Andrew Shekhar Breckenridge

Novi High School

February 2014

Comparing Apples and Oranges – An Investigation Into Pointwise Mutual Information As A Measure of Association

Word Count: 2686

Introduction

"It's like comparing apples to oranges."

The ancient adage. Linguistically an idiom; defined as *to examine the similarities of things that are completely different*. In this paper, ignoring the obvious and apparent signs of uselessness in attempting a comparison of the two, I'm going to take a crack at it. My approach will be to interpret the computational probability of the chosen word's (in this case 'apple' or 'orange') linguistic importance.; doing this by by analyzing a 'corpus' of twitter data through a test: a measure of linguistic association called *Pointwise Mutual Information* (PMI).

A corpus is a collection of written material in machine-readable form, in my case Tweets. I chose twitter as my linguistic database because firstly tweets are readily available, they give a good indication of the usage of a word in modern conversation/culture, and I thought a program that connects to twitter and carries out live analysis would be pretty cool.

I found an online record of some million tweets from last year, organized into 364 '.txt' files, each with about 2800 tweets, totaling to a sample of about 1,019,200 tweets or *distinct random sources* to be my source material.

One of the reasons I chose this - an analysis of linguistics on twitter - as a topic to investigate was the extensibility of the research. The tools designed and used in the process of gathering and interpreting this data can be extended to be used in any number of other similar projects. For example, you can just change the token word (apple or orange) in one of the source code files and you have a program that can recreate my experiment for any word. Seeing the difference between other similar words like geek and nerd or the usage of synonyms, what someone means when the say 'yea' instead of 'yes' and vice versa. Objectively, making this a correlation test for the differing uses of words.

Theory

Pointwise Mutual Information is commonly used in informational retrieval literature to measure the concurrence of words and phrases in text, it's a statistic for measuring how much company two words tend to keep. 'Mutual Information' (MI) is the measure of information overlap between two random variables. In this case 'Pointwise' because it treats each source – each tweet – as a unique datapoint to learn from.[1]

The Mutual Information between random variables *X* and *Y*, whose values have marginal probabilities (i.e. they are a subset of all possible values) *p(x)* and *p(y)*, and joint probabilities *p(x,y)*, is as defined as:

$$PMI(X;Y) \equiv \ln \frac{p(x,y)}{p(x)p(y)} \quad (1)$$

$$ln \frac{p(x,y)}{p(x)p(y)} = ln \frac{p(x|y)}{p(x)} = ln \frac{p(y|x)}{p(y)} \quad (2)$$

Pointwise Mutual Information is a measure of how much the actual probability of the event *p(x,y)* differs from what we would expect it be on the basis of individual events and the assumption of independence *p(x)p(y)*.[2] Using the conditional probabilities of the occurrence of a word in a large number of tweets, it is possible to extract the association between the two words.

To get at this information, I first acquired a corpus, or database, of random tweets spanning a year. Then I wrote a program in *Python*[3], a popular scripting language that calculates both the individual probabilities *p(x)* and *p(y)* and the joint probability *p(x,y)*. In the python algorithm I wrote, I decided to use the simplest version of the PMI function (See Equation (1)). I could have used the

---

[1] Bouma, Gerlof. "Normalized (pointwise) mutual information in collocation extraction." Proceedings of GSCL (2009): 31-40.

[2] Ibid.

[3] "Python Programming Language – Official Website." Python Programming Language – Official Website. N.p., n.d. Web. 14 Feb. 2014. <http://www.python.org/>.

more sophisticated, arguably more efficient '*y* given *x*' formula from equation (3), but programming that into the algorithm would have taken more time and since the processing time saved would be marginal, I decided against it. The algorithm I wrote to calculate PMI is available on both my GitHub page and is included in the Appendix of this paper.

A final point to bring up before getting into the method of my investigation is a twitter idiosyncrasy – a tradition if you will – the hashtag. The # "is [a symbol] used to mark keywords or topics in a Tweet. It was created organically by Twitter users as a way to categorize messages."[4] In my algorithm, I decided to let hash-tagged (#) words in a tweet be counted separately from it's non-hash-tagged equivalent, for example, letting the score of '#shiny' and 'shiny' be counted separately, treating them as distinct tokens.

Method

Accessing Twitter Data

Initial Idea

To get the twitter data, a list of raw tweets, I was hoping to use the *Search* and *Streaming API's* from Twitter[5]. They are both freely provided by Twitter, the *Streaming API* giving about 10% of the tweets being posted live and the search returning an indexed record of the queried term. I looked up the API documentation, created a developer key, received the OAuth client and secret key (following the instructions on the documentation). I ran the cURL command they told me to run, but my Terminal was blank. Nothing was being returned.

It turns out Twitter updated their terms of privacy – like many social networks do – they no longer allow dumps of random tweets, saying that it is an invasion of the the privacy of their user

[4] "Using Hashtags on Twitter." Twitter Help Center. N.p., n.d. Web. 16 Feb. 2014. <https://support.twitter.com/articles/49309-using-hashtags-on-twitter>.

[5] "Twitter API Documentation." Twitter Developers. N.p., n.d. Web. 14 Feb. 2014. <https://dev.twitter.com/docs>.

base. Big win for privacy, but my plan was now dead in the water. I had to find a different way to get at a database of tweets.

Plan B

Without access to twitter's *Streaming* API, I started looking online for directories of tweets, zipped archives of millions of tweets that I was hoping *somebody* saved. My search turned up two sources:

I. The U.S. Department of Commerce's Information Technology Laboratory, which held TREC (Text REtrieval Conference conferences, "to encourage research in information retrieval from large text collections."[6] On their website, they had a microblog track, where they had a corpus of about 650 megabytes of tweets from 2011 to run text analysis on. To access it, they said you needed to fax a signed form to request the information. It looked as though getting the information would take a few weeks, so I kept looking

II. *Illocution Incorporated*, which is a company that provides free document review. They keep a corpus of social media: "2012 Twitter SRS, English, 1,000,000 Tweets (46MB)." It was free to download so I accessed it.

Now that I had tweets to run analysis on, 364 return separated files full (a return separated file is a document that has a data entry followed by a line break), I could start writing my algorithm.

---

[6] "Text REtrieval Conference (TREC)." Text REtrieval Conference (TREC) Home Page. N.p., n.d. Web. 14 Feb. 2014. <http://trec.nist.gov/>.

Parsing The Tweets

Parsing is a kind of computer-based syntactic analysis on a string of characters according to the rules of a formal grammar to find some meaning or significance. So for me, it meant turning my return separated tweets into a list of words and their respective PMI scores with 'apple' and with 'orange'.

GATE

I found an open source software package called *GATE*[7] (General Architecture for Text Engineering). It had a core plugin called *TermRaider*[8] with *PMI Bank* as one of it's features. Seeing it, I was relieved. I'd been going into this investigation thinking I'd have to write an algorithm to get run a PMI test but now, with *GATE*, I wouldn't (or so I thought).

I downloaded the package and gave it my tweets. Turns out, *TermRaider*, the plugin I was hoping to use, was extremely volatile. The program wouldn't even accept my tweets as input.

GATE was not an option. I would have to write a program instead.

python

I had a basic grasp of what needed to be done to conduct a PMI test, and since *GATE*, my easy sans-coding option failed, I would be writing my own algorithm. Even though writing it would take a few hours, I was happy that I would be able to customize every aspect of how the data would be calculated and displayed.

I chose python as a language to write in because of its flexibility; there are many third party libraries written by regular people who saw a need for an additional tool that extend the language's

---

[7] "GATE Index." GATE.ac.uk - Index.html. N.p., n.d. Web. 14 Feb. 2014. <http://gate.ac.uk/>.

[8] "GATE Plugins - TermRaider." Gate.ac.uk. N.p., n.d. Web. 14 Feb. 2014. <http://gate.ac.uk/gate/doc/plugins.html#TermRaider>.

usability. Furthermore, python is an extremely easy language to code in, it gets more done with less lines of code.

Writing the code took longer than expected, about eight hours in all. I ended up using the *NLTK* (Natural Language ToolKit) library[9], a open source framework I found, to turn my directory of tweets into a corpora (another word for corpus), and then count the occurrences of my tokens, 'apple' and 'orange'. With a corpora, I then used standard python to count the number of occurrences of every word in the corpora with my tokens, find the 'background' probability of the word, *p(y),* and then calculate the respective PMIs.

The code I wrote is included as a supplement to this paper in the appendix and is also available on GitHub[10] for easy reading and code reuse.

Making Sense Of The Data

Organizing

Now that I had two *comma separated value* files (CSV's are text files formatted with commas. i.e. there is a word and then a PMI score separated by a comma) filled with words and their respective PMI scores, I could start to extract some meaning. After sorting by the highest PMI score I noticed that I was getting a lot of words with the same score. In the orange CSV file, 1245 words had a PMI score of `12.51`, mathematically:

$$\ln\left(\frac{1}{p(of\ orange)\cdot p(of\ word)}\right)=12.51 \qquad (3)$$

[9] "Natural Language Toolkit." Natural Language Toolkit — NLTK 3.0 Documentation. N.p., n.d. Web. 14 Feb. 2014. <http://nltk.org/>.

[10] "Andrew Breckenridge's PMICalc Repository." GitHub. N.p., n.d. Web. 14 Feb. 2014. <https://github.com/AndrewSB/TwitterPMI>.

I was worried that my algorithm wasn't working properly. After going through the data for a while, I realized that the problem was a lot of these words were in the same kind of situation – they showed up only once in the corpus and the one time they did, it was with the token word. All of those words will obviously have the same score because all of the terms in the PMI function are identical. For example, the word '*#1984*', the fourth entry in *Fig. A*, shows up just once in the entire corpus and the one time it does is in a tweet **with** the word orange. Thats why '*#1984*', along with all these other words, have a score of `12.51`.

#template	13.2021740925
horange	13.2021740925
orange..	12.9144920201
#1984	12.509026912
#1rtごとにどうでもいい個人	12.509026912
#2012worldseries	12.509026912
#4lions	12.509026912
#6/10	12.509026912
#aclockworkorange	12.509026912
#artstudentwoes	12.509026912
#asondeguerrajlg	12.509026912
#aubonpain--	12.509026912
#badsanta	12.509026912
#bclions	12.509026912
#blackandorange	12.509026912
#blackshirt	12.509026912
#bleedorange	12.509026912
#blueandredalltheway	12.509026912
#bossorange	12.509026912

Fig. A

So the problem was that my corpus was too small. Evidently, my source of 1,034,852 data points was not large enough to get totally accurate results. As consolation, 1245 (though it sounds like a **lot** of my data, almost all of it) isn't actually that much, 1245 values only make up 8.9% of the words I had, so it's not as bad as I first thought.

Plotting

Plotting the data was an unexpected difficulty. I was hoping to just run my values into excel and have it generate for me a beautiful scatterplot. Boy was I wrong.

I knew I wanted a 2 axis plot with values from 0 to 10, just a quick way for you to tell if a specific word had high association with 'apple' or 'orange', but I couldn't find a software package or graph included in excel that would get me *anywhere* close to the results I was hoping for.

This plot was in a way the fruit of my investigation, it would be the only tangible way to see what I found out from my research. So it was of some importance to me.

Again, I had to write a program to get what I wanted done. I learned how to use a programing language for statistical computing called 'R'[11]. It's syntax wasn't impossibly difficult, so I was able to grasp the major concepts pretty quickly and get aon to making a script to accomplish what I wanted. The script itself is included as an Appendix after the python script and is also available in my Github repository[12].

Normalizing PMI

In the process of plotting my data, after I had my script written and I was messing around with the axes and scaling, I noticed that almost all the values I was choosing to plot were all in the upper range – around the range of 6-11 in both the apple and orange axes. This was definitely a problem, my plot was lopsided with most of the values towards the top right and almost none in the 0-5 range on the lower left side of the plot. I realized to get a better distributed plot, I'd have to adjust my PMI values to make it more even. I'd have to normalize.

I first tried a really simple transformation, I 'percentiled' my data. My values ranged from about 3 - 13, so I just divided by the maximum value (13) and multiplied by 10:

$$\frac{PMI}{13} \cdot 10 = normalized\ PMI \qquad (4)$$

So it was basically just the same PMI in a percentileized form. I was disappointed that it didn't really change anything. My clump that was between 6 and 11 had just moved with the axes, now in the 5-8 range. I had only succeeded in moving the problem.

---

[11] "The R Project for Statistical Computing." The R Project for Statistical Computing. N.p., n.d. Web. 13 Feb. 2014. <http://www.r-project.org/>.

[12] "Andrew Breckenridge's Twitter Repository." GitHub. N.p., n.d. Web. 14 Feb. 2014. <https://github.com/AndrewSB/TwitterPMI>.

I needed to do something more complex to actually *normalize* the scores. I came across a paper written by Bourma Gerlof titled "Normalized (Pointwise) Mutual Information in Collocation Extraction."[13] Though I wasn't trying to extract sentiment from my PMI values like he was – I wasn't interested in 'collocation extraction' – the *normalizing* part of his research, I hoped, would work the same for me. So I implemented his changes into my python scripts to get a new CSV of values. The difference between the PMI function I was previously using and the new Gerlof normalized function was that he added a few terms. Here is my old PMI function, reproduced:

$$PMI(X;Y) \equiv \ln \frac{p(x,y)}{p(x)p(y)} \tag{1}$$

And this next equation is Gerlof's, which he found empirically to normalize PMI using the "joint entropy of the mutual information"[14]:

$$I(X;Y) = \frac{p(x,y) \cdot \ln \frac{p(x,y)}{p(x) \cdot p(y)}}{p(x,y) \cdot \ln p(x,y)} \tag{5}$$

Substituting (Equation 1) into (Equation 5) you get:

$$I(X;Y) = \frac{p(x,y) \cdot PMI(X;Y)}{p(x,y) \cdot \ln p(x,y)} \tag{6}$$

Which is equivalent to:

$$I(X;Y) = PMI(X;Y) \cdot \frac{p(x,y)}{p(x,y) \cdot \ln p(x,y)} \tag{7}$$

Simplifying you get:

$$I(X;Y) = PMI(X;Y) \cdot \frac{1}{(\ln p(x,y))} \tag{8}$$

---

[13] Bouma, Gerlof. "Normalized (pointwise) mutual information in collocation extraction." Proceedings of GSCL (2009): 31-40.

[14] Ibid.

So Gerlof's function uses entropy in the form of $\ln p(x,y)$ to normalize the PMI. I implemented this change in my algorithm and it worked, my values were now pretty evenly distributed. Now that I had normalized PMI data, I could plot it properly.

Results

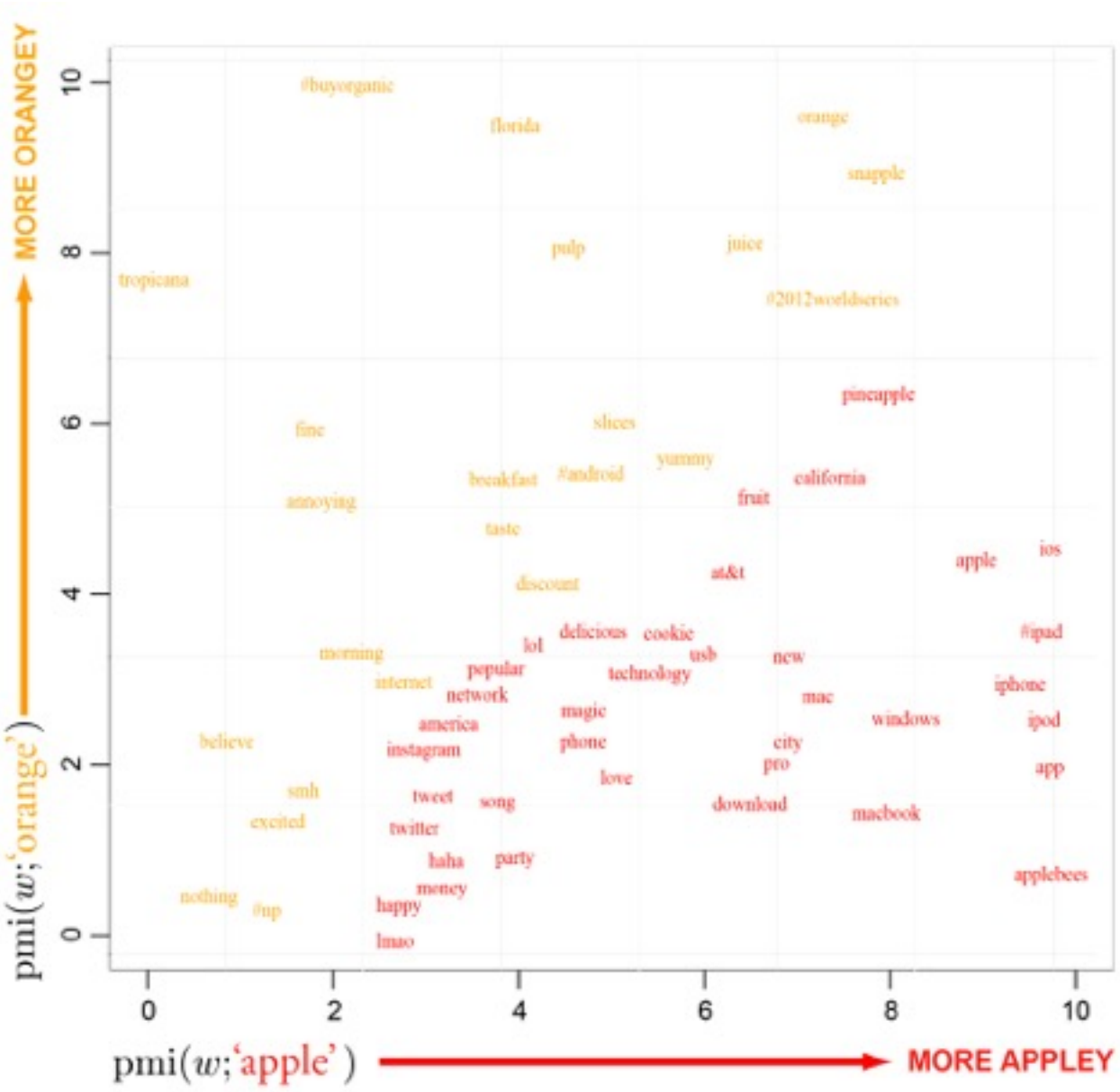

Fig. B

So this is what I ended up with. A plot with words I thought were significant on a scale from zero to ten with the y *axis* value the score of a word with orange and the x *axis* value with apple.

There is, of course, some significant bias in my result; I manually chose which words to include in the graph on the basis of them 'sounding significant'. To try and balance that out, I did

include a few random words (using a random number generator from 1 to 13865 to choose another entry) as well, to try and reduce that bias. The entire spreadsheet is also available on my Github page[15] for further examination.

The results are more or less what I expected.

- Words that are to do with both apple and orange - 'fruit', delicious', 'slices' – are towards the middle $y=x$ line of the graph, they are equally associated with apple and orange
- Words that are significant only to apple, so #ipad', 'app', or 'macbook' are in the lower right quadrant. High 'apple' PMI score, low 'orange'
- Words that are significant only to orange: 'tropicana', '#buyorganic', florida', and 'pulp' are the exact opposite, upper left quadrant. High 'orange' PMI, low 'apple'

So what did I learn by investigating this mathematically linguistic idea? Well first of all, big data is *annoying*. The first time I tested my algorithm, it said it would take **28 hours** to finish giving me the PMI CSV chart. Efficiency is a prerequisite to make an application scalability and deceptively hard to code for.

I had my eyes opened in terms of incorporating math with computer science. I could use something I enjoyed playing with to do research; consequently discovering the validity of Twitter in prerequisite-scholarly (probably scholarly as well) research.

Project specific, I liked my final result but am disappointed with the accuracy. Some of the PMI values came out as 'invalid', which is a result of division by zero. Division by zero in the PMI function means either $p(x)$ or $p(y)$ was 0, which obviously is an error since $p(x)$ can definitely not be 0 and the very existence of value Y suggests that $p(y)$ should have a probability of showing up in the corpus.

---

[15] "Andrew Breckenridge's Twitter Repository." *GitHub*. N.p., n.d. Web. 14 Feb. 2014. <https://github.com/AndrewSB/TwitterPMI>.

As a closing point, this paper really felt like a precursor to the research I'll be doing in some of my college classes, it was fun to do.

Bibliography

Works Cited

"Andrew Breckenridge's Twitter Repository." *GitHub*. N.p., n.d. Web. 14 Feb. 2014. <https://github.com/AndrewSB/TwitterPMI>.

"Using Hashtags on Twitter." Twitter Help Center. N.p., n.d. Web. 16 Feb. 2014. <https://support.twitter.com/articles/49309-using-hashtags-on-twitter>.

"GATE Index." *GATE.ac.uk - Index.html*. N.p., n.d. Web. 14 Feb. 2014. <http://gate.ac.uk/>.

"GATE Plugins - TermRaider." *Gate.ac.uk*. N.p., n.d. Web. 14 Feb. 2014. <http://gate.ac.uk/gate/doc/plugins.html#TermRaider>.

"Natural Language Toolkit." *Natural Language Toolkit — NLTK 3.0 Documentation*. N.p., n.d. Web. 14 Feb. 2014. <http://nltk.org/>.

"Python Programming Language – Official Website." *Python Programming Language – Official Website*. N.p., n.d. Web. 14 Feb. 2014. <http://www.python.org/>.

"The R Project for Statistical Computing." *The R Project for Statistical Computing*. N.p., n.d. Web. 13 Feb. 2014. <http://www.r-project.org/>.

Settles, Burr. "Slackpropagation." *Slackpropagation*. N.p., n.d. Web. 14 Feb. 2014. <http://slackprop.wordpress.com/2013/06/03/on-geek-versus-nerd/>.

"The Streaming APIs." *Twitter Developers*. N.p., n.d. Web. 14 Feb. 2014. <https://dev.twitter.com/docs/streaming-apis>.

"Text REtrieval Conference (TREC)." *Text REtrieval Conference (TREC) Home Page*. N.p., n.d. Web. 14 Feb. 2014. <http://trec.nist.gov/>.

"Twiter API Documentation." *Twitter Developers*. N.p., n.d. Web. 14 Feb. 2014. <https://dev.twitter.com/docs>.

Appendix

Python Scripts

PMICalc

```
from __future__ import division
__author__ = 'asb'
import driver, main, csv, math

PMITerm = driver.PMITerm

#panswer[0]  = string
#panswer[1]  = p(x,y)
#panswer[2]  = p(y)
#pofx       = p(x)
#PMI(x;y)   = p(x,y)/(p(x)*p(y))

#Output: string, PMI(x,y)
answer = driver.answer

numberOfTweets = len(main.allTweets)
numberOfWords = len(driver.totalTweets)


panswer = [] #panswer becomes (x,x,x) with string, p(x,y), p(y)
for element in answer:
    panswer.append((element[0], (element[1]/numberOfTweets),
(element[2]/numberOfWords)))

pofx = driver.totalTweets.count(PMITerm)/numberOfWords


PMIs = []
for element in panswer:
    if (pofx != 0 and element[2] != 0):
        top = -element[1]*math.log((element[1])/(pofx*element[2]))
        bottom = element[1]*math.log(element[1])
        PMI = top/bottom
        #PMI = (element[1]/((pofx*element[2])))
        #PMI = math.log(PMI)
    else:
        PMI = 'invalid'
    PMIs.append((element[0], PMI))


f = open('output.csv', 'wb')
```

```
for element in PMIs:
    f.write(str(element[0]))
    f.write(',')
    f.write(str(element[1]))
    f.write("\n")
f.close()
```

Main

```
__author__ = 'asb'
#Thursday & Friday 2014-01-16,17
#Andrew Shekhar Breckenridge
#asbreckenridge@me.com  @Andrew_Breck



import nltk, re, itertools
from nltk.corpus import PlaintextCorpusReader
import collections


def GetTweets():
    corpusdir = 'DB/'

    newCorpus = PlaintextCorpusReader(corpusdir, '.*\.txt$') #Regex
allows you to ignore .DS_Store

    pattern = '\r\n' #Regex accepts \r\n as the next line encoding in
each 'tweet' in the database
    tweets = nltk.regexp_tokenize(newCorpus.raw(), pattern, gaps=True)
#iterate through list, creating 'tweets'
    tweets = [x.lower() for x in tweets] #make all strings lowercase
to make matching easier
    return tweets

allTweets = GetTweets() #GLOBAL

def GetRelevantTweets(PMITerm):
    matching = []
    for s in allTweets:
        if PMITerm in s:
            matching.append(s)

    backgroundcorp = []
    for element in matching:
        current = re.split(r'\s', element)
        backgroundcorp.append(current)
```

```
    return backgroundcorp


def GetSortedRelevantTweets(PMITerm):
    corpus = []
    backgroundcorp = GetRelevantTweets(PMITerm)

    for list in backgroundcorp:
        for word in list:
            if word != 'orange' and word != 'oranges':
                if (word.find('"') != -1):
                    word = word[0:word.index('"')] +
word[word.index('"')+1:]
                if (word.find("'") != -1):
                    word = word[0:word.index("'")] +
word[word.index("'")+1:]
                corpus.append(word)

    return sorted(corpus)


def CountInstancesGlobal():
    line =[]
    for sentances in GetTweets():
        words = sentances.split()
        line.append(words)
    line = [item for item in list(itertools.chain(*line)) if (item)]

    counted = collections.Counter(line)
    iterater = (sorted(counted.most_common()))

    return iterater




def CountInstances(PMITerm):
    list = []
    for words in GetSortedRelevantTweets(PMITerm):
        list.append(words)

    counted = collections.Counter(list)

    iterater = (sorted(counted.most_common()))
    return iterater
```

```
def WriteToExcel():
    f = open('wordbank.csv','w')
    list = CountInstancesGlobal()
    for twoplet in list:
        f.write(str(twoplet[0]))
        f.write(',')
        f.write(str(twoplet[1]))
        f.write('\n')
    f.close()


def WriteEAndToExcel(PMITerms):
    f = open('output.csv', 'w')
    for word in CountInstances(PMITerms):
        f.write(word[0])
        f.write(',')

        count = -1
        query = word[0]
        list = CountInstancesGlobal()
        for i in list:
            count = count + 1
            if i[0] == query:
                break
        f.write(list[count][0])
        f.write(',')
        f.write(str(list[count][1]))
        f.write(',')

        f.write(str(word[1]))
        f.write('\n')
    f.close()
```

Driver

```
__author__ = 'asb'

import main, collections, itertools

PMITerm = 'apple'
listOfBGWords = main.GetSortedRelevantTweets(PMITerm)


lister = [] #this code block calcs tuplesIT variable
for words in listOfBGWords:
    lister.append(words)
counted = collections.Counter(lister)
```

```
sortedCounter = counted.most_common()
tuplesIT = sorted(sortedCounter) #tuples form ('word', 'instances') of
all the words in the tweets - "tuples In Tweets"


lister = [] #this code block calcs
cleanList = []
for sentences in main.allTweets:
    words = sentences.split()
    lister.append(words)

cleanList = list(itertools.chain(*lister))

totalTweets = cleanList #GLOBAL

counted = collections.Counter(cleanList)
tuplesG = sorted(counted.most_common())

dict_tuplesG = dict(tuplesG) #codeblock concatenates the two tuples
into one of the form ('word', instances in relevant tweets, instances
in corpus)
answer = [(string, count, dict_tuplesG.get(string, 0)) for string,
count in tuplesIT]
```

Runall

```
__author__ = 'asb'

#Runs the entire algorithm.
#Enter the word you want to test for in driver.py
#The corpus location and other basic config is in main

#Project is structured like:
#           |---FormatOutput
#runall ----|
#           |---PMICalc ---> driver -----> main

import PMICalc
import FormatOutput
```

FormatOutput

```
__author__ = 'asb'


def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

numberOfLines = int(file_len('output.csv'))


r = open('output.csv', 'r')
w = open('formattedOutput.csv', 'w')


for i in range(numberOfLines):
    line = r.readline()
    numberOfCommas = line.count(',')
    while numberOfCommas > 1:
        indexOfComma = line.index(',')
        print line
        tempLine = line[0:indexOfComma] + ' ' + line[indexOfComma+1:]
        print tempLine
        line = tempLine
        numberOfCommas = line.count(',')
    w.write(line)
```

R Console

Log

```
//prerequisites fread, data.table, ggplot2

>library(data.table)

>library(ggplot2)

>df = fread(final.csv)

>r=c(range(df$apple.pmi),range(df$orange.pmi))

>ggplot(df,aes(apple.pmi,orange.pmi,color=factor(apple.pmi>orange.pmi)
,label=words)) + geom_text(size=5, family="Times New Roman") +
```

```
theme(panel.grid=element_blank(),legend.position="None",
panel.background = element_rect(fill='light grey')) +
scale_color_manual(values=c("#FFA500","#FF0800")) +
scale_x_continuous(limit=c(min(r),max(r))) +
scale_y_continuous(limit=c(min(r),max(r))) + theme_minimal()
```