Pokemon Go sentiment during and after the Pokemon GO Fest

Holly Johnsen

Abstract

On July 22, 2017, Niantic held its Pokemon GO Fest in Chicago to celebrate the 1-year anniversary of its game, Pokemon Go. Unfortunately, the event was plagued by long lines, server overload, and glitchy user experience and was panned on social media. Inspired by this event, I queried and downloaded tweets related to Pokemon Go and the Pokemon GO Fest over the course of a month to see if there were changes in sentiment towards the game and the event on Twitter. I used two versions of a NLTK Naive Bayes classifier and found that sentiment stayed mostly the same over time.

Motivation

I wanted to see whether the failure of the Pokemon GO Fest would have a lasting effect on sentiment towards Pokemon Go. This analysis would be useful to Niantic to determine what steps, if any, would be necessary to recover its reputation with its users. It would also be useful to other companies considering hosting similar events so that they could determine the potential cost of a poorly-planned event.

Dataset(s)

I built my own dataset by querying Twitter most days for about a month (July 22 to August 29, minus a week). I built two data sets in parallel. One queried using one of three different terms: “#pokemongofest, pokemongo, and Pokemon Go” (53 queries) The other searched for any of these terms, excluding retweets and non-English tweets, originating in Chicago or with location unspecified (64 queries). Each query retrieved up to 100 tweets per search term or location, for a total of about 30,000 tweets.

Sentiment classifiers were trained using the built in NLTK Twitter corpus and a second corpus from thinknook.com (Twitter Sentiment Analysis Dataset)

Data Preparation and Cleaning

Before analyzing sentiment, the status text had to be extracted from the Twitter queries and tokenized. In preparation for my first classifier, I used NLTK's built-in token classifier. For the second classifier, I used a different classifier in provided by NLTK that is particularly suited for Twitter.

Unfortunately, I had already been collecting tweets for a while before beginning to analyze the sentiment. Looking back, I should have done all search queries in a way to exclude non-English tweets. I also should have done a version of the query that excluded retweets. For example, one day was dominated by numerous retweets of the same item. Although this can provide interesting information, collecting non-retweet data in parallel would maximize the data I could use, since filtering out repeat tweets post-query results in a smaller set of data.

Research Question(s)

The question I aimed to answer was “How will user sentiment on Twitter regarding Pokemon Go and the Pokemon GO Fest event change over time following the event?” I looked at two different sets of data. One compared sentiment for three different search queries (two about Pokemon Go broadly and one specific to the event), and the other compared sentiment in Chicago (where the event occurred) to sentiment in the world at large.

I also became interested in investigating how different choices in data analysis affected these results.

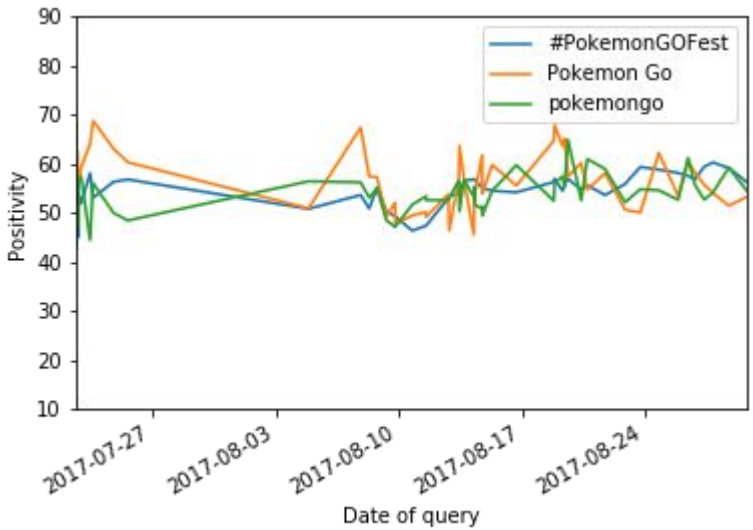
Methods

I trained two NLTK Naive Bayes classifiers for positive and negative sentiments. Although the classifiers were of the same type, they were trained on two separate datasets. The first was trained on the built-in NLTK twitter_samples corpus, which includes 10,000 tweets (equally split between positive and negative). These were already available in a tokenized form. The second classifier was trained on a different Twitter Sentiment Analysis Dataset, which includes about 1.5 million tweets (approximately equally split between positive and negative). These were not pre-tokenized, so I used a “Twitter-aware” tokenizer (nltk.tokenize.casual.TweetTokenizer). The tokenized corpera were turned into bag-of-words for training.

I tokenized my own tweets, created bag-of-words, and used the classifiers to predict the probability of each tweet being positive. For each query on each day, I calculated an “approval rating” as the average of these probabilities. Using the probability of being positive rather than actually using the classifier to make a binary prediction reduced variance, which was important given my relatively small dataset. I plotted the approval ratings over time.

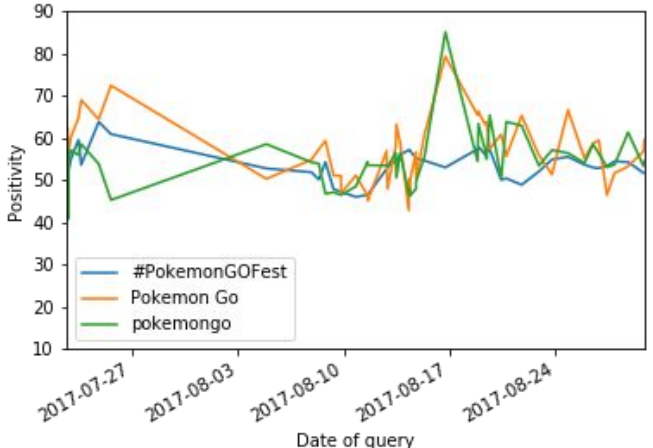
Findings

- Sentiment was generally the same over the time period
  - Although I expected the sentiment about the event to be strikingly negative at and immediately following the event, it was not. I did not have the data to establish the baseline sentiment prior to the event, but it did not appear to change in a dramatic way over the next month.
  - When I investigated the tweets during the early time points in more detail, I found that there were many non-English tweets that I had not filtered out, which were receiving positive sentiment scores. Also, legendary pokemon had been announced, which created some positive tweets among those decrying the event. These might have buffered the sentiment score.



Findings

- Including retweets can have a big effect
  - For one of my searches, I did not filter out retweets or duplicate tweets. I found that one day, August 16th, had a major peak in positivity related to Pokemon Go (or pokemongo) but not the GO Fest. Upon further investigation, I found that the tweets on that day were mostly identical retweets of a single tweet announcing the shiny versions of the Pikachu family. This tweet was rated positively by the sentiment analyzer and dominated that day's rating. This peak disappeared when duplicate tweets were filtered (see graph on previous slide).
  - Keeping retweets might decrease the diversity of tweets seen, but it can identify times with great interest in a trending topic, which would likely be of interest to a company.

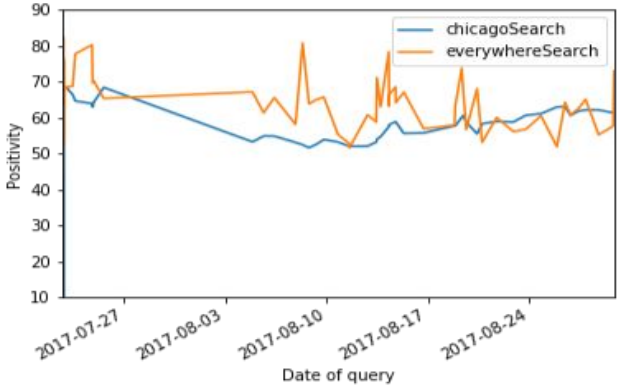


74% of hits for the "pokemongo" query on August 16th were identical: "RT @Pokemon: Spotted: Shiny Pikachu, Pichu, and Raichu in #PokemonGO! Be on the lookout for these Shiny versions as you explore: https://t.…"
Probability of being a positive tweet: 0.8540243989310298

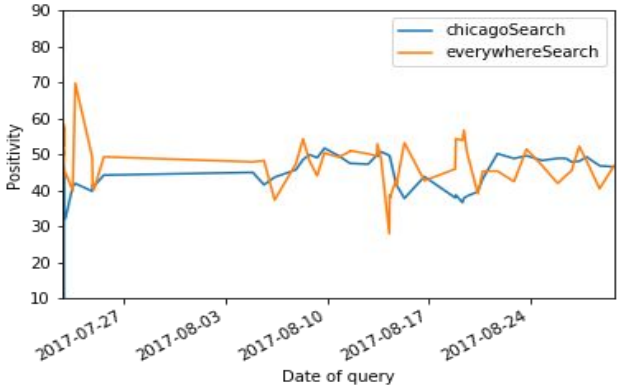
Findings

- Sentiment in Chicago might have been lower than elsewhere
  - The first classifier found that sentiment in searches within 100 miles of Chicago was lower than non-geocode-limited searches for most of the month (top graph)
  - However, this difference was not observed in analysis using the second classifier, so it might have been an artifact.
  - The two classifiers had very different average positivity scores.

First Classifier:

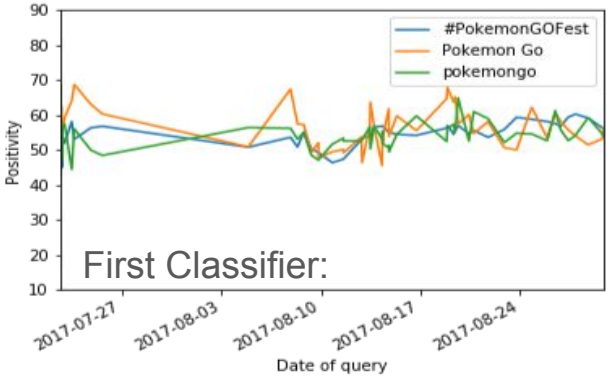


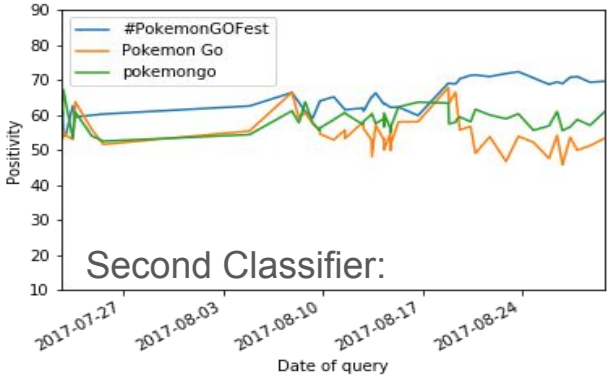
Second Classifier:

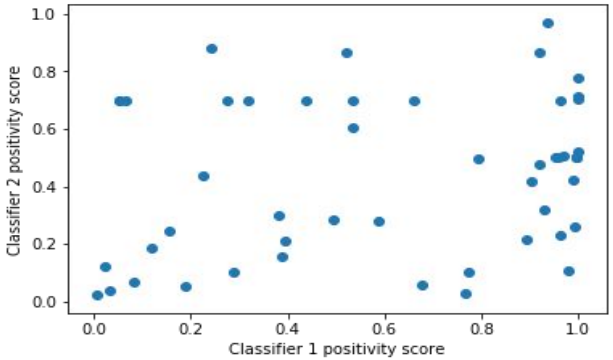


Findings

- Models trained on different training sets can give quite different results
  - The two classifiers I used were the same type of classifier, but they were trained on different Twitter corpera.
  - The two classifiers gave fairly different results. For example, with the first classifier, it seems that the three search terms had similar sentiment scores, but with the second classifier, it seemed that “#PokemonGOFest” (blue line) outscored the other two
  - A scatterplot of the scores for some tweets by the two classifiers shows that the first classifier tends to give high positivity scores, but the second classifier is more evenly distributed.
  - I classified some of the tweets as positive or negative myself to calculate the log-likelihood for each classifier to see which is better. They both had several hits and misses with fairly similar log-likelihoods (-7.96 vs -7.71).







Limitations

There were several limitations to my study.

First, it would be better if I had a baseline of sentiment related to Pokemon Go before the event to compare to. Second, I was only able to download 100 tweets per query. For some of my searches, I did not filter out foreign language tweets, for which the classifiers inexplicably predicted a non-neutral sentiment.

The built-in NLTK Twitter corpus seemed to be biased, as all tweets in it contained an emoticon. The other corpus I used was larger and likely more representative, but I often disagreed with its predictions. These datasets are possibly not well-matched to the tweets about Pokemon Go. Many of the tweets in my dataset did not seem to have much sentiment, but were rather simply informative, sharing updates without much emotion or linking to a video they watched. It does not necessarily make sense to predict sentiment for such tweets.

There are some outputs that I cannot explain, which would require further investigation. For example, I am unsure why the two classifiers have similar average scores during the three-search-query analysis, but very different average scores when comparing Chicago to. non-geocode-tagged locations. I also calculated the accuracy for my classifiers using a very small manually-annotated set of tweets. For this, I only used tweets that I felt confident about annotating as positive or negative, so they might also not have been representative of the dataset as a whole.

Conclusions

I conclude that based on the sentiment classifiers I used, overall sentiment towards Pokemon Go and the Pokemon GO Fest did not change much over the month following the GO Fest event. However, the sentiment predictions that the classifiers made did not always make sense. A more accurate way of analyzing sentiment might still reveal interesting changes or patterns in the data.

Acknowledgements

I collected the data myself, following the methods and code introduced in this MOOC. My husband assisted me with calculating the log-likelihood, debugging, and general feedback. I used Python and NLTK documentation and Stack Overflow to learn new techniques.

References

The second classifier I used was trained on the dataset available here:

http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/

In this notebook, I made code that could download and save tweets about Pokemon Go. I decided to do this after seeing the bad reactions on social media about the Pokemon Go Fest in Chicago on July 22nd, which was plagued by long lines, server overload, and glitchy user experience. I ran these scripts every day (except while on vacation) for about a month so that I could see if I could follow the sentiment of Pokemon Go users over time.

The steps in this project were to:

1. Search for tweets about pokemon go and save them for later analysis. I did two searches. One looked at a few different search terms individually, and the other looked for any of those search terms specifically in Chicago vs. everywhere.
2. Training a sentiment classifier (Naive Bayes Classifier). I used the built-in NLTK twitter corpus as a gold standard at first, but then didn't feel like it was doing a great job. In particular, the NLTK corpus was established using emoticons to determine the true sentiment. The result of that is that emoticons became absurdly informative features, but I think that emoticons are less used now than emoji, so I didn't feel like the corpus was well-matched to present-day tweets.
3. Perparing the saved tweets for analysis and applying the classifier to see trends over time.
4. Training a second sentiment classifier. I found a massive twitter corpus online which I used to train a second sentiment classifier.
5. Comparing the two classifiers
6. Looking at the tweets from the best and worst days

In [ ]:
```
import pickle
import os
import nltk
import twitter
from datetime import datetime
import pandas as pd
import string
import json
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
%cd "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP-Databases/pi
```

In [ ]:
```
pwd = !pwd
print(pwd)
if not pwd[0] == "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP
    %cd "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP-Database

if not os.path.exists('secret_twitter_credentials.pkl'):
    Twitter={}
    Twitter['Consumer Key'] = ''
    Twitter['Consumer Secret'] = ''
    Twitter['Access Token'] = ''
    Twitter['Access Token Secret'] = ''
    with open('secret_twitter_credentials.pkl','wb') as f:
        pickle.dump(Twitter, f)
else:
    Twitter=pickle.load(open('secret_twitter_credentials.pkl','rb'))

auth = twitter.oauth.OAuth(Twitter['Access Token'],
                           Twitter['Access Token Secret'],
                           Twitter['Consumer Key'],
                           Twitter['Consumer Secret'])

twitter_api = twitter.Twitter(auth=auth)

%cd pickles/
```

I did not end up using these trends for analysis.

In [ ]:
```
worldID = 1
usID = 23424977
chicagoID = 2379574
grantparkID = 12784255 #using zipcode 60601 for lookup

trends = {}
trends["world_trends"] = twitter_api.trends.place(_id=worldID)
trends["us_trends"] = twitter_api.trends.place(_id=usID)
trends["chicago_trends"] = twitter_api.trends.place(_id=chicagoID)
#not working
#trends["grantpark_trends"] = twitter_api.trends.place(_id=grantparkID)

#since trends might be time sensitive, I want to save them
currTime = str(datetime.now())
with open(currTime + "trends.pkl",'wb') as f:
    pickle.dump(trends, f)
```

In [ ]:
```
#alltrends = {}
#for trend in trends.keys():
#    trendlist = trends[trend][0]["trends"]
#    for item in trendlist:
#        #print(item["name"] + ", " + str(item["tweet_volume"]))
#        currentValue = alltrends.get(item["name"], 0)
#        if item["tweet_volume"] is None or item["tweet_volume"]==0:
#            pass
#        else: alltrends[item["name"]] = item["tweet_volume"] + currentValue
##for item in alltrends.keys():
##    print (item + ", " ,alltrends[item])
#df = pd.Series(alltrends)
##print(df)
#df.sort_values(ascending = False)
```

In [ ]:
```
#cTrends = {}
#for item in trends["chicago_trends"][0]["trends"]:
#    cTrends[item["name"]] = item["tweet_volume"]
#dfc = pd.Series(cTrends)
#dfc.sort_values(ascending = False)
```

Getting and saving tweets about PoGo

Here I searched for tweets about pokemon go or the specific event. I did not realize it, but the case doesn't matter in Twitter's search, so "#PokemonGOFest" and "#pokemongofest" were identical searches.

100 is the maximum number that could be found and saved with these searches.

In [ ]:
```
a = 'Pokemon Go'
b = "#PokemonGOFest"
c = "#pokemongofest"
d = "pokemongo"
```

In [ ]:
```
number = 100
searchresults = {}
searchresults["a"] = twitter_api.search.tweets(q=a, count=number)
searchresults["b"] = twitter_api.search.tweets(q=b, count=number)
searchresults["c"] = twitter_api.search.tweets(q=c, count=number)
searchresults["d"] = twitter_api.search.tweets(q=d, count=number)

currTime = str(datetime.now())
with open(currTime + "search.pkl",'wb') as f:
    pickle.dump(searchresults, f)
```

In [ ]:
```
for search in searchresults.keys():
    print(search)
    print(len(searchresults[search]["statuses"]))
```

Getting and saving tweets about PoGo in Chicago vs. elsewhere

In [ ]:
```
q = '-RT Pokemon Go OR #PokemonGOFest OR #pokemongofest OR pokemongo'
# This is centered on Grant Park, where the event took place.
loc = "41.8722,-87.621887,100mi"
lang = "en"
number = 100

search = {}
search["chicagoSearch"] = twitter_api.search.tweets(q=q, geocode=loc, lang=l
search["everywhereSearch"]  = twitter_api.search.tweets(q=q, lang=lang, cour

currTime = str(datetime.now())
with open(currTime + "Csearch.pkl",'wb') as f:
    pickle.dump(search, f)
```

In [ ]:
```
for i in search.keys():
    print(i)
    print(len(search[i]["statuses"]))
```

I am not sure why the non-geocode-limited search result returns fewer queries.

In [ ]:
```
#for a given "search" (stored as a dictionary),
#extract the a list of status texts for a location (a key in that dict)
def getStatuses(d,k):
    return [s['text'] for s in d[k]['statuses']]
```

In [ ]:
```
#filter, from "Using the Twitter API for Tweet Analysis"
# modified since statuses here is a list of just the text
def filterRepeats(statuses):
    all_text = []
    filtered_statuses = []
    for s in statuses:
        if not s in all_text:
            filtered_statuses.append(s)
            all_text.append(s)
    return filtered_statuses
```

In [ ]:
```
everywhereTexts = getStatuses(search, "everywhereSearch")
#print(everywhereTexts)
print(len(everywhereTexts))
everywhereTextsFiltered = filterRepeats(everywhereTexts)
print(len(everywhereTextsFiltered))
```

I'm not entirely sure why this filtere didn't seem to work (it basically never changed the length of the tweet list), except that maybe since retweets had already been excluded, there were few identical repeats. Something I also noticed is that when there were apparent repeats, they often had URLs that were different. For example, there might be two shortened URLs that pointed to the same place, but were perhaps different so that the clicks could be tracked.

In [ ]:
```
everywhereTexts[:5]
```

In [ ]:
```
chicagoTexts = getStatuses(search, "chicagoSearch")
print(len(chicagoTexts))
chicagoTextsFiltered = filterRepeats(chicagoTexts)
print(len(chicagoTextsFiltered))
#print(chicagoTextsFiltered)
```

I wasted a lot of time clicking on links in the tweets printed by this line :)

In [ ]:
```
chicagoTexts[:5]
```

In [ ]:
```
everywhereTexts[:5]
```

I notice that my search is also finding things that are unrelated to Pokemon Go but just have those two words in them. For example:

"@ChildrensITV It won't let me watch pokemon sun and moon on ITV Hub. It says unavailable when I click on go on the app?" or 'When you go in the tall grass without your starting pokemon. https://t.co/dg9lrnsRp0 (https://t.co/dg9lrnsRp0)'

Training a twitter sentiment classifier

I trained a classifier in the same way that was shown in the MOOC examples. I trained it on the NLTK twitter sample corpus, which already tokenized the tweets.

In [ ]:
```
nltk.download("twitter_samples")
from nltk.corpus import twitter_samples
```

In [ ]:
```
len(twitter_samples.fileids())
```

In [ ]:
```
print(twitter_samples.fileids())
```

In [ ]:
```
# I am attempting to remove @s and URLs since they are not real, useful word
def build_bag_of_words_features_filtered(words):
    bag = {}
    useless_words = nltk.corpus.stopwords.words("english") + list(string.pun
    for word in words:
        if not word in useless_words:
            if not "http" in word:
                if not "@" in word:
                    bag[word]=1
    return bag
```

In [ ]:
```
negstrings = twitter_samples.strings("negative_tweets.json")
#print(negstrings[:5])
```

In [ ]:
```
negtokens = twitter_samples.tokenized("negative_tweets.json")
postokens = twitter_samples.tokenized("positive_tweets.json")
#print(negtokens[:5])
```

In [ ]:
```
negbag = [build_bag_of_words_features_filtered(i) for i in negtokens]
negfeatures = [(bag, "neg") for bag in negbag]
posbag = [build_bag_of_words_features_filtered(i) for i in postokens]
posfeatures = [(bag, "pos") for bag in posbag]
```

In [ ]:
```
#print(len(negfeatures))
#print(len(posfeatures))
```

In [ ]:
```
from nltk.classify import NaiveBayesClassifier
```

In [ ]:
```
split = 4000
```

In [ ]:
```
sentiment_classifier = NaiveBayesClassifier.train(posfeatures[:split]+negfea
```

The model seems highly accurate, but this is likely because it is somewhat overfit in that the corpus is not representative of real tweets-- they are filtered to include emoticons.

In [ ]:
```
nltk.classify.util.accuracy(sentiment_classifier, posfeatures[split:]+negfea
```

It is highly overfit for identifying emoticons, which is how neg and pos were originally defined.

In [ ]:
```
sentiment_classifier.show_most_informative_features()
```

Determining positivity for the PoGo tweets

In [ ]:
```
chicagoTexts[0]
```

In [ ]:
```
def tokenizeTweets(tweetList):
    wordsList = []
    for tweet in tweetList:
        wordsList.append(nltk.word_tokenize(tweet))
    return wordsList
```

In [ ]:
```
chicagoTokens = tokenizeTweets(chicagoTexts)
```

In [ ]:
```
chicagoBag = [build_bag_of_words_features_filtered(i) for i in chicagoTokens
#print(chicagoBag[:3])
```

The following cell prints some sample tweets along with their probability of being positive. Based on these results, I didn't feel like my classifier was doing a great job. But I also realized that many tweets didn't have a particular obvious sentiment-- many were simply giving information about game updates.

In [ ]:
```
classifications = []
for tweet in chicagoBag:
    classifications.append(sentiment_classifier.prob_classify(tweet))
for i in range(10):
    print(chicagoTexts[i])
    print(classifications[i].prob("pos"))
```

In [ ]:
```
#dir(classifications[0])
```

Instead of actually classifying tweets as positive or negative, I use the probability of being positive as the score for the tweet. I averaged over all tweets collected in one session to get an overall approval rating for pogo or the pogo fest for that day.

In [ ]:
```
def approvalRating(classifList):
    runningScore = 0
    count = 0
    for tweet in classifList:
        runningScore += tweet.prob("pos")
        count += 1
    return 100*runningScore/count
```

In [ ]:
```
approvalRating(classifications)
```

After trying out my code, I made a pipeline that could be run for each group of saved tweets to process them from raw tweets into an average approval rating.

In [ ]:
```
#if repeatFilterOn is true, then this will filter repeats out of the tweets.
repeatFilterOn = False

def pipeline(query):
    scores = {}
    for place in query.keys():
        #print(place)
        if repeatFilterOn:
            statuses = filterRepeats(getStatuses(query, place))
        else:
            statuses = getStatuses(query, place)
        #print(statuses[0])
        bag = [build_bag_of_words_features_filtered(i) for i in tokenizeTwee
        #print(bag[0])
        classifications = []
        for tweet in bag:
            classifications.append(sentiment_classifier.prob_classify(tweet)
        #    print(classifications[-1:])
        nTweets = len(classifications)
        if nTweets == 0:
            print("No tweets saved; skipping")
        else:
            print("number of tweets: ", nTweets)
            score = approvalRating(classifications)
            #print(place, score)
            scores[place]=score
    print(scores)
    return scores
```

Using saved historical tweets to find trends over time

In [ ]:
```
files = !ls
datetimes = []
output = []
for filename in files:
    if "search" in filename:
        if "Csearch" not in filename:
            print(filename)
            searchresults = pickle.load(open(filename, "rb"))
            datetimes.append(datetime.strptime(filename[:-10], "%Y-%m-%d %H:
            output.append(pipeline(searchresults))
```

...

In [ ]:
```
data = {a:[], b:[], c:[], d:[], "dt":[]}
for i in range(len(output)):
    data[a].append(output[i]["a"])
    data[b].append(output[i]["b"])
    data[c].append(output[i]["c"])
    data[d].append(output[i]["d"])
    data["dt"].append(datetimes[i])
datadf =  pd.DataFrame.from_dict(data)
datadf.set_index(datadf["dt"], inplace = True)
datadf.pop("dt")
datadf.head()
```

In [ ]:
```
print(datadf.corr())
print()
print(datadf.describe())
```

I found it interesting that tweets related to the Go Fest event did not correlate all that well to general pokemon go tweets. Overall, the approval scores varied a lot over the month. The best day for pogo, with an approval of 85%, was 8/16. At the end of the analysis, I look into what happened that day.

In [ ]:
```
datadf[datadf["pokemongo"]>85]
```

I plotted the raw data as points, lines, and then also as a rolling average to smooth out the high variance. The open space is when I was on vacation and didn't collect data. There are no blue dots/lines because they are all identical to and written over by the orange (since the search is case insensitive).

In [ ]:
```
del datadf["#pokemongofest"]
datadf.plot(style=".", ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
datadf.plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
pd.rolling_mean(datadf,3).plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
#, figsize=(15,10)

#plt.figure(figsize=(20,10))
#plt.plot_date(datadf, xdate=True, ydate=False)
```

I'm not sure why, but many of the early days found no tweets for Chicago. I can't remember now if I modified the query or if it started working better on its own.

In [ ]:
```
cdatetimes = []
coutput = []

files = !ls
for filename in files:
    if "Csearch" in filename:
        print(filename)
        searchresults = pickle.load(open(filename, "rb"))
        pipeline(searchresults)
        cdatetimes.append(datetime.strptime(filename[:-11], "%Y-%m-%d %H:%M:
        coutput.append(pipeline(searchresults))
```

...

In [ ]:
```
coutput[:5]
```

In [ ]:
```
cdata = {"chicagoSearch":[], "everywhereSearch":[], "dt":[]}
for i in range(len(coutput)):
    if "chicagoSearch" not in coutput[i].keys():
        cdata["chicagoSearch"].append(0)
    else:
        cdata["chicagoSearch"].append(coutput[i]["chicagoSearch"])
    cdata["everywhereSearch"].append(coutput[i]["everywhereSearch"])
    cdata["dt"].append(cdatetimes[i])

cdatadf =  pd.DataFrame.from_dict(cdata)
cdatadf.set_index(cdatadf["dt"], inplace = True)
cdatadf.pop("dt")
cdatadf.head()
```

I graph using a ymin of 10, because these days with a rating of 0 are just because no tweets were collected in Chicago at those times.

In [ ]:
```
print(cdatadf.corr())
print()
print(cdatadf.describe())
```

In [ ]:
```
cdatadf.plot(style=".", ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
cdatadf.plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
pd.rolling_mean(cdatadf,3).plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
#, figsize=(15,10)

#x=cdatadf.plot(style=".", figsize=(15,10), ylim=[40,90])
#plt.figure(figsize=(20,10))
#plt.plot_date(cdata["dt"], cdata["everywhereSearch"],label = "everywhereSea
#plt.plot_date(cdata["dt"], cdata["chicagoSearch"],label = "chicagoSearch",
#plt.legend()
#plt.ylim([40,90])
```

Training a second twitter sentiment classifier

The previous classifier was probably not very accurate. It was based off a model that used emoticons to define the "gold standard," so the most important features for the classifier were :) and :(emoticons by far. I expect that this makes tweets without emoticons hard to analyze. At the same time, many of the tweets that I've seen could better be thought of as informative rather than emotive, so it's hard to know what kind of sentiment it should have.

Looking at some sample tweets, I don't think I would have given positive rating predictions like the classifier, for example:

I liked a @YouTube video https://t.co/3mnvMLL74a (https://t.co/3mnvMLL74a) This Problem with Pokémon Go NEEDS to be Solved NOW... 0.7723498203721197

I found a second corpus of tweets online to try out: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/ (http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/)

In [ ]:
```
pwd = !pwd
print(pwd)
if not pwd[0] == "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP
    %cd "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP-Database
trainingData=pd.read_csv("SentimentAnalysisDataset.csv")
```

In [ ]:
```
trainingData.head()
```

In [ ]:
```
del trainingData["ItemID"]
del trainingData["SentimentSource"]
```

In [ ]:
```
trainingData.head()
```

I found an NLTK tokenizer that is designed for tweets.

In [ ]:
```
tknzr = nltk.tokenize.casual.TweetTokenizer(preserve_case=False)
#tknzr.tokenize(trainingData["SentimentText"][0])
```

In [ ]:
```
trainingData["tokenizedbag"]=trainingData["SentimentText"].map(tknzr.tokeniz
```

In [ ]:
```
trainingData.head()
```

In [ ]:
```
negData = trainingData[trainingData["Sentiment"]==0]["tokenizedbag"]
posData = trainingData[trainingData["Sentiment"]==1]["tokenizedbag"]
```

In [ ]:
```
negbag = [(build_bag_of_words_features_filtered(i), "neg") for i in negData]
posbag = [(build_bag_of_words_features_filtered(i), "pos") for i in posData]
```

In [ ]:
```
#print(len(negbag))
#print(len(posbag))
```

In [ ]:
```
from nltk.classify import NaiveBayesClassifier
nsplit = int(.8*len(negbag))
psplit = int(.8*len(posbag))

sentiment_classifier2 = NaiveBayesClassifier.train(posbag[:psplit]+negbag[:n
print("Score on training set:")
print(nltk.classify.util.accuracy(sentiment_classifier2, posbag[:psplit]+neg
print("Score on test set:")
print(nltk.classify.util.accuracy(sentiment_classifier2, posbag[psplit:]+neg

sentiment_classifier2.show_most_informative_features()
```

While this sentiment classifier appears to have less accuracy, it's probably a more believable value than the other one's 99% accuracy (especially given that humans are only about 80% accurate).

Using the new classifier to do the same analysis

In [ ]:
```
pwd = !pwd
print(pwd)
if not pwd[0] == "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP
    %cd "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP-Database
# !ls
```

In [ ]:
```
repeatFilterOn = True
# I will tokenize using the same method as my training set this time.
def tokenizeTweets2(tweetList):
    wordsList = []
    for tweet in tweetList:
        wordsList.append(tknzr.tokenize(tweet))
    return wordsList

#chicagoTokens = tokenizeTweets(chicagoTexts)

#chicagoBag = [build_bag_of_words_features_filtered(i) for i in chicagoToker

#classifications = []
#for tweet in chicagoBag:
#    classifications.append(sentiment_classifier.prob_classify(tweet))
#for i in range(10):
#    print(chicagoTexts[i])
#    print(classifications[i].prob("pos"))

def approvalRating(classifList):
    runningScore = 0
    count = 0
    for tweet in classifList:
        runningScore += tweet.prob("pos")
        count += 1
    return 100*runningScore/count

#approvalRating(classifications)

def pipeline2(query):
    scores = {}
    for place in query.keys():
        #print(place)
        if repeatFilterOn:
            statuses = filterRepeats(getStatuses(query, place))
        else:
            statuses = getStatuses(query, place)
        #print(statuses[0])
        bag = [build_bag_of_words_features_filtered(i) for i in tokenizeTwee
        #print(bag[0])
        classifications = []
        for tweet in bag:
            classifications.append(sentiment_classifier2.prob_classify(tweet
        #    print(classifications[-1:])
        nTweets = len(classifications)
        if nTweets == 0:
            print("No tweets saved; skipping")
        else:
            score = approvalRating(classifications)
            #print("number of tweets: ", nTweets)
            #print(place, score)
            scores[place]=score
    print(scores)
    return scores

# Using saved historical tweets to find trends over time
```

```
files = !ls
datetimes = []
output = []
for filename in files:
    if "search" in filename:
        if "Csearch" not in filename:
 #           print(filename)
            searchresults = pickle.load(open(filename, "rb"))
            datetimes.append(datetime.strptime(filename[:-10], "%Y-%m-%d %H:
            output.append(pipeline2(searchresults))


data2 = {a:[], b:[], c:[], d:[], "dt":[]}
for i in range(len(output)):
    data2[a].append(output[i]["a"])
    data2[b].append(output[i]["b"])
    data2[c].append(output[i]["c"])
    data2[d].append(output[i]["d"])
    data2["dt"].append(datetimes[i])
datadf2 =  pd.DataFrame.from_dict(data2)
datadf2.set_index(datadf2["dt"], inplace = True)
datadf2.pop("dt")
datadf2.head()

print(datadf2.corr())
print()
print(datadf2.describe())

#datadf[datadf["pokemongo"]>85]


cdatetimes = []
coutput = []

files = !ls
for filename in files:
    if "Csearch" in filename:
        print(filename)
        searchresults = pickle.load(open(filename, "rb"))
        pipeline2(searchresults)
        cdatetimes.append(datetime.strptime(filename[:-11], "%Y-%m-%d %H:%M:
        coutput.append(pipeline2(searchresults))

coutput[:5]

cdata2 = {"chicagoSearch":[], "everywhereSearch":[], "dt":[]}
for i in range(len(coutput)):
    if "chicagoSearch" not in coutput[i].keys():
        cdata2["chicagoSearch"].append(0)
    else:
        cdata2["chicagoSearch"].append(coutput[i]["chicagoSearch"])
    cdata2["everywhereSearch"].append(coutput[i]["everywhereSearch"])
    cdata2["dt"].append(cdatetimes[i])

cdatadf2 =  pd.DataFrame.from_dict(cdata2)
cdatadf2.set_index(cdatadf2["dt"], inplace = True)
```

```
cdatadf2.pop("dt")
cdatadf2.head()

print(cdatadf2.corr())
print()
print(cdatadf2.describe())

#, figsize=(15,10)

#x=cdatadf.plot(style=".", figsize=(15,10), ylim=[40,90])
#plt.figure(figsize=(20,10))
#plt.plot_date(cdata["dt"], cdata["everywhereSearch"],label = "everywhereSea
#plt.plot_date(cdata["dt"], cdata["chicagoSearch"],label = "chicagoSearch",
#plt.legend()
#plt.ylim([40,90])
```

...

In [ ]:
```
del datadf2["#pokemongofest"]
datadf2.plot(style=".", ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
datadf2.plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
pd.rolling_mean(datadf2,3).plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
#, figsize=(15,10)

#plt.figure(figsize=(20,10))
#plt.plot_date(datadf, xdate=True, ydate=False)

#plt.figure(figsize=(20,10))
#plt.plot_date(data["dt"], data[a],label = a, xdate=True, ydate=False)
#plt.plot_date(data["dt"], data[b],label = b, xdate=True, ydate=False)
#plt.plot_date(data["dt"], data[c],label = c, xdate=True, ydate=False)
#plt.plot_date(data["dt"], data[d],label = d, xdate=True, ydate=False)
#plt.legend()
```

In [ ]:
```
cdatadf2.plot(style=".", ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
cdatadf2.plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
pd.rolling_mean(cdatadf2,3).plot(ylim=[10,90])
plt.xlabel("Date of query")
plt.ylabel("Positivity")
```

Comparing the two classifiers

To get a sense of which classifier performed better, I asked them to each classify some tweets. I did this on the chicagoBag because it was already easily available in the memory, however it's probably important to note that it is not tokenized in the same way as either of these models' training data were, which might skew things. I took this information and tried to classify some of the tweets as positive or negative myself to calculate the RMSE and log-likelihood for each classifier to see which is better. They both had several hits and misses with pretty similar RMSEs (0.480237111 vs 0.488219941 for the first and second classifier, respectively) and log-likelihoods (-7.958536425 vs -7.708474697).

Although based on these niether classifier seems hugely better than the other, if I had to choose one for further use I would probably use the second given that it was based on a more representative and much larger dataset.

In [ ]:
```
for i,tweet in enumerate(chicagoBag[:10]):
    print(chicagoTexts[i])
    print(sentiment_classifier.prob_classify(tweet).prob("pos"))
    print(sentiment_classifier2.prob_classify(tweet).prob("pos"))
```

Exploring specific days

In [ ]:
```
pwd = !pwd
if not pwd[0] == "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP
    %cd "/Users/hjohnsen/Dropbox (Personal)/Data Science/Week-8-NLP-Database
# !ls
```

Earlier, I found that approval of pogo peaked on August 16th. I couldn't identify clearly what announcements had been made that day by searching the web, so I wanted to look at the tweets themselves:

In [ ]:
```
def tweetTester(tweet):
    tokenized = tokenizeTweets([tweet])
    bag = build_bag_of_words_features_filtered(tokenized[0])
    print(sentiment_classifier2.prob_classify(bag).prob("pos"))
```

In [ ]:
```
peakday="2017-08-16 16:45:17.978716search.pkl"

peakdaytweets = pickle.load(open(peakday, "rb"))
for key in peakdaytweets.keys():
    rtcount = 0
    othercount = 0
    peakstatuses= getStatuses(peakdaytweets, key)
    print(key+ "="*100)
    for i in peakstatuses:
        print(i)
        if "RT @Pokemon: Spotted: Shiny Pikachu," in i:
           rtcount +=1
        else:
            othercount +=1
        tweetTester(i)
    print(rtcount)
    print(othercount)
```

Pogo's "best" day was a day when the shiny versions of the pikachu family came out. The tweet "RT @Pokemon: Spotted: Shiny Pikachu, Pichu, and Raichu in #PokemonGO! Be on the lookout for these Shiny versions as you explore:" was highly positive and retweeted a lot! (This search didn't filter out retweets, unlike the Chicago vs Everywhere query.) With numerous retweets of this very positive post, it was a good day according to the sentiment classifier.

Next I looked at some of the tweets on the first day, which didn't have quite as bad of a score as I expected given all the tweets and posts I was reading online that day.

In [ ]:
```
firstday="2017-07-22 19:04:57.902835search.pkl"
firstdaytweets = pickle.load(open(firstday, "rb"))
for key in firstdaytweets.keys():
    firststatuses= getStatuses(firstdaytweets, key)
    print(key+ "="*100)
    for i in firststatuses:
        print(i)
        tweetTester(i)
```

I realized that I didn't filter out retweets or foreign languages in this search query, but my sentiment analyzer has no idea what to do with that. I was curious why my analyzer was even giving it a positive score instead of giving a 0.5 or something, and I found that just "RT" on its own seems to be a positive feature.

Additionally, by this time in the day, legendary pokemon had been announced, which buffered their approval somewhat.

In [ ]:
```
tweet = "RT @famitsu: 『Pokemon GO』に伝説のポケモン“ルギア”、“フリーザー”が登場！
tweetTester(tweet)
```

In [ ]:
```
tweet = "RT"
tweetTester(tweet)
```

In [ ]:
```
tweet = "米国のポケモンGOイベントで大規模サーバ障害が発生、運営元ナイアンティックはチケッ
tweetTester(tweet)
```

In [ ]:
```
firstday="2017-07-22 15:32:06.903443search.pkl"
firstdaytweets = pickle.load(open(firstday, "rb"))
for key in firstdaytweets.keys():
    firststatuses= getStatuses(firstdaytweets, key)
    print(key+ "="*100)
    for i in firststatuses:
        print(i)
        tweetTester(i)
```

In [ ]:
```
firstday="2017-07-22 17:40:45.802043search.pkl"
firstdaytweets = pickle.load(open(firstday, "rb"))
for key in firstdaytweets.keys():
    firststatuses= getStatuses(firstdaytweets, key)
    print(key+ "="*100)
    for i in firststatuses:
        print(i)
        tweetTester(i)
```

Earlier in the day you can see some of hte negative tweets. I think the worst was probably "Pretty sad how bad the Pokemon Go Chicago Event turned out. Cellular lines jammed up everywhere & people boo-ing the CEO of Niantic" with a 3e-05 probability of being positive!

In [ ]:
```
datadf[:10].plot(style=".", ylim=[10,90])
datadf[:10].plot(ylim=[10,90])
pd.rolling_mean(datadf[:10],3).plot(ylim=[10,90])
cdatadf[:10].plot(style=".", ylim=[10,90])
cdatadf[:10].plot(ylim=[10,90])
pd.rolling_mean(cdatadf[:10],3).plot(ylim=[10,90])
```

In [ ]:
```

```

In [ ]:
```
# This line is commented out because there is too much text in this notebool

#sentiment_classifier2.show_most_informative_features(10000)
```

According to the above, retweets are positve:

rt = 1 pos : neg = 2.7 : 1.0

In [ ]:
```
    tokenized = tokenizeTweets(["RT"])
    bag = build_bag_of_words_features_filtered(tokenized[0])
    print(sentiment_classifier2.prob_classify(bag).prob("pos"))
```

In [ ]:
```
    tokenized = tokenizeTweets(["RT"])
    bag = build_bag_of_words_features_filtered(tokenized[0])
    print(sentiment_classifier.prob_classify(bag).prob("pos"))
```

In [ ]:
```
len(datadf)
```

In [ ]:
```

```

In [ ]:
```
bag
```

In [ ]:
```
    tokenized = tokenizeTweets(["RT"])
    bag = build_bag_of_words_features_filtered(tokenized[0])
    print(sentiment_classifier.classify(bag))
```

In [ ]:
```
x = []
y = []
for i,tweet in enumerate(chicagoBag[:50]):
    print(chicagoTexts[i])
    print(sentiment_classifier.prob_classify(tweet).prob("pos"))
    print(sentiment_classifier2.prob_classify(tweet).prob("pos"))
    x.append(sentiment_classifier.prob_classify(tweet).prob("pos"))
    y.append(sentiment_classifier2.prob_classify(tweet).prob("pos"))
plt.scatter(x,y)
plt.xlabel("Classifier 1 positivity score")
plt.ylabel("Classifier 2 positivity score")
```

In [ ]:
```

```