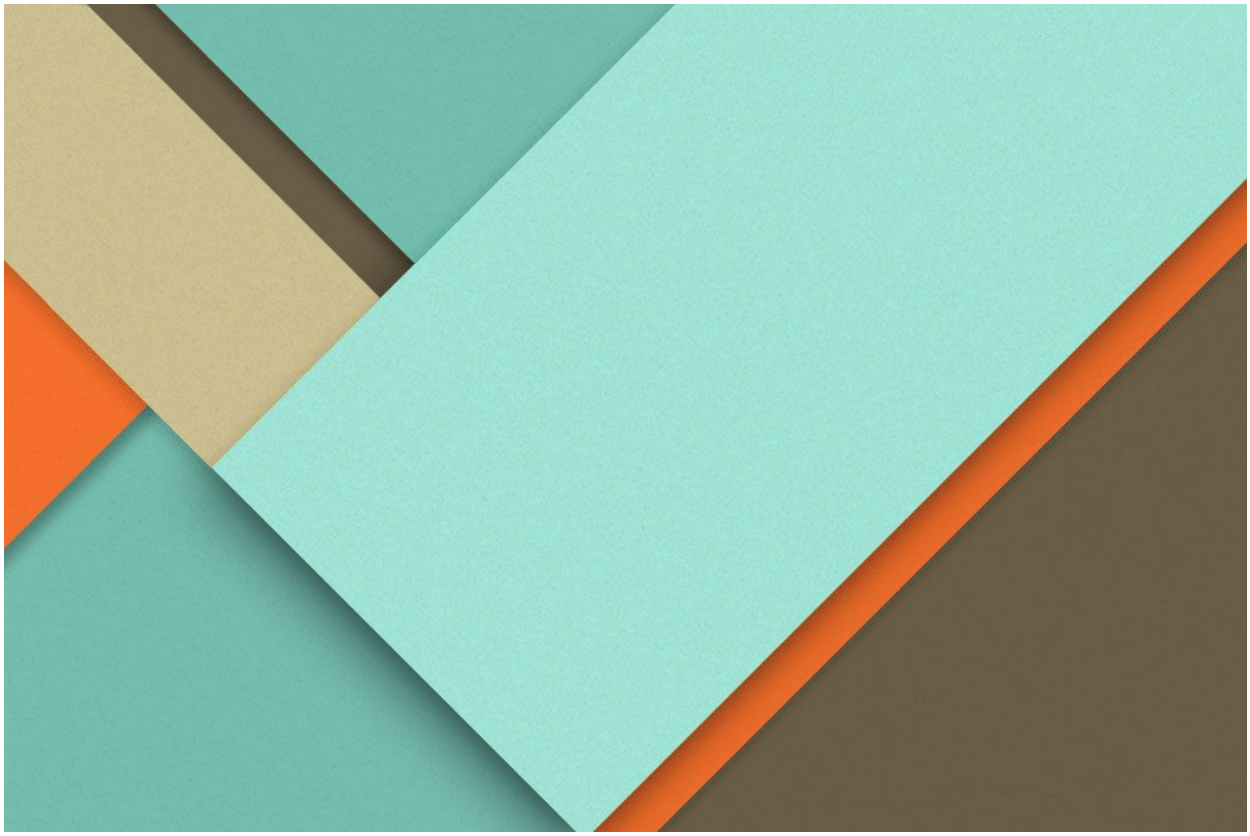


Regional differences in America

05. 05. 2019

Yili Luo, Jiayou Zhao, Minrui Zhang,Yuwen Wu,Haoyan Wang,Qianru Zhao
Syracuse University
900 S Crouse Ave
Syracuse, NY 13210

Background

In the US, there are over 50 states which may be different in weather, political opinions, and other daily life preferences. In general, such differences may not affect people significantly. However, for enterprises, such as auto dealers, manufactures, and other companies, even a tiny difference can result in huge distincts in marketing strategies. For example, Auto dealers may find out that people in warmer states like California is more likely to buy muscle cars, while on the other hand, people in states snowing frequently is more willing to buy all-wheel-drive SUVs. As result, generating strategies according to demands is one of the most significant factors for companies.

It's estimated that 80% of the world's data is unstructured and not organized in a pre-defined manner. Most of this comes from text data, like emails, support tickets, chats, social media, surveys, articles, and documents. These texts are usually difficult, time-consuming and expensive to analyze, understand, and sort through.

Sentiment Analysis, also known as Opinion Mining, is a field within Natural Language Processing (NLP) that builds systems to identify and extract opinions within text. is contextual mining of text which identifies and extracts subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations. However, analysis of social media streams is usually restricted to just basic sentiment analysis and count based metrics. This is akin to just scratching the surface and missing out on those high value insights that are waiting to be discovered. So what should a brand do to capture that low hanging fruit?

With the recent advances in deep learning, the ability of algorithms to analyse text has improved considerably. Creative use of advanced artificial intelligence techniques can be an effective tool for doing in-depth research. We believe it is important to classify incoming customer conversation about a brand based on following lines:

- Key aspects of a brand's product and service that customers care about.
- Users' underlying intentions and reactions concerning those aspects.

These basic concepts, when used in combination, become a very important tool for analyzing millions of brand conversations with human level accuracy. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

- Polarity: if the speaker express a positive or negative opinion,
- Subject: the thing that is being talked about,
- Opinion holder: the person, or entity that expresses the opinion.

Project Overview

This project is to explore and analyze the regional differences in United States. By collecting data from social network, we can reasonably identify regional preference among different states. Sentiment analysis will be the most effective method to help us find out the reason behind those facts. After getting the results, we can give general advice to correlated enterprises to help them catch the wave before it's gone.

In this project, we'll classify data collected from twitter by topic and by location and then perform sentiment analysis, which allows us to obtain the overall opinion among each state about each selected topic.

Sentiment Analysis Tools Comparison

I. TextBlob

TextBlob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and so on.

II. NLTK

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. It can provide you with different dataset in multiple languages which you can deploy according to the functionality required.

NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

III. Naive Bayes Classifier

Naive Bayesian Method is a classification method based on Bayes' Theorem and Characteristic Condition Independent Hypothesis. Compared with the decision tree model, Naive Bayes Classifier (NBC) originates from classical mathematical theory and has a solid mathematical foundation and stable classification efficiency. At the same time,

the NBC model requires few parameters for estimation, is less sensitive to missing data, and the algorithm is relatively simple.

We have tried to implement a Naïve Bayes Classifier which can help us to classify the sentiment of each tweet. To train this model, we will follow these steps:

1). Get the train data (1 million tweets) and test data (1000 labelled tweets).
Pandas package is used here to read csv file.

```
train_data = pandas.read_csv('train.csv', nrows=1000000, encoding="ISO-8859-1")
test_data = pandas.read_csv('test.csv', nrows=10000, encoding="ISO-8859-1")
```

2). Remove @ symbol, network address, punctuations, stop words which are not useful for following training.
Regular expression is used here to remove the matched patterns. For the regular expression, the findall method in re package is used. Then do replace all the matched patterns.
Remove @ symbol. (the pattern for @ symbol is @[\w]*)
Remove network address. (the patterns for network address is http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+)
Remove punctuations. (use replace function, the pattern is ^[a-zA-Z#])
Remove stop words. (stop words is stored in the stop list)

3). Do tokenizing and stemming.
TweetTokenizer is used here as tokenizer.
PortStemmer is used here as stemmer.

4). Get all words features in train data.
First, get all appeared words in train data and store them as word and sentiment pair.
Second, find the features of train data's words by using most_common function. This is also an important parameters to adjust and optimize the Naive Bayes Classifier. We set the number of common words as 10000.

5). Extract features from train data.
Extract features of each test tweet by using above features of train data's words. If one word is contained in the test tweet, then set the corresponding feature as true. Otherwise, set it as false.

6). Train the Naïve Bayes Classifer in nltk package.

7). Store the classifier results in pickle file.

8). Naïve Bayes Classifier Results

Most informative features from train data

```
Last login: Thu Apr 25 22:34:51 on ttys000
wanghaoyandeMacBook-Pro:~ wanghaoyan$ cd /Users/wanghaoyan/Desktop/Sentiment\ analysis\ files/Sentiment_analysis_Naive_Bayes
wanghaoyandeMacBook-Pro:Sentiment_analysis_Naive_Bayes wanghaoyan$ python3 Naive_Bayes.py
Most Informative Features
            contains(sad) = True           negati : positi =     21.6 : 1.0
    contains(twittervers) = True           positi : negati =     20.0 : 1.0
        contains(goooood) = True           positi : negati =     17.3 : 1.0
        contains(hotword) = True           positi : negati =     17.3 : 1.0
           contains(poor) = True           negati : positi =     15.3 : 1.0
            contains(ach) = True           negati : positi =     14.8 : 1.0
          contains(yayyy) = True           positi : negati =     14.7 : 1.0
   contains(surprisingli) = True           positi : negati =     14.7 : 1.0
            contains(rad) = True           positi : negati =     14.7 : 1.0
            contains(dew) = True           positi : negati =     14.7 : 1.0
     contains(timberlak) = True           positi : negati =     14.7 : 1.0
           contains(ruin) = True           negati : positi =     13.9 : 1.0
           contains(hugh) = True           positi : negati =     13.6 : 1.0
            contains(fml) = True           negati : positi =     13.4 : 1.0
         contains(anymor) = True           negati : positi =     13.0 : 1.0
        contains(headach) = True           negati : positi =     12.5 : 1.0
     contains(disappoint) = True           negati : positi =     12.5 : 1.0
          contains(giggl) = True           positi : negati =     12.0 : 1.0
         contains(mmmmmm) = True           positi : negati =     12.0 : 1.0
      contains(throwdown) = True           positi : negati =     12.0 : 1.0
            contains(aid) = True           positi : negati =     12.0 : 1.0
          contains(lovee) = True           positi : negati =     12.0 : 1.0
          contains(efron) = True           positi : negati =     12.0 : 1.0
        contains(suspect) = True           positi : negati =     12.0 : 1.0
         contains(loooov) = True           positi : negati =     12.0 : 1.0
          contains(wahoo) = True           positi : negati =     12.0 : 1.0
       contains(giveaway) = True           positi : negati =     12.0 : 1.0
           contains(kewl) = True           positi : negati =     12.0 : 1.0
      contains(barcelona) = True           positi : negati =     12.0 : 1.0
            contains(yan) = True           positi : negati =     12.0 : 1.0
          contains(xoxox) = True           positi : negati =     12.0 : 1.0
    contains(nonetheless) = True           positi : negati =     12.0 : 1.0
        contains(funnier) = True           positi : negati =     12.0 : 1.0
          contains(nicki) = True           positi : negati =     12.0 : 1.0
```

Test data part results (text of the tweet and its sentiment classified)

```
read kindl love lee child good read : negative
first asses #kindl fuck rock : negative
love kindl mine month never look back new big one huge need remors : negative
fair enough kindl think perfect : negative
big quit happi kindl : negative
fuck thi economi hate aig non loan given ass : negative
jqueri new best friend : positive
love twitter : positive
not love obama make joke : negative
check thi video presid obama white hous correspond dinner : negative
firmli believ obama pelosi zero desir civil charad slogan want destroy conservat : negative
hous correspond dinner wa last night whoopi barbara amp sherri went obama got stand ovat : negative
watchin espn ju seen thi new nike commer puppet lebron wa hilari lmao : negative
dear nike stop flywir shit wast scienc ugli love : negative
#lebron best athlet gener not time basketbal relat don want get inter sport debat : negative
wa talk thi guy last night wa tell die hard spur fan also told hate lebron jame : negative
love lebron : negative
lebron beast still cheer til end : negative
lebron boss : negative
lebron hometown hero lol love laker let cav lol : negative
lebron zydruna awesom duo : negative
lebron beast nobodi nba come even close : negative
download app iphon much fun liter app anyth : negative
good news call visa offic say everyth fine relief sick scam steal : negative
awesom come back via : positive
montreal long weekend amp much need : negative
booz allen hamilton ha bad ass homegrown social collabor platform way cool #ttiv : positive
#mluc custom innov award winner booz allen hamilton : positive
current use nikon love not much canon chose video featur mistak : negative
need suggest good filter canon got pl : negative
check googl busi blip show second entri huh good emhv : positive
googl alway good place look mention work mustang dad : negative
play android googl phone slide screen scare would break fucker fast still prefer iphon : negative
plan resum militari tribun guantanamo bay onli thi time trial aig exec chrysler debt holder : positive
```

IV. Test Accuracy Comparison

Accuracy for Naive Bayes Classifier, it is around 0.48.

```
lam love bobbi flay favorit need place phoenix great pepper : positive
creat first latex file scratch didn work veri well see great time waster : positive
use linux love much nicer window look forward use wysiwyg latex editor : positive
use latex lot ani typeset mathemat look hideou : negative
ask program latex indesign submit calcio link comment : positive
note hate word hate page hate latex said hate latex texn rd come kill : negative
ahhh back real text edit environ latex : negative
troubl iran see hmm iran iran far away #flockofseagullsweregeopoliticallycorrect : negative
read tweet come iran whole thing terrifi incred sad : negative
love thi car : negative
thi view amaz : negative
feel great thi morn : negative
excit concert : positive
best friend : negative
not like thi car : negative
thi view horribl : negative
feel tire thi morn : negative
not look forward concert : negative
enemi : negative
accuracy is 0.485207
```

Then we compare the accuracy of Naïve Bayes Classifier and textblob. With the same testing dataset, the accuracy of textblob library is around 0.85.

```
accuracy for textblob is 0.850888
```

Textblob has a higher accuracy which is a suitable model for following analysis.

V. Result

As we can see from the testing accuracy comparison above, Textblob sentiment analysis API provides a much higher accuracy(0.85) than our model with Naive Bayes Classifier(0.49).

When comparing with other libraries, another widely used library will be NLTK. Both NLTK and Textblob are excellent libraries in the field of Natural Language Processing. However, as it's declared in the documentation, Textblob stands on the giant shoulders of NLTK and pattern, and plays nicely with both. In other words, Textblob is design to gain more benefits which does not included in NLTK. Since it's easy to use and has a good interface and excellent documentation, we finally decided to introduce TextBlob library into our project as the sentiment analysis toolkit.

Implementation

Overall, the project can be divided into three parts: data preparation, sentiment analysis, and result analysis. The first two parts were completed using python in jupyter notebook.

I. Crawling

Design Spec

The very first part is data preparation. Good data is important to perform efficient analysis later. In this section, we will focus on design spec, which is that what we wanted to achieve before we actually started and how.

What we wanted to do here is that we were trying to find the relation between people's daily preference and the region they live in. As result, we prefer real-time and data-intensive collection. As the most popular social media platform in the United State, twitter is a desired way to get started. Meanwhile, twitter API provides us a convenient tool. Since we decided to rely on the twitter API, that left us two possible ways to do that. One was using twitter Search API, and the another one was using twitter Streaming API. We chose to work with Streaming API, because Streaming provided real-time and data-intensive data collection, and that was exactly what we wanted.

Once we had made a choice on the tool. The next thing we considered was how to filtered the data we got. The Streaming API provided us a huge amount of data, while there was no need to store or deal with every single tweet regardless of whatever topic it was about. As result, here came the question. What standard should we use. Finally we decided to select data based on keywords and locations. Considering the topic of our project, analyzing the sentiment difference of people in various region on the same topic is the most reasonable and efficient way. At the same time, the Streaming API provided us exactly the functions we needed, which made the process rather straightforward and simple.

After data selecting, we had already collected some data to be analyzed. We could just feed the data into classifier and perform analysis. While the problem was that the raw data might contain lots of noise, such as emojis, misspelled words, some trivial words. To get a better result, we decided to do some preprocessing. In general, the process could be divided into several parts: tokenization, stemming and lemmatization, words deletion, emoji deletion, and spell checking.

The tokenization process would break each tweet into single tokens. Each token was a single word. It served later processes.

Stemming and lemmatization was a process that would transform a single word to its normal form. For example, the process will transform a word “books” to its normal form “book”. The reason to do that was to feed better data into the classifier to improve the efficiency and accuracy.

Word deletion was a process that would delete most of the common trivial words that did not affect the meaning of a sentence at all. For example, “and”, “or”, “we” are words that can be considered as trivial words.

Because most of the tweets contained emojis, and emojis by itself did not mean anything valuable, the emoji deletion process would just delete those words. To some extent, it was similar to word deletion

Spell checking was a process that would check and correct misspelled words. Misspelling is common in tweets, no matter the people did that unintentionally or intentionally. In rare case, excessively misspelling will distort the sentiment of a sentence being analyzed by machine learning. Even in common case that misspelling is not dominant, such misspelled words will also affect the efficiency of analysis.

The final step is to save the data which is simple.

Software Implementaion

Streaming API

The function of Streaming API is already defined in twitter API. We just called `twitter_api = oauth_login()` function from twitter cookbook to login. We then used the two functions below to start a connection.

```
twitter_stream = twitter.TwitterStream(auth=twitter_api.auth)
stream = twitter_stream.statuses.filter(track = q, locations=geo, language='en')
```

During the streaming, we filtered out tweets that contain “TT” and “RT”. Also, we had a counter to count the tweets we had collected, so that we can stop collecting once we reached the desired amount.

Tokenization

There was a function in nltk package. It mainly break a stream of characters into tokens is trivial for a person familiar with the language structure because certain characters are sometimes token delimiters and sometimes not, depending on the application. We just called that function with each raw tweet we collected as a parameter. That gave us a list of tokens for each single tweet. In total, we had a list of list of tokens after tokenizing every single tweets.

Stemming and lemmatization

Once a character stream (string) has been segmented into a sequence of tokens, the next step is to convert each of the tokens to a standard form (text normalization), this process usually referred to as stemming or lemmatization.

Still, the nltk package provided us these functions to use. What we did was just calling those functions.

```
lancaster = LancasterStemmer()
stems = [lancaster.stem(t) for t in token]
tokens_filtered = [w for w in stems if w.lower() not in stop and
w.lower() not in string.punctuation]
```

The code above set up a lemmatizer and performed lemmatization on each list of tokens.

Word deletion

Stopwords are common words that almost never have any predictive power for classifying texts, such as articles a and the and pronouns such as it and they.

```
stop = stopwords.words('english')
```

The function above gave us a list of common stopwords in english.

```
tokens_filtered = [w for w in stems if w.lower() not in stop and
w.lower() not in string.punctuation]
```

The code above also deleted common stop words.

Emoji deletion

For the emoji delete, we need to install the regular expression operation package.Then we can import and use it as the emoji deleter. The main function for this is to delete the emoji,symbols,transport and map symbols you want. And return the text without any emoji.

```
emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"
        u"\U0001F300-\U0001F5FF"
        u"\U0001F680-\U0001F6FF"
        u"\U0001F1E0-\U0001F1FF"
                           "]+", flags=re.UNICODE)
print(emoji_pattern.sub(r'', text))
```

Spell checking

First, we need to stall hunspell package in the system. Then we can import and use it as the spell checker. The main function is check recorded words' spell.

```
for w in add_to_dict:
     spellchecker.add(w)
corrected = []
for w in words:
  ok = spellchecker.spell(w)
```

If the word is misspelled, the spell checker will automatically correct it. If it meet some new words, after the correction, the new words will be added to the dictionary.

```
if not ok:
             suggestions = spellchecker.suggest(w)
     if len(suggestions) > 0:
          best = suggestions[0].decode(enc)
          corrected.append(best)
     else:
           corrected.append(w)
     else:
        corrected.append(w)

   return corrected
```

Problem encountered

There were problems, and we did solved them.

1. At first, we decided to use Search API because it can return the results in a short time. But after we tried to use it, we realized that it has limit data amount to collect. For example, we want to collect 2000 tweets. But it can only collect 200 tweets. After we tried it again, we decide to change it into the Streaming API. Streaming API is much better than Search API, it can collect large amount of real-time data. But it would take longer time.
2. After using the Streaming API. The first problem we encountered was that there was not a proper way to close the Streaming API once we started a connection. The reason was that there was predefined function to start a connection, but there was not such as function for closing. If we just let it run, it would reach rare limit at some point and crash the whole program. What we did here was that we set up a thread. Once we started a connection, we threw the streaming process into a thread, and let it run itself. We used a counter to count the amount of data that we had collected. Once we had collected enough data, we killed that thread by a helper function we defined ourselves.

3. During the stemming process, we were trying to use stemmer instead of lemmatizer. However, the problem was that the stemmer, even though powerful, sometimes left ugly output. For instance, the word "debate" would be transformed as "deb" which was not even a single word. Meanwhile, lemmatizer outputted better result, but there was tradeoff. Some of the word would not be handled by lemmatizer. We were not sure about the effects, but we decided to work with lemmatizer eventually.

II. Sentiment Analysis

After implementing the data crawling part, we will now start analyzing the sentiment behind these words using Textblob Library.

Software Implementaion

Sentiment Analysis API

The function of sentiment analysis API is already defined in Textblob library,. We can just call `analysis = TextBlob(new_line)` to load each single line of words and then it will return the polarity value of this sentence by calling built-in function: `polarity = analysis.sentiment.polarity`.

The sentiment property returns a namedtuple of the form Sentiment(polarity, subjectivity). The polarity score is a float within the range [-1.0, 1.0]. The subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

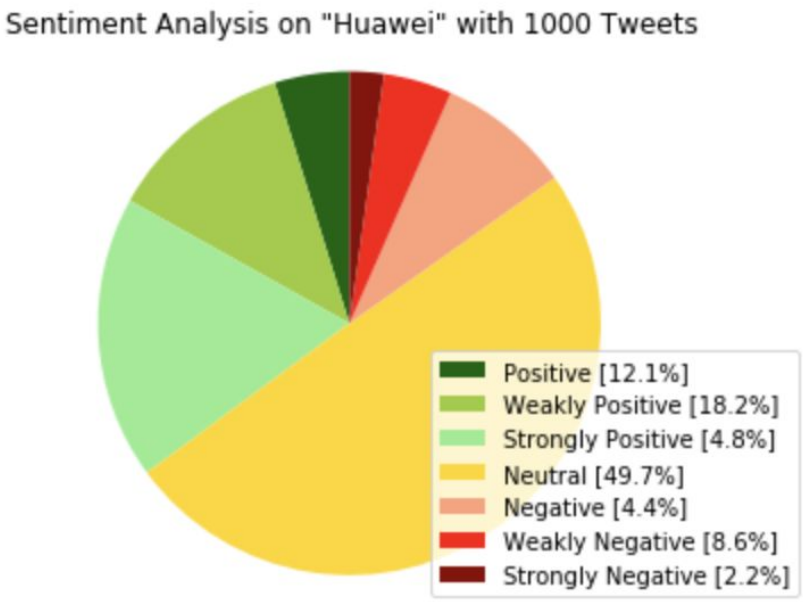
Polarity Calculation

To gain a more precise conclusion, we are recording both polarity of each single sentence and the total of all polarity values among all the tweets collected in this round to calculate the average polarity as the overall opinion. The polarity range was divided into seven different classes: Weak Positive (0, 0.3], Positive (0.3, 0.6], Strong Positive (0.6, 1.0], Neutral [0], Weak Negative[-0.3, 0), Negative [-0.6, -0.3) and Strong Negative[-1, -0.6).

Report Generation

In order to display our data results more intuitively, a report generation module was developed to create a detailed report with both a general idea among this topic in current region and the percentage of people in each class. A pie-chart is also plotted using Matplotlib to show comparisons between different categories.

```
Evaluation on "Huawei" based on 1000 tweets:

1. General Idea => Weak Positive

2. Detailed Report:

    12.10% people thought it's Positive.
    18.20% people thought it's Weakly Positive.
     4.80% people thought it's Strongly Positive.

     4.40% people thought it's Negative.
     8.60% people thought it's Weakly Negative.
     2.20% people thought it's Strongly Negative.

    49.70% people thought it's Neutral.
```



Data Collection

The figure below shows the test result we collected. We used keyword “starbucks”, and we put the boundbox around New York state.

There are three headers: index, sentiment, and content. At that time we had not integrated emoji deleters yet, so there were still emojis around. Each tweet was tokenized, and most words were transformed into its normal type. Those words “NULL” were just placeholder for now.

Index	sentiment	content
0	NULL	Mirror,mirror,wall,realest,em,..
1	NULL	like,crash,hey,semantics
2	NULL	``,Small,encampment,garbage,feces,'',SOMA,http,//t.co/F6xRjmzRrb,http,//t.co/vpYQiuuOQU
3	NULL	tonyg_nj,Defensively,yes,Overall,think,Stearns,Plus,Grote,tried,run,2,different,time,tried,getting,autograph
4	NULL	Together,make,difference,VNSNY,ha,open,employment,opportunity,various,department,invite,s…,http,//t.co/LvU07ikkz3
5	NULL	Ya,know,💜,SelenaQuintanilla,NYCLovesSelena,asked,Tribute,Show,Honor,the🎟,'ll,p…,http,//t.co/VvHWrZM0rX
6	NULL	NYC,LOVES,SELENA,--,gt,http,//t.co/85f5TuYopo
7	NULL	Darryl1960,dog,',easy,...
8	NULL	samcoryrose,excited,hear,whatever,wrongdoing,'re,referencing,next,week
9	NULL	Fix
10	NULL	ONE
11	NULL	Sonrisitass_M,Men,',shit,tho
12	NULL	Someone,love,much,😭😭
13	NULL	Stop,staring,clock,every,day,'s,time,new,career,Follow,u,view,job,title,like,``,Cashier-S…,http,//t.co/9YqPvxNSwW
14	NULL	Click,link,bio,see,currently,open,Accounting,job,like,``,Accountant,'',Serino,Coyne,LLC,NewYork,NY
15	NULL	Looking,score,job,Cellular,Sales,'re,luck,Click,link,bio,job,description,inf…,http,//t.co/ETkFBi2FjQ
16	NULL	RobinKinkster,think,``,Dom,'',side,like,incorporate,belief,concept,``,``,…,http,//t.co/VttOf2Kr5p
17	NULL	SnarkyAcademic_,Dont,',get,forehead,ache
18	NULL	Bennyy_Blanco,YaaBoyRay,make,',fine,🔮
19	NULL	',upside,🏠,',New,York,New,York,http,//t.co/7WiB1tHcuh
20	NULL	Damn,',even,lie,new,Starbucks,chocolate,chip,cookie,actually,taste,like,wa,made,love..,congrats
21	NULL	Heavy,cream,cappo,',🥤,Starbucks,Yelp,http,//t.co/F6QPJmXUNs
22	NULL	lot,Sundevils,across,pond,Starbucks,ASU,announced,expanding,their…,http,//t.co/Q2X5ZuBlJt
23	NULL	Girls,lacrosse,UPDATE,Centaurs,11,Fitch,5,...,Centaurs,score,4,goal,2,Gelhaus,first,5,1/2,minute,2nd,half
24	NULL	American,dreamus⚾🗽,nyc,en,New,York,New,York,http,//t.co/JqwZyGKH4O
25	NULL	NaijaBwoy,let,live,Lmao

Problem encountered

1. First we tried to collect data city by city Since we're using the streaming API, which collects tweets in real-time, when we tried to crawl tweets in some city/states with a small population, it might take quite a while to meet the minimum requirement of 1000 records.

2. Certain topics which are not popular may not meet the requirements. For example, we tried " seafood" in Michigan. At first, we're planning to collect 1000 tweets, but the results are only 680 tweets. But in another state, it may vary in a different way.

Data Analysis

After retrieving adequate data we need from every states, we can do further analysis based on the data set. We can manipulate the data by plotting it out then find correlations or by other means. What we do instead is create a table for us to better sort and filter data by different variables and to calculate easier. And here we are going to introduce the two interesting findings we have discovered.

Interesting finding(I) Attitudes toward prestigious universities

- **Overall Education in different states**

In our preliminary research, we came across a ranking of the education level of different states, we found that the least educated states are mostly Southern states and the most educated ones are mostly Northeastern states[1]. Therefore, we're interested in finding the differences between these states, as well as what do they have in common.

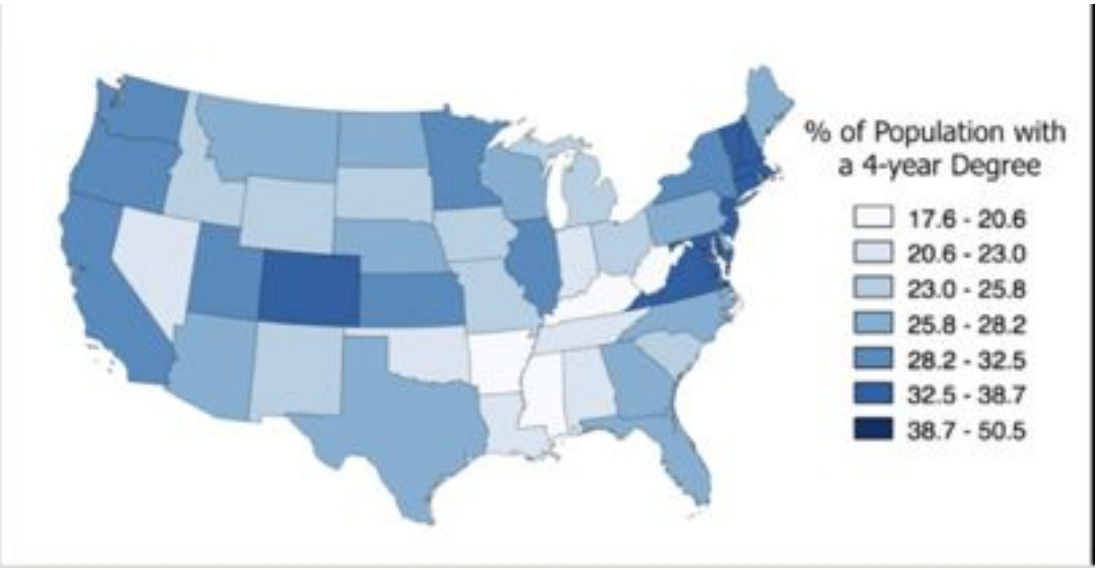


Figure 1, % of Population with a 4-year Degree in Different States

In **Figure 1**, we can see that percentage of population with a 4-year degree in Southern states are generally much lower than northeastern states and those in the west coast. The top 5 educated states are: Massachusetts, Maryland, Vermont, Connecticut, and Colorado. The bottom 5 educated states are: Mississippi, West Virginia, Louisiana, Arkansas, and Alabama.

We're interested in the regional difference about people's opinion towards universities, so we set a list of elite universities as key words such as: "Stanford University", "Yale University", "Princeton University" and so on. We crawled 1000 tweets from each state for each key word and here are some parts of data table we created:

[1] Adam McCann (2019). *Most & Least Educated States in America.* Retrieved from https://wallethub.com/edu/most-educated-states/31075/

Overall Rank	State	Positive(%)	Negative(%)	Keyword
1	Massachusetts	38.1	15.7	
2	Maryland	41.8	11.2	
3	Vermont	37.7	13	
4	Connecticut	38	11.8	
5	Colorado	40.4	10.1	
6	Virginia	39.5	14.5	
...	...	...	...	Stanford University
45	Kentucky	35.5	12.8	
46	Alabama	32	20.1	
47	Arkansas	30.2	15.5	
48	Louisiana	35.7	15.3	
49	West Virginia	37.6	14.1	
50	Mississippi	33.8	18.4	

Table 1, Stanford University

In **Table 1**, It shows that people in states with higher education level, people tend to have more positive opinions towards Stanford University than people who live in less education level states. Respectively, the states rank lower have relatively higher negative response towards Stanford University than those in the states with average higher education level.

Overall Rank	State	Positive(%)	Negative(%)	Keyword
1	Massachusetts	38	9.1	
2	Maryland	40.2	13	
3	Vermont	38.8	11.2	
4	Connecticut	37.9	10.6	
5	Colorado	39.2	11.7	
6	Virginia	37.6	14.5	
...	...	...	...	Yale University
45	Kentucky	33.1	11.5	
46	Alabama	36.1	18.4	
47	Arkansas	31.2	16.5	
48	Louisiana	35.6	13.3	
49	West Virginia	31.4	13.3	
50	Mississippi	30.2	19.8	

Table 2, Yale University

Overall Rank	State	Positive(%)	Negative(%)	Keyword
1	Massachusetts	40.2	15.5	Princeton University
2	Maryland	40.3	12	
3	Vermont	37.7	9.9	
4	Connecticut	38.1	13.5	
5	Colorado	36.7	11.1	
6	Virginia	37.8	12.5	
...	...	...	...	
45	Kentucky	30.5	17.2	
46	Alabama	38.4	14.1	
47	Arkansas	31	16.6	
48	Louisiana	33.3	18.5	
49	West Virginia	34.6	11.6	
50	Mississippi	32.6	15.9	

Table 3, Princeton University

In **Table2**, and **Table 3**, we did the same test and the results is similar. Sometime it appears some exceptions, like the data in Alabama state. The positive value in Alabama is much higher than the average value from the bottom 5 states. But when searching with other keywords, values of Alabama are close to other states, so we assume this will not affect the overall conclusion.

In conclusion, people in states with higher education level in average are more likely to hold a positive view on those world-class elite universities, and the fewer portion of them are holding a negative view on prestigious universities.

Objectively, there are other elements (location of the testing universities, the time period we did the crawling for example) affecting the results. But all the testing universities are world-class elite universities, the results can still relatively show objective results of people's attitudes towards higher education and elite institutions.

Interesting Findings(II) Word expression preference

State	Strong Positive	Strong Negative	Neutral	Keyword
New York	5.5	1.1	48.5	Bagel
Maine	4.1	1.2	49.4	
Vermont	3.2	1.8	40.8	
Pennsylvania	3.8	1.5	50.2	
Connecticut	3.5	2	40.2	
South Carolina	7	5.2	50.1	
Tennessee	6.5	3.8	49.5	
Texas	7.2	5.5	48.8	
Georgia	7.5	7	51.2	
Alabama	7.9	4.5	55.3	

Table 4, Bagel

We used another list of neutral words as keywords like “Bagel”, “Cookie”, “Paper towel” which we did not expect much difference to do the test. Use Data retrieved from the keyword “Bagel” as an example in Table 4.

In our findings, the percentage of positive and negative for each state do not show much difference from each other, but there is another interesting finding among these data. In Alabama, Georgia, and South Carolina, the strong positive value is always higher than the average. Meanwhile, their strong negative value is also always higher than the average (In **Table 4**, highlighted part are the states whose strong sentiment occurs more frequent).

People who live in South Carolina, Georgia, Texas, Tennessee, and Alabama have much stronger emotion towards “Bagel”, “Cookie” and “Paper towel”, including both positive and negative emotions. The table also shows that the percentage of people with neutral attitudes are very similar between all the states. We think the reason behind might be the cultural difference between the North and the South. People in the South often “exaggerate” more about their feeling, so they tend to use stronger words like “very”, “absolutely” etc, which lead a higher value of their strong sentiments.

Summary

In general, in this project we refined the data collect system. The whole system comprises a couple of tasks such as Streaming API, Naive Bayes Classifier, Polarity Calculation, Report Generation, Stemming and lemmatization, and Stop words. While the system is running, data are shared in the whole system. At the same time, in order to generate report, a report generation module was developed to create a detailed report with both a general idea among this topic in current region and the percentage of people in each class. A pie-chart is also plotted using Matplotlib to show comparisons between different categories. For the accuracy, we are recording both polarity of each single sentence and the total of all polarity values among all the tweets collected in this round to calculate the average polarity as the overall opinion. The results, after testing, achieved all of the requirements.

Conclusion

As a whole, in this project gives us a better understanding of the regional differences in United States. In the meantime, our team also learn how to use Streaming API,Search API and Naive Bayes Classifier. Some encountered problems are complicated. But we discussed together and finally figured out. We gained a lot of experiences on how to tokenize, use stopwords and lemmatize.

For marketing purposes, we can use this data mining project to collect people's attitudes towards certain products, then tailor our marketing strategy based on different geographical areas. For example, by capturing the keywords that customers feel strongly negative with, service industries can better target individuals thus improve their service. Meanwhile, industries can use certain words trigger stronger emotion in their advertisements to attract people who live in an area where they tend to show stronger emotions. Accordingly, this can help companies to design corresponding marketing strategy based on the preference of target customer segment.

In addition, we can use this as a tool to track customer sentiment over time, gather data for further analysis. For instance, we can track people's attitudes before and after marketing campaign, or changing in specific product interface, or adjusting price structure from which can reflect market reaction to some extent. Helping industries know about their customers and their requirements, preference better and deeper.

Appendices

In [1]:
```
# -*- coding:utf-8 -*-

import matplotlib.pyplot as plt
import networkx as nx
from Cookbook import oauth_login, get_friends_followers_ids, get_user_profile, make_twitter_request
from builtins import sorted, reversed
import twitter
import threading
import time
import inspect
import ctypes
import string
import csv
```

In [2]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

from nltk.stem.wordnet import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /Users/jojo/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/jojo/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/jojo/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:
```
True
```

In [3]:
```
def _async_raise(tid, exctype):
    """raises the exception, performs cleanup if needed"""
    tid = ctypes.c_long(tid)
    if not inspect.isclass(exctype):
        exctype = type(exctype)
    res = ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, ctypes.py_object(exctype))
    if res == 0:
        raise ValueError("invalid thread id")
    elif res != 1:
        ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, None)
        raise SystemError("PyThreadState_SetAsyncExc failed")

def stop_thread(thread):
    _async_raise(thread.ident, SystemExit)
```

In [4]:
```
def Get_Tweets(keyword, geoCode):
    twitter_api = oauth_login()

    ## Search Tweets by Keyword(s)
    twitter_stream = twitter.TwitterStream(auth=twitter_api.auth)
    stream = twitter_stream.statuses.filter(track=keyword, locations=geoCode, language='en')

    i = 0

    for tweet in stream:
        if ('text' not in tweet): continue
        if 'TT' not in tweet['text'] and 'RT' not in tweet['text']:
            # remove links from the tweets
            text = re.sub(r"(?:\@|https?\://)\S+", "", tweet['text'])
            result.append(Remove_Emoji(text))

            if(i % 50 == 0): print(i)
            i = i + 1
```

In [5]:
```
import emoji

def Remove_Emoji(text):
    allchars = [str for str in text]
    emoji_list = [c for c in allchars if c in emoji.UNICODE_EMOJI]
    clean_text = ' '.join([str for str in text.split() if not any(i in str for i in emoji_list)])
    return clean_text
```

In [6]:
```
def Stem_Lemm():

    ## tokenize
    for tweet in result:
        token_t = word_tokenize(tweet)
        tokenized.append(token_t)

    ## stem & remove stopworda
    stop = stopwords.words('english')
    for token in tokenized:
        wnl = WordNetLemmatizer()
        stems = [wnl.lemmatize(t) for t in token]
        tokens_filtered = [w for w in stems if w.lower() not in stop and w.lower() not in string.punctuation]
        tokenized[tokenized.index(token)] = tokens_filtered
```

In [7]:
```
def Save_To_Csv(filename):

    with open(filename,'w') as f:
        writer=csv.writer(f)

        for t in tokenized:
            data=','.join(t)
            writer.writerow([data])
```

In [8]:
```
def Tweet_Crawler(keyword, geoCode, tweetNum, csvFile):
    # start threading
    t = threading.Thread(target=Get_Tweets, args=(keyword, geoCode))

    t.start()
    print("Thread Start")

    ## waiting for data collection
    while len(result) < tweetNum:
        pass

    ## kill thread
    stop_thread(t)
    print("Thread End")

    ## normalize noise
    Stem_Lemm()

    ## save to csv
    Save_To_Csv(csvFile)
```

In [9]:
```
import sys,re
from textblob import TextBlob
import matplotlib.pyplot as plt
```

In [10]:
```
class SentimentAnalysis:

    def __init__(self, keyword, geoCode, tweetNum, csvFile):
        self.keyword = keyword
        self.geoCode = geoCode
        self.tweetNum = tweetNum
        self.csvFile = csvFile

    def Get_Polarity(self, p):
        if (p == 0): return "Neutral"
        elif (0.0 < p <= 0.3): return "Weak Positive"
        elif (0.3 < p <= 0.6): return "Positive"
        elif (0.6 < p <= 1.0): return "Strong Positive"
        elif (-0.3 < p <= 0): return "Weak Negative"
        elif (-0.6 < p <= -0.3): return "Negative"
        elif (-1.0 <= p <= -0.6): return "Strong Negative"
```

```
def Analysis(self):
 # creating some variables to store info
    polarity, total_polarity = 0, 0
    dic = {"Strong Positive": 0, "Positive": 0, "Weak Positive": 0, "Neutral": 0, \
           "Weak Negative": 0, "Negative": 0, "Strong Negative": 0}

    new_line = []

    # read in the collected tweets in csv file
    with open(self.csvFile) as csv_file:
        content = csv_file.read()
        i = 0

    for tweet in (content).split('\n'):
        # skip the last line (empty space due to the split function)
        if(i >= self.tweetNum): break

        new_line = ' '.join(tweet.split(','))

        # use TextBlob to calculate polarity of current tweet
        analysis = TextBlob(new_line)
        polarity = analysis.sentiment.polarity

        # add the total polarity to get the general opinion
        total_polarity += polarity
        current = self.Get_Polarity(polarity)
        if(current not in dic): print(current, polarity)
        dic[current] += 1

        i += 1

    # calculate an overall opinion among all these tweets by taking average
    general = total_polarity / float(self.tweetNum)
    print("\n1. General Idea =>", self.Get_Polarity(general))
```

```
    # calculate the percentage of each opinion
    for num in dic:
        dic[num] = 100 * dic[num] / float(self.tweetNum)

    # print out a more detailed report
    print("\n2. Detailed Report: \n")

    print("{:>9,.2f}% people thought it's Positive.".format(dic["Positive"]))
    print("{:>9,.2f}% people thought it's Weakly Positive.".format(dic["Weak Positive"]))
    print("{:>9,.2f}% people thought it's Strongly Positive.\n".format(dic["Strong Positive"]))

    print("{:>9,.2f}% people thought it's Negative.".format(dic["Negative"]))
    print("{:>9,.2f}% people thought it's Weakly Negative.".format(dic["Weak Negative"]))
    print("{:>9,.2f}% people thought it's Strongly Negative.\n".format(dic["Strong Negative"]))

    print("{:>9,.2f}% people thought it's Neutral.\n".format(dic["Neutral"]))

    # write result to output file

    '''
    header = "Keyword(s)", "Geocode", "Number of Tweets",
             "Positive", "Weak Positive", "Strong Positive",
             "Negative", "Weak Negative", "Strong Negative", "Neutral"
    '''

    with open('results.csv', 'a', newline='') as csvfile:
        writer = csv.writer(csvfile, delimiter='\t')

        writer.writerow([self.keyword]+[self.geoCode]+[self.tweetNum]+[dic["Positive"]]+[dic["Weak Positive"]]+\
                        [dic["Strong Positive"]]+[dic["Negative"]]+[dic["Weak Negative"]]+[dic["Strong Negative"]]+\
                        [dic["Neutral"]])
        csvfile.close()

    self.Draw_Pie_Chart(dic)

def Draw_Pie_Chart(self, dic):
    if not dic: return
    labels = ['Positive [' + str(dic["Positive"]) + '%]', 'Weakly Positive [' + str(dic["Weak Positive"]) + '%]',\
              'Strongly Positive [' + str(dic["Strong Positive"]) + '%]', 'Neutral [' + str(dic["Neutral"]) + '%]',\
              'Negative [' + str(dic["Negative"]) + '%]', 'Weakly Negative [' + str(dic["Weak Negative"]) + '%]', \
              'Strongly Negative [' +  str(dic["Strong Negative"]) + '%]']
```

```
        # define the value and color of each pie
        pies = dic.values()
        colors = ['darkgreen','yellowgreen','lightgreen', 'gold', 'lightsalmon','red','darkred']
        patches, texts = plt.pie(pies, colors=colors, startangle=90)

        plt.legend(patches, labels, loc="best")
        plt.title('Sentiment Analysis on ' + '"' + self.keyword + '"' + ' with ' + str(self.tweetNum) + ' Tweets')

        plt.axis('equal')
        plt.tight_layout()

        plt.show()
```

In [53]:
```
if __name__ == '__main__':

    global result
    result = []

    global tokenized
    tokenized = []

    # number of tweet to retrive
    tweetNum = 1000

    # file to store tweets
    csvFile = 'test1.csv'

    # define keyword(s) to search on
    keyword = 'Huawei'

    # define location with 4 values (bound box)
    geoCode =  '-118.6682, 33.7037, -118.1552, 34.3368,-122.75,36.8,-121.75,37.8,-122.0047, 37.3231, -121.9298, 37.4190,-122.202
7, 37.2888, -122.0891, 37.4658'
          #'-74.2589, 40.4774, -73.7004, 40.9176,-78.9125, 42.8260, -78.7952, 42.9664,-76.2045, 42.9844, -76.0741, 43.0861'

    Tweet_Crawler(keyword, geoCode, tweetNum, csvFile)

    print("Evaluation on " + '"' + keyword + '"' + " based on " + str(tweetNum) + " tweets:")

    # sentiment analysis
    sa = SentimentAnalysis(keyword, geoCode, tweetNum, csvFile)
    sa.Analysis()
```