In [1]:
```
# Imports. Nothing to see here.

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import string

from imblearn.over_sampling import SMOTE
from gensim.models import Word2Vec
from collections import Counter

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import FreqDist, word_tokenize
from nltk.tokenize import RegexpTokenizer

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, plot_confusion_
matrix, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
```

EDA

Here we check out the data. We explore, look for inconsistencies, and use what we see to define our modeling plan. We start with general data cleaning practice, checking for duplicates and missing values. As we get deeper into the data, our techniques get more specific to the data.

In [2]:
```
# Import and inspect data

data = pd.read_csv('data.csv', encoding = "ISO-8859-1")

data.head()
```

Out[2]:

	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

In [3]:
```
# Read content. These are a dataset of Tweets from SXSW in Austin from 2011.

data.iloc[0,0]
```

Out[3]:
```
'.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead!  I need to upgrade. Plugin stations at #
SXSW.'
```

In [4]:
```
# Check for missing values in the primary column and remove the one we find.

print(data['tweet_text'].isna().sum())

data = data[~data['tweet_text'].isna()]
```

```
1
```

In [5]:
```
# Check for duplicated rows and preserve unique entries.

a = len(data)
data = data.drop_duplicates()
b = len(data)
print('# Number of duplicate rows dropped: {}'.format(a-b))
```

```
# Number of duplicate rows dropped: 22
```

In [6]:
```
# Explore and simplify. We're defining our project goals more closely here.

print(data['emotion_in_tweet_is_directed_at'].value_counts())

company = {'iPad': 'Apple',
           'Apple': 'Apple',
           'iPad or iPhone App': 'Apple',
           'Google': 'Google',
           'iPhone': 'Apple',
           'Other Google product or service': 'Google',
           'Android App': 'Google',
           'Android': 'Google',
           'Other Apple product or service': 'Apple'}

data['emotion_in_tweet_is_directed_at'] = data['emotion_in_tweet_is_directed_at'].map(company)
```

```
iPad                               945
```

```
iPad                               945
Apple                              659
iPad or iPhone App                 469
Google                             428
iPhone                             296
Other Google product or service    293
Android App                         80
Android                             77
Other Apple product or service      35
Name: emotion_in_tweet_is_directed_at, dtype: int64
```

In [7]:
```
# Simplify column names for convenience.

data.rename(columns={'tweet_text': 'text', 'emotion_in_tweet_is_directed_at': 'brand', 'is_there_an_emotion_directed_at_a_brand_or_product': 'feelings'}, inplace=True)
```

In [8]:
```
# Here we inspect and encode the labels for our target column. We also define the boundaries of our task, narrowing our dataset to entries with clear positive or negative expressions.

print('Original dataset values:\n', data['feelings'].value_counts(), '\n')

feels = {'Negative emotion': 0,
        'Positive emotion': 1,
        'No emotion toward brand or product': 2,
        "I can't tell": 3}

data['feelings'] = data['feelings'].map(feels)

data = data[data['feelings'] <= 1]

print('Encoded and chosen dataset values:\n', data['feelings'].value_counts(), '\n')

print('Total entries:', len(data))
```

```
Original dataset values:
 No emotion toward brand or product    5375
Positive emotion                      2970
Negative emotion                       569
I can't tell                           156
Name: feelings, dtype: int64

Encoded and chosen dataset values:
 1    2970
0     569
Name: feelings, dtype: int64

Total entries: 3539
```

Now the NLP Begins

Our data is clean and ready to be processed. Now we process. We start by creating tools, then use the tools on the Tweet text to reshape the data into a manageable and meaningful form.

In [9]:
```
# Create tools to process the Tweets. We use Regex to narrow our data to words, and stopwords to return only the words that are significant.

tokenizer = RegexpTokenizer(r'\w+')

stops = stopwords.words('english')
stops += list(string.punctuation)
stops.extend(['sxsw', 'sxswi', 'quot', 'mention', 'link', 'rt', 'amp', 'http', 'sxswrt', 'google', 'googles', 'app', 'apps', 'android', 'austin', 'quotgoogle', 'new', 'today', 'one', 'apple', 'ipad', 'iphone', 'ipad2', 'apples', 'quotapple','store'])
```

In [10]:
```
# Use our new tools!

data['tokens'] = data['text'].apply(tokenizer.tokenize)

# This is inefficient and can be worked to be better. First pass eliminates stopwords that are already lowercase, and turns the rest of the words lowercase. Second pass removes the remaining stopwords. It also leaves our data in list form, which can be an issue later. Our lemmatizing function is built to work with a list. We fix the issue before we Count Vectorize or TD-IDF in the main function below.
data['tokens'] = data['tokens'].apply(lambda x: [word.lower() for word in x if word not in stops])
data['tokens'] = data['tokens'].apply(lambda x: [word.lower() for word in x if word not in stops])

data
```

Out[10]:

	text	brand	feelings	tokens
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	Apple	0	[wesley83, 3g, 3, hrs, tweeting, rise_austin, ...
1	@jessedee Know about @fludapp ? Awesome iPad/i...	Apple	1	[jessedee, know, fludapp, awesome, likely, app...
2	@swonderlin Can not wait for #iPad 2 also. The...	Apple	1	[swonderlin, wait, 2, also, sale]
3	@sxsw I hope this year's festival isn't as cra...	Apple	0	[hope, year, festival, crashy, year]
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	1	[sxtxstate, great, stuff, fri, marissa, mayer,...
...	...	...	...	...
9077	@mention your PR guy just convinced me to swit...	Apple	1	[pr, guy, convinced, switch, back, great, cove...
9079	"papyrus...sort of like the ipad" - ...	Apple	1	[papyrus, sort, like, nice, lol, lavelle]

9080	Diller says Google TV "might be run over ...	Google	0	[diller, says, tv, might, run, playstation, xb...
9085	I've always used Camera+ for my iPhone b/c it ...	Apple	1	[always, used, camera, b, c, image, stabilizer...
9088	Ipad everywhere. #SXSW {link}	Apple	1	[everywhere]

3539 rows × 4 columns

Creating Functions for User Interface

It is important that we create functions for each of the modeling steps. We will be selecting which ones to use individually later and they all need to be defined before we know which ones will be used. It is extra important that they all work well individually and also do not interfere with each other.

In [11]:
```
# Count Vectorize or TF-IDF. Our first and most vital choice.
def CV(X_train, X_test):
    count_vectorizer = CountVectorizer()
    X_train_counts = count_vectorizer.fit_transform(X_train)
    X_test_counts = count_vectorizer.transform(X_test)
    return X_train_counts, X_test_counts

def tf_idf(X_train, X_test):
    tfidf = TfidfVectorizer()
    X_train_counts = tfidf.fit_transform(X_train)
    X_test_counts = tfidf.transform(X_test)
    return X_test_counts, X_train_counts
```

In [12]:
```
# Lemmatizing will or won't happen. Two functions, one nested inside the other.
def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(w) for w in text]

def lemmatize():
    data['lemm'] = data['tokens'].apply(lemmatize_text)
    data['lemm'] = data['lemm'].apply(lambda x: ' '.join(x))
```

In [13]:
```
# SMOTE either will or will not run.
def smote(X_train_counts, y_train):
    smote = SMOTE()
    X_train_counts, y_train = smote.fit_sample(X_train_counts, y_train)
    return X_train_counts, y_train

# Train Test Split. The col variable is important and will be different depending on whether we lemmatize.
def TTS(col):
    X_train, X_test, y_train, y_test = train_test_split(data[col], data['feelings'])
    return X_train, X_test, y_train, y_test
```

In [14]:
```
# Logistic Regression
def logreg(X_train_counts, y_train, X_test_counts):
    clf = LogisticRegression(C=30.0, class_weight='balanced', solver='newton-cg',
                             multi_class='multinomial', n_jobs=-1, random_state=40)
    clf.fit(X_train_counts, y_train)
    y_predicted_counts = clf.predict(X_test_counts)
    return y_predicted_counts
```

In [15]:
```
# Random Forest
def rf(X_train_counts, y_train, X_test_counts):
    rf = RandomForestClassifier()
    rf.fit(X_train_counts, y_train)
    y_predicted_counts = rf.predict(X_test_counts)
    return y_predicted_counts
```

In [16]:
```
# Multinomial Naive Bayes
def multiNB(X_train_counts, y_train, X_test_counts):
    nb = MultinomialNB()
    nb.fit(X_train_counts, y_train)
    y_predicted_counts = nb.predict(X_test_counts)
    return y_predicted_counts
```

In [17]:
```
# Metrics that will work with any model selected, along with a confusion matrix.
def classify(y_test, y_predicted_counts):
    print('\n\nClassification Report - TEST')
    print('-------------------------------------------------------------------------')
    print(classification_report(y_test, y_predicted_counts))
    print('-------------------------------------------------------------------------')
    print('Confusion Matrix - TEST')
    print('-------------------------------------------------------------------------')
    print(pd.crosstab(y_test, y_predicted_counts, rownames=['True'], colnames=['Predicted'], margins=True))
    print('-------------------------------------------------------------------------')
```

In [ ]:
```
```

Our Modeling System!

We made the choice to create an interactive modeling system. The user will be asked to supply inputs and choose his own path to

We made the choice to create an interactive modeling system. The user will be asked to supply inputs and choose his own path to model the data. First, we choose to Count Vectorize or TF-IDF, whether or not to Lemmatize or to SMOTE. There are 8 possible combinations of choices, each accounted for. We've even processed every combination beforehand to have recommended modeling depending on the user's choices. Modeling is fun again!

In [18]:
```
def user_models(data):

    # Prepare some variables.
    col = None
    model = None
    lemm = None
    smt = None
    reco = None
    picks = []
    choices = [logreg, rf, multiNB]
    t = None

    # User choices will influence recommended model selection.
    log = [[1, 2, 2], [2,2,2]]
    forest = [[1,1,2],[2,1,2]]
    NB = [[1,2,1],[2,2,1],[1,1,1,],[2,1,1]]

    # Canned response for every response that isn't '1' or '2.'
    jerk = "Don't waste my time. Try again wiseguy."

    # Here we ask three binary questions, leading to 8 possible combinations. The answers affect what functions are run, an
d also are saved in a list and used to compare for model selection.
    print('How would you like to analyze the data?\nType "1" to Count Vectorize or "2" to implement TF-IDF.\n')
    model = input()
    picks.append(int(model))
    if model not in ['1', '2']:
        return print(jerk)

    print('Would you like to lemmatize?\nType "1" for Yes or "2" for No.\n')
    lemm  = input()
    picks.append(int(lemm))
    if lemm not in ['1', '2']:
        return print(jerk)

    print('Would you like to SMOTE?\nType "1" for Yes or "2" for No.\n')
    smt = input()
    picks.append(int(smt))
    if smt not in ['1','2']:
        return print(jerk)

    # Lemmatize function runs or not. The function requires the data to be in list form. The function will remove the list
 form.
    # If not lemmatized, data['tokens'] column is taken out of list form.
    if lemm == '1':
        col = 'lemm'
        lemmatize()
        print('Lemmatized changed this many rows:', (len(data) - (sum(data['tokens'] == data['lemm']))),'\n')
    elif lemm == '2':
        col = 'tokens'
        data['tokens'] = data['tokens'].apply(lambda x: ' '.join(x))

    # Here we compare the user's choices to our previously established lists. They are organized and hand picked from perso
nal results to produce the best results.
    if picks in log:
        t = 0
    elif picks in forest:
        t = 1
    elif picks in NB:
        t = 2

    # Train Test Split. Our column choice is based on lemmatization choice.
    X_train, X_test, y_train, y_test = TTS(col)

    # Count Vectorize or TF-IDF.
    if model == '1':
        X_train_counts, X_test_counts = CV(X_train, X_test)
    elif model == '2':
        X_test_counts, X_train_counts = tf_idf(X_train, X_test)

    # SMOTE or not.
    if smt == '1':
        X_train_counts, y_train = smote(X_train_counts, y_train)

    # Now that choices are made and we've picked our recommendation, we give the user a choice. They have control of the en
tire modeling process.
    print('Would you like to pick your model or use our recomendation?\nType "1" to choose or "2" to let us.')
    reco = input()
    if reco not in ['1','2']:
        return print(jerk)
    elif reco == '1':
        print('\n\n"1" for Logistic Regression\n"2" for Random Forest\n"3" for Multinomial Naive Bayes')
        t = (int(input()) - 1)

    # t variable is either recommended by us or chosen by user. It picks an option from a list of models.
    mod = choices[t]
    if mod == logreg:
        print('\n\nLogistic Regression Report')
    elif mod == rf:
        print('\n\nRandom Forest Report')
    elif mod == multiNB:
        print('\n\nMultinomial Naive Bayes')
```

```
    # We run the chosen model and print out metrics for evaluation, along with a confusion matrix.
    y_predicted_counts = mod(X_train_counts, y_train, X_test_counts)
    classify(y_test, y_predicted_counts)
```

In [19]:
```
user_models(data)
```

```
How would you like to analyze the data?
Type "1" to Count Vectorize or "2" to implement TF-IDF.

Would you like to lemmatize?
Type "1" for Yes or "2" for No.

Would you like to SMOTE?
Type "1" for Yes or "2" for No.

Lemmatized changed this many rows: 3539

Would you like to pick your model or use our recomendation?
Type "1" to choose or "2" to let us.


"1" for Logistic Regression
"2" for Random Forest
"3" for Multinomial Naive Bayes


Logistic Regression Report


Classification Report - TEST
--------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.45      0.59      0.51       148
           1       0.91      0.86      0.88       737

    accuracy                           0.81       885
   macro avg       0.68      0.72      0.70       885
weighted avg       0.83      0.81      0.82       885

--------------------------------------------------------------------------
Confusion Matrix - TEST
--------------------------------------------------------------------------
Predicted    0    1  All
True
0           87   61  148
1          106  631  737
All        193  692  885
--------------------------------------------------------------------------
```

In [20]:
```
data
```

Out[20]:

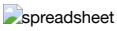
	text	brand	feelings	tokens	lemm
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	Apple	0	[wesley83, 3g, 3, hrs, tweeting, rise_austin, ...	wesley83 3g 3 hr tweeting rise_austin dead nee...
1	@jessedee Know about @fludapp ? Awesome iPad/i...	Apple	1	[jessedee, know, fludapp, awesome, likely, app...	jessedee know fludapp awesome likely appreciat...
2	@swonderlin Can not wait for #iPad 2 also. The...	Apple	1	[swonderlin, wait, 2, also, sale]	swonderlin wait 2 also sale
3	@sxsw I hope this year's festival isn't as cra...	Apple	0	[hope, year, festival, crashy, year]	hope year festival crashy year
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	1	[sxtxstate, great, stuff, fri, marissa, mayer,...	sxtxstate great stuff fri marissa mayer tim re...
...	...	...	...	...	...
9077	@mention your PR guy just convinced me to swit...	Apple	1	[pr, guy, convinced, switch, back, great, cove...	pr guy convinced switch back great coverage pr...
9079	"papyrus...sort of like the ipad" - ...	Apple	1	[papyrus, sort, like, nice, lol, lavelle]	papyrus sort like nice lol lavelle
9080	Diller says Google TV "might be run over ...	Google	0	[diller, says, tv, might, run, playstation, xb...	diller say tv might run playstation xbox essen...
9085	I've always used Camera+ for my iPhone b/c it ...	Apple	1	[always, used, camera, b, c, image, stabilizer...	always used camera b c image stabilizer mode s...
9088	Ipad everywhere. #SXSW {link}	Apple	1	[everywhere]	everywhere

3539 rows × 5 columns

Our Favorite Models

We took an extremely scientific approach to analyzing our models: Predict, Experiment, Observe, Record the Results. With 3 possible models, a choice of processing the data, a choice to lemmatize and a choice to SMOTE, we end up with 24 (3 x 2 x 2 x 2) unique models. Observing the dataset to be imbalanced in favor of positive sentiments, we decided to focus primarily on the recall of the minority class. In practical terms, we are attempting to identify as many unhappy users of Google and Apple products as we can and prefer to err on the side of classifying extra unhappy users as oppoosed to missing them.

We run all 24 models individually and record their most important metrics: Accuracy, Precision, Recall, and F1 Score. Looking at the spreadsheet of results, a few things jump out at us. Excepting several outliers of lower recall, most of our recall scores are in the 0.40 - 0.60 range. Thus, we decided to factor in an overall metric, Accuracy when making our decisions. In each combination we use these metrics to choose our best model. In many categories it was a close decision and could have easily been justified to go in another direction.



Our favorite model overall used Term Frequency–Inverse Document Frequency to transform our data, Lemmatization, Smoting, and used Multinomial Naive Bayes to model it. We used all the bells and whistles for our best result. It produced our third highest recall score, and a higher overall accuracy than the two models with higher recall. Again, it was a tough call and other model choices could have been justified just as well.