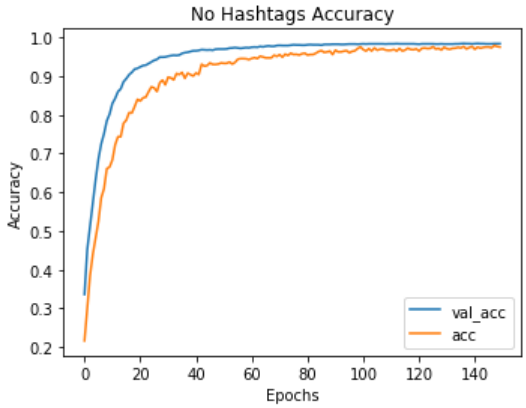
	SpaCy No Hash Holdout	SpaCy w/ Hash Holdout
Loss	1.432201	1.723528
Accuracy	0.581818	0.596970

In [131]:
```
nn_tf_prod = ModelComparison(pipeline=model_tfprod_1, data_list=[prod_hash], y_format='ohe', n
nn_tf_prod.fit_models(batch_size=256, epochs=150)

```

...

In [160]:
```
nn_tf_prod.plot_val_history()
nn_tf_prod.score_comparison
```





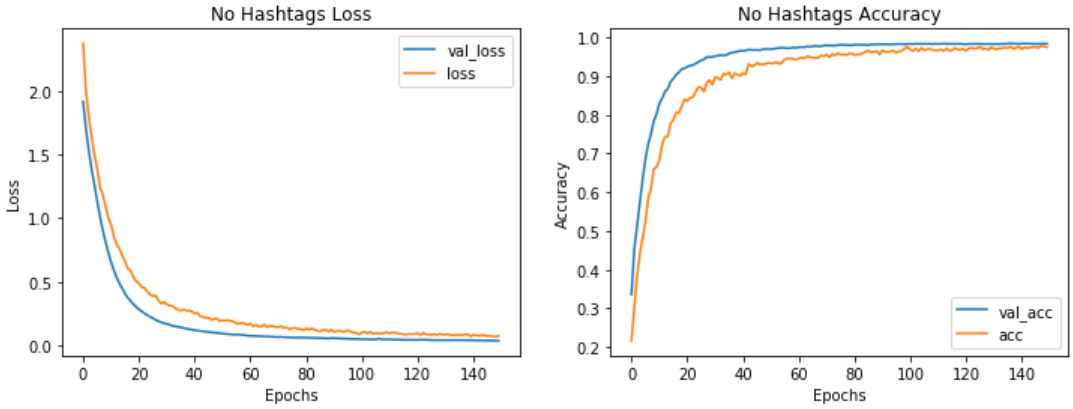
Out[160]:

	No Hashtags Holdout
Loss	1.290548
Accuracy	0.760606

In [132]:
```
nn_tf_prod = ModelComparison(pipeline=model_tfprod_2, data_list=[prod_no_hash], y_format='ohe'
nn_tf_prod.fit_models(batch_size=256, epochs=150)
nn_tf_prod.plot_val_history()
nn_tf_prod.score_comparison
```

...

In [161]:
```
nn_tf_prod.plot_val_history()
nn_tf_prod.score_comparison
```



Out[161]:

	No Hashtags Holdout
Loss	1.290548
Accuracy	0.760606

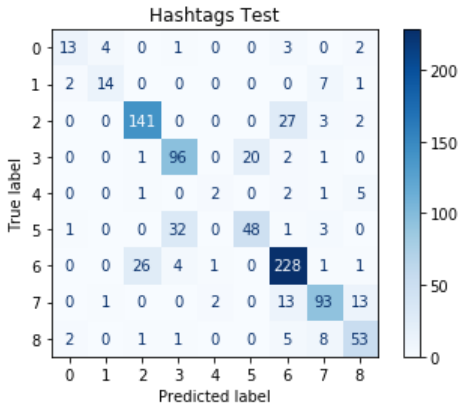
Observations

SVM still the most accurate model, beating naive baseline 79.5% to 29% and beating string detection methods by 5%. Once again, model is suprisingly underfit, meaning that our larger dataset is far noisier than our validation sets. The neural networks did fairly we ultimately, the Support Vector Machine was the most accurate model.

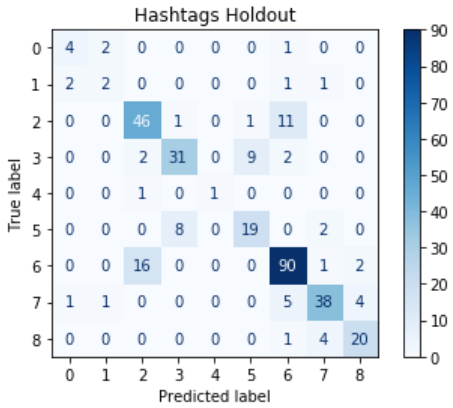
In [153]:
```
svm_products.fit_models([prod_hash])
```

```
--- 26.7781240940094 seconds to process ---
```

In [134]:
```
svm_products.compare_confusion(data=[prod_hash])
```



In [135]:
```
svm_products.compare_confusion(data=[prod_hash], data_type='holdout')
```



In [164]:
```
svm_products.calc_scores(data=[prod_hash], data_type='holdout')
svm_products.score_comparison
```

Out[164]:

	Hashtags Holdout
Accuracy	0.760606
Precision (Macro)	0.723936
Recall (Macro)	0.660593
F1 (Macro)	0.682159

Cleaning Up Those Nulls

Now that we have a fit model, we can use it to identify the products mentioned in the unlabeled tweets. Because of the TextSet obje
configuration, we can use the same label encoder to change the prediction outputs back into the string categories of the original da
From there we can see the frquency distributions of the predicted products against the distribution of the training data.

In [147]:
```
unknown_df.dropna(subset=['tweet_text'], inplace=True)
```

In [148]:
```
unknown_df['tweet_tokens'] = unknown_df['tweet_text'].map(lambda x: process_text(x, keep_hasht
```

In [154]:
```
unknown_df['model_vectorizor'] = unknown_df['tweet_tokens'].map(lambda x: prod_hash.vectorizor
unknown_df['model_prediction'] = unknown_df['model_vectorizor'].map(lambda x: svm_products.pip
unknown_df['emotion_in_tweet_is_directed_at'] = unknown_df['model_prediction'].map(lambda x: p
```

In [155]:
```
1 unknown_df
```

Out[155]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_produ
5	@teachntech00 New iPad Apps For #SpeechTherapy...	iPad or iPhone App	No emotion toward brand or produ
16	Holler Gram for iPad on the iTunes App Store -...	Other Apple product or service	No emotion toward brand or produ
32	Attn: All #SXSW frineds, @mention Register fo...	Android	No emotion toward brand or produ
33	Anyone at #sxsw want to sell their old iPad?	iPad	No emotion toward brand or produ
34	Anyone at #SXSW who bought the new iPad want ...	iPad	No emotion toward brand or produ
...	...	...	
9087	@mention Yup, but I don't have a third app yet...	Android App	No emotion toward brand or produ
9089	Wave, buzz... RT @mention We interrupt your re...	Other Google product or service	No emotion toward brand or produ
9090	Google's Zeiger, a physician never reported po...	Google	No emotion toward brand or produ
9091	Some Verizon iPhone customers complained their...	iPad	No emotion toward brand or produ
9092	¢Ï¡ ÏàŠü_‹ Ê‹ Î‹ Ò‹ £‹ Á‹ââ‹ _‹ £‹ ‹â_‹ÛâRT @...	Google	No emotion toward brand or produ

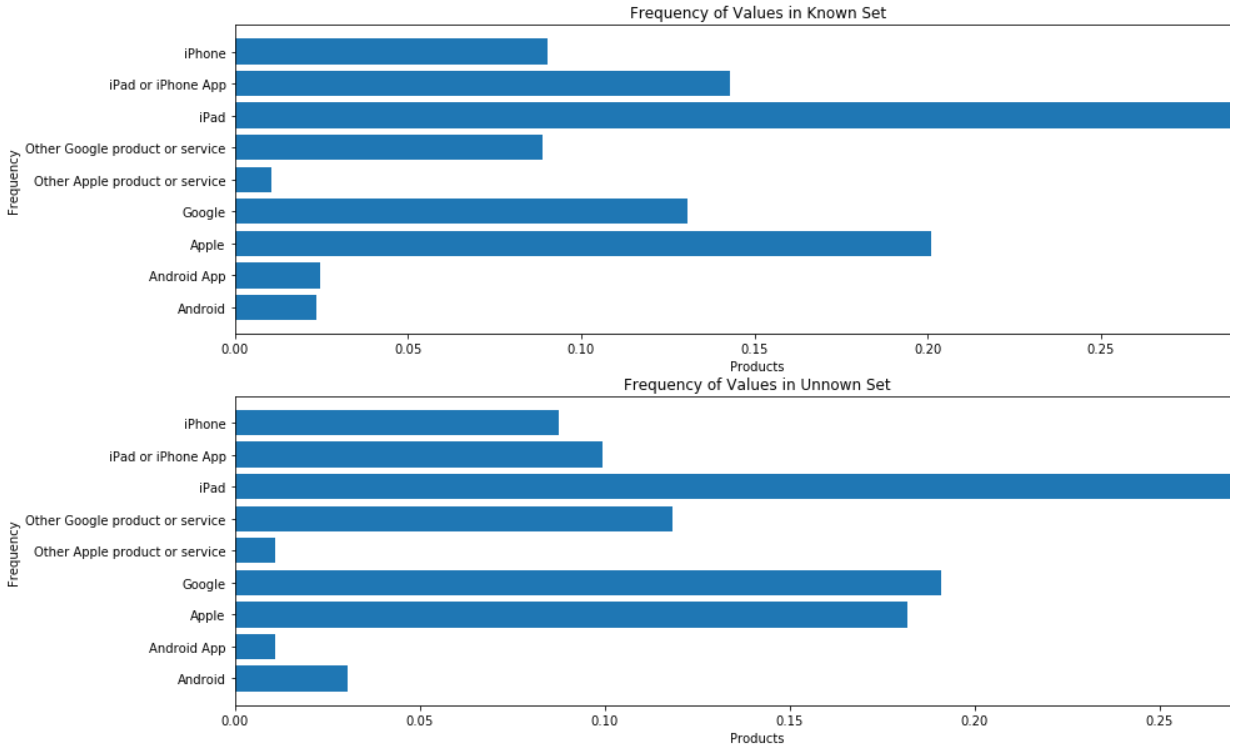
5801 rows × 6 columns

In [156]:
```
1 unknown_df['emotion_in_tweet_is_directed_at'].value_counts(normalize=True)
```

Out[156]:
```
iPad                               0.269264
Google                             0.190829
Apple                              0.181865
Other Google product or service    0.118255
iPad or iPhone App                 0.099466
iPhone                             0.087571
Android                            0.030684
Other Apple product or service     0.011033
Android App                        0.011033
Name: emotion_in_tweet_is_directed_at, dtype: float64
```

In [157]:
```
1 known_values = known_df['emotion_in_tweet_is_directed_at'].value_counts(normalize=True).to_dic
2 unknown_values = unknown_df['emotion_in_tweet_is_directed_at'].value_counts(normalize=True).to
```

In [158]:
```
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(15, 10))
x_1 = [x[0] for x in sorted(known_values.items())]
y_1 = [y[1] for y in sorted(known_values.items())]

x_2 = [x[0] for x in sorted(unknown_values.items())]
y_2 = [y[1] for y in sorted(unknown_values.items())]

ax1.barh(y=x_1, width=y_1)
ax1.set_title('Frequency of Values in Known Set')
ax1.set_xlabel('Products')
ax1.set_ylabel('Frequency')

ax2.barh(y=x_2, width=y_2)
ax2.set_title('Frequency of Values in Unnown Set')
ax2.set_xlabel('Products')
ax2.set_ylabel('Frequency')

plt.show()
```



While the distributions aren't identical, they are similar enough to build some basic assumptions that can be used to solve multiple business challenges.

Conclusions

There are a number of conclusions that can be drawn from this process, but more than anything it opens up questions about next s
Using tweets to measure Word of Mouth is fraught with challenges, but could be endlessly helpful. Twitter, as a platform, is meant to
short, quick responses to single items, meaning there is a lot to be gleaned from the Twitter population. The only medium that might
better at gauging fanbase sentiment would be Reddit.

Vectorization Methods

It's clear why TF-IDF succeeded over SpaCy's arguably 'smarter' vectorization methods with this dataset. SpaCy is designed to rea
natural language in sentences, with correct parts of speech and punctuation. Unfortunately, Twitter is as kind to proper grammar rul
dynamite is to fine china. For tasks such as identifying entities, while SpaCy has an inherent Named Entity Recognizer built into its p
that NER has to be trained on enough data to be effective. TF-IDF can identify those objects just by sheer token frequency, which m
sense. An iPad tweet will contain the word iPad and probably won't contain the word Samsung.

As for sentiment analysis, the differences weren't as massive. Sentiment is incredibly nuanced and, more often than not, subjective.
order to use natural language as training data, it must be tagged by a human and humans have all kinds of invisible biases.

Lastly, if we were to build a similar model for say Reddit or Facebook posts, there's a good chance that SpaCy would fare far better
becasuse both of those platforms encourage long statements and are more conducive to self editing before posting. The immediate
of Twitter makes it more noisy.

Hashtags

This is specific to Twitter and any other Social Media platform that uses hashtags. In the end, the data that retained hashtags fared
on both tasks. This will be useful information for building any data pipelines in the future.

Business Applications

There is no end to the usefulness of sentiment analysis. If this were to be transformed into a business-ready data pipeline (which I w
discuss in a moment) it could be used to measure the aggregated effect of marketing campaigns on word of mouth, which, hopefull
would be a lead measure to an increase in sales. One could measure a baseline of interest in a brand before and after a product rele
perhaps, to see how much the general public is engaging with your product. Coupled with Sentiment Analysis, a marketing team co
build KPIs to understand their reach and how effective the product is, or if they need to create a new campaign to shift attention aw
negative press.

Alertnatively, these processing methods could be used to identify upticks in complaints about certain products or aspects of produc
could be instrumental to community engagement and feedback. I know from my own experience that video game companies rely h
on the feedback posted on Twitter and Reddit to make changes to their games over time. A datapipeline like this could help reduce
workload on that department, increasing efficiency and help community managers respond to challenges and bugs far faster.

Next Steps

There is a lot that can be done to improve this process, as with any NLP analysis. A language detector could be useful to reduce the
tweets down to English only. Also, for the product recognition model, we have no 'No Product' tweets. In order to make this pipeline
business ready, it would need to be trained on random tweets that do not mention any of the products we are identifying.

Lastly, the datasets are still small, so in order to improve the model's effectiveness, it will require far more labeled data, which can b
costly in terms of implementation. That said, once the data is labeled, the TextSet and ModelComparison classes make model analy
testing extremely efficient and can be applied to future pipelines using any manner of text-based medium.

Overall, with 70% accuracy on sentiment and 80% accuracy on product recognition, it's clear that these methods are successful or
the messiest of tweets. There are many directions these tools can go and it will be exciting to see how the field of Natural Language
Processing evolves over the coming years.