**Downloading of Data:**

To get large publically available Twitter datasets, we use Twitter API.

# Obtaining access tokens

In order to make authorized calls to Twitter’s APIs, your application must first obtain an OAuth **access token** on behalf of a Twitter user or you could issue Application-only authenticated requests when user context is not required. The way you will obtain such tokens will depend on your use case.[4]

The Twitter API has two different flavors: RESTful and Streaming.

# REST APIs

The [REST APIs](https://dev.twitter.com/rest/public) provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using [OAuth](https://dev.twitter.com/oauth); responses are available in JSON. The RESTful API is useful for getting things like lists of followers and those who follow a particular user, and is what most Twitter clients are built off of.[1]

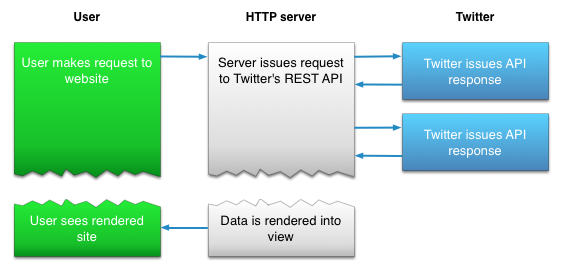
# The Streaming APIs

The Streaming APIs give developers low latency access to Twitter’s global stream of Tweet data. A proper implementation of a streaming client will be pushed messages indicating Tweets and other events have occurred, without any of the overhead associated with polling a REST endpoint.Twitter offers several streaming endpoints, each customized to certain use cases.The Streaming API works by making a request for a specific type of data — filtered by keyword, user, geographic area, or a random sample — and then keeping the connection open as long as there are no errors in the connection.[2]

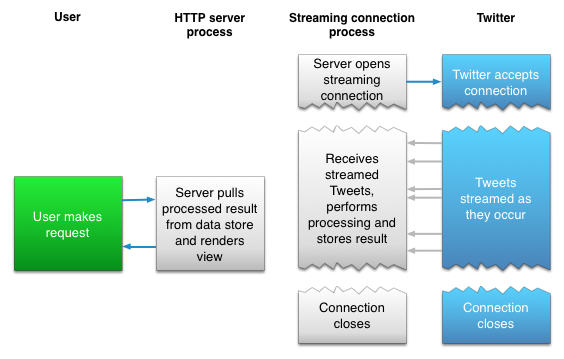
|  |  |
| --- | --- |
| [Public streams](https://dev.twitter.com/streaming/public) | Streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining. |
| [User streams](https://dev.twitter.com/streaming/userstreams) | Single-user streams, containing roughly all of the data corresponding with a single user’s view of Twitter. |
| [Site streams](https://dev.twitter.com/streaming/sitestreams) | The multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users. |

## Differences between Streaming and REST

Connecting to the streaming API requires keeping a persistent HTTP connection open. In many cases this involves thinking about your application differently than if you were interacting with the REST API. For an example, consider a web application which accepts user requests, makes one or more requests to Twitter’s API, then formats and prints the result to the user, as a response to the user’s initial request:[2]



An app which connects to the Streaming APIs will not be able to establish a connection in response to a user request, as shown in the above example. Instead, the code for maintaining the Streaming connection is typically run in a process separate from the process which handles HTTP requests:



The streaming process gets the input Tweets and performs any parsing, filtering, and/or aggregation needed before storing the result to a data store. The HTTP handling process queries the data store for results in response to user requests. While this model is more complex than the first example, the benefits from having a realtime stream of Tweet data make the integration worthwhile for many types of apps. To access the twitter API we have used tweepy python package.[3]

**Proposed Scoring Modules**

We have proposed several scoring methods/formulas to calculate the average emotion value of tweets of a handle (username) or hashtag:

1. Emotion value = sum of adjective values + sum of adverb and verb values

(no. of adjectives \* 5) + no. of adverb and verb

1. Multiply the value of adverb/verb with the upcoming adjective .If two verb /adverb are next to each other, simply multiply them. And then all these products are added. Further divided by 5\*number of adjectives encountered.
2. If the value of adverb/verb is less than 0 i.e, negative, then for the upcoming adjective, subtract its value from 5 instead of multiplying. And if value of verb/adverb is positive and >= 0.5 then multiply it with the upcoming adjective else multiply 0.5 with upcoming adjective. Later these products are added and the sum is divided by 5\*number of adjectives encountered.
3. Since the emotions are independent of each other, we should treat them independently and use a different formula for each emotion. We have used normalising values of different emotions. These predefined values are used to normalize the emotion values. In this method an assumption is taken that if an emotion is strong it’s respective value is > 3 (out of a scale of 5).

**Scoring Module-1:**

* This algorithm is based on the idea that every emotion is distinct from one another and thus the formula used to calculate the required emotion value is different from one another. The general method used is:

if sum of emotion values of all adjectives/number of adjectives > 3

then emotion value = sum of adjectives/normalizing factor + sum of adverb & verbs

total number of adjectives, adverbs and verbs

else

emotion value = sum of adjectives/normalizing factor - sum of adverb & verbs

total number of adjectives, adverbs and verbs

* The normalizing factor varies for each emotion and hence gives a different formula for each emotion. This normalizing factor has been calculated on a common scale of 0 to 5 using the images of brain scans of prefrontal cortex.
* There is an assumption that words with emotion value (0-5) > 3 for an adjective express stronger feelings. Whereas the other express weaker feelings.

**Scoring Module-2:**

* This algorithm is based on the idea that whenever there is an adverb/verb in front of an adjective there is bound to be an effect on the emotion expressed by the same adjective. It can be either negative or negative. Keeping this in mind we have created method to calculate the emotion values.
* If the value of adverb/verb is less than 0 i.e, negative, then for the upcoming adjective, subtract its value from 5 instead of multiplying . And if value of verb/adverb is positive and >= 0.5 then multiply it with the upcoming adjective else multiply 0.5 with upcoming adjective. Later these products are added and the sum is divided by 5\*number of adjectives encountered.
* And if there are more than 1 adverbs/verbs before an adjective then they are multiplied together.

**Dictionary Expansion Module:**

* The main idea of this module is to create a dictionary apart from the original dictionary being used.
* This module will be run after the scoring module for a hashtag or user-handle has been run.
* The emotion values from the scoring modules will be fed into this module and those values will be assigned to words (adjective/verb/adverb) that haven’t been identified by the original dictionary. Now this module is separate from the model and uses emotion values of every iteration of the model to expand it’s dictionary and update already used words.
* This module is designed to input emotion values of a hashtag or a user-handle. But it can be modified to process the tweets one by one.

References:

1) https://dev.twitter.com/rest/public

2) https://dev.twitter.com/streaming/overview

3) <https://media.readthedocs.org/pdf/tweepy/latest/tweepy.pdf>

4) <https://dev.twitter.com/oauth>

**Source Code**

**The Twitter Application credentials**

consumer\_key = "RGVO3CTujE60TW5IQy1JwmyxF"

consumer\_secret = "ziWzApZCAqlwOt3xK3L0B02VjEsDFZg4Fniy76TsTLKgtnjqlG"

access\_key = "1587880604-1tjWpdETzVE4fPALCGeNs6O2oHi4y8ShwIsDQSl"

access\_secret = "wJHBahm2y3KnTXuuj2JX18GAolaHVMnLKZ7Ygc0LxnQMH"

auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_key, access\_secret)

**Function to download tweets corresponding to a hash tag.**

hashtag = name

api = tweepy.API(auth)

print("FYI , the following trending topics are available")

trends1 = api.trends\_place(1)

trends = set([trend['name'] for trend in trends1[0]['trends']])

print (trends)

print("Tweets downloading has started !!")

'''

cricTweet = tweepy.Cursor(api.search, q=hashtag).items(179)

for tweet in cricTweet:

print (tweet.text)

with open('hashtag\_tweets','w') as f:

for tweet in cricTweet:

f.write(tweet.text)

'''

searchQuery = '#someHashtag' # this is what we're searching for

maxTweets = 5000 # Some arbitrary large number

tweetsPerQry = 100 # this is the max the API permits

fName = 'tweets.txt' # We'll store the tweets in a text file.

# If results from a specific ID onwards are reqd, set since\_id to that ID.

# else default to no lower limit, go as far back as API allows

sinceId = None

# If results only below a specific ID are, set max\_id to that ID.

# else default to no upper limit, start from the most recent tweet matching the search query.

max\_id = -1

tweetCount = 0

print("Downloading max {0} tweets".format(maxTweets))

with open(fName, 'w') as f:

while tweetCount < maxTweets:

try:

if (max\_id <= 0):

if (not sinceId):

new\_tweets = api.search(q=hashtag, count=tweetsPerQry)

else:

new\_tweets = api.search(q=hashtag, count=tweetsPerQry, since\_id=sinceId)

else:

if (not sinceId):

new\_tweets = api.search(q=hashtag, count=tweetsPerQry, max\_id=str(max\_id - 1))

else:

new\_tweets = api.search(q=hashtag, count=tweetsPerQry, max\_id=str(max\_id - 1), since\_id=sinceId)

if not new\_tweets:

print("No more tweets found")

break

list\_tweets = []

for tweet in new\_tweets:

list\_tweets.append(tweet.text)

tweetCount += len(new\_tweets)

print("Downloaded {0} tweets".format(tweetCount))

max\_id = new\_tweets[-1].id

except tweepy.TweepError as e:

# Just exit if any error

print("some error : " + str(e))

break

z=open(hashtag+'.txt','w')

for tweet in list\_tweets:

tweet=tweet.encode('ascii','ignore')

z.write(tweet.decode())

z.write("\n")

z.close()

return list\_tweets

#print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))

**Function to download tweets corresponding to an username.**

def get\_all\_tweets(username):

alltweets = []

user\_timeline = twitter.get\_user\_timeline(screen\_name=username,count=200,include\_rts=1)

alltweets.extend(user\_timeline)

oldest = alltweets[-1]['id'] - 1

#keep grabbing tweets until there are no tweets left to grab

while len(user\_timeline) > 0:

user\_timeline = twitter.get\_user\_timeline(screen\_name=username,count=200,include\_rts=1,max\_id=oldest)

alltweets.extend(user\_timeline)

oldest = alltweets[-1]['id']

tweets = []

for tweet in alltweets:

tweets.append(tweet['text'])

return tweets

**Data cleaning Functions:**

def isEnglish(s):

try:

s.encode('ascii')

except UnicodeEncodeError:

return False

else:

return True

#The following function removes the part of the string that contains the substring eg. if

#substring = 'http' , then http://www.google.com is removed, that means, remove until a space is found

def rem\_substring(tweets,substring):

m=0;

#print(len(tweets))

for i in tweets:

while i.find(substring)!=-1:

k=i.find(substring)

d=i.find(' ',k,len(i))

if d!=-1: #substring is present somwhere in the middle(not the end of the string)

i=i[:k]+i[d:]

else: #special case when the substring is present at the end, we needn't append the

i=i[:k] #substring after the junk string to our result

tweets[m]=i #store the result in tweets "list"

#print(i)

m += 1

return tweets

#The following function removes the non English tweets .Makes use of the above written isEnglish Function

def removeNonEnglish(tweets):

result=[]

for i in tweets:

if isEnglish(i):

result.append(i)

return result

#the following function converts all the text to the lower case

def lower\_case(tweets):

result=[]

for i in tweets:

result.append(i.lower())

return result

def rem\_punctuation(tweets,punct):

#print(len(tweets))

m=0

for i in tweets:

i=i.replace(punct,'')

tweets[m]=i

m=m+1

return tweets

**Scoring Module 1:**

def score(name):

filename = 'data\_emotions\_words\_list.csv'

count\_adj=0

count=0

#ans=[]

ans.append(0)

ans.append(0)

ans.append(0)

ans.append(0)

ans.append(0)

special\_word=None

special\_score=None

#r=open(sys.argv[1],'r')

for line in name:

line=line.lower()

line=line.replace("."," ")

line=line.split(" ")

list\_words=line

#print(list\_words[0])

for word in list\_words:

with open(filename,encoding="ISO-8859-1",newline='') as f:

reader = csv.reader(f)

for row in reader:

my\_list=row

word=word.lower()

if (word == my\_list[0] and len(word)>=1):

ans\_list=[]

count\_adj+=1

#print(count)

ans\_list.append(word)

ans\_list.append(my\_list[1])

ans\_list.append(my\_list[3])

ans\_list.append(my\_list[5])

ans\_list.append(my\_list[7])

ans\_list.append(my\_list[9])

if special\_word is None:

ans[0]+=float(my\_list[1])

ans[1]+=float(my\_list[3])

ans[2]+=float(my\_list[5])

ans[3]+=float(my\_list[7])

ans[4]+=float(my\_list[9])

else:

print(special\_word,word)

print(special\_score)

if special\_score >=0:

ans[0]+=float(my\_list[1])\*max(0.5,float(special\_score))

ans[1]+=float(my\_list[3])\*max(0.5,float(special\_score))

ans[2]+=float(my\_list[5])\*max(0.5,float(special\_score))

ans[3]+=float(my\_list[7])\*max(0.5,float(special\_score))

ans[4]+=float(my\_list[9])\*max(0.5,float(special\_score))

else:

ans[0]+=5-float(my\_list[1])

ans[1]+=5-float(my\_list[3])

ans[2]+=5-float(my\_list[5])

ans[3]+=5-float(my\_list[7])

ans[4]+=5-float(my\_list[9])

special\_word=None

#final\_list.append(ans\_list)

#print(ans\_list)

break

with open('adverb.csv',encoding="ISO-8859-1",newline='') as a:

reader=csv.reader(a)

for row in reader:

list\_adverb=row

#print(list\_adverb)

word=word.lower()

if (word==list\_adverb[0]):

count+=1

ans\_list=[]

ans\_list.append(word)

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

#ans[0]+=float(list\_adverb[1])

#ans[1]+=float(list\_adverb[1])

#ans[2]+=float(list\_adverb[1])

#ans[3]+=float(list\_adverb[1])

#ans[4]+=float(list\_adverb[1])

if special\_word is None:

special\_word=word

special\_score=float(list\_adverb[1])

else:

special\_word=word

special\_score=float(special\_score)\*float(list\_adverb[1])

#print(ans\_list)

break

with open('verb.csv',encoding="ISO-8859-1",newline='') as v:

reader=csv.reader(v)

for row in reader:

list\_verb=row

word=word.lower()

#print(list\_adverb)

if (word==list\_verb[0]):

count+=1

ans\_list=[]

ans\_list.append(word)

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

#ans[0]+=float(list\_verb[1])

#ans[1]+=float(list\_verb[1])

#ans[2]+=float(list\_verb[1])

#ans[3]+=float(list\_verb[1])

#ans[4]+=float(list\_verb[1])

if special\_word is None:

special\_word=word

special\_score=float(list\_verb[1])

else:

special\_word=word

special\_score=float(special\_score)\*float(list\_verb[1])

#print(ans\_list)

break

#print(count)

#print(ans)

for i in range(0,5):

#ans[i]=ans[i]/((5\*count\_adj)+count)

ans[i]=ans[i]/(5\*count\_adj)

print(ans)

return ans

**Scoring Module 2:**

def score(name):

#count\_adj=0

count=0

ans=[]

ans.append(0)

ans.append(0)

ans.append(0)

ans.append(0)

ans.append(0)

#r=open(name,'r')

count\_adj=0

#count\_verb=0

adj\_ans=[]

adj\_ans.append(0)

adj\_ans.append(0)

adj\_ans.append(0)

adj\_ans.append(0)

adj\_ans.append(0)

adv\_ans=[]

adv\_ans.append(0)

adv\_ans.append(0)

adv\_ans.append(0)

adv\_ans.append(0)

adv\_ans.append(0)

#r=open(sys.argv[1],'r')

for line in name:

line=line.lower()

line=line.replace("."," ")

line=line.split(" ")

list\_words=line

# print(list\_words[0])

for word in list\_words:

with open("data\_emotions\_words\_list.csv",encoding="ISO-8859-1",newline='') as f:

reader = csv.reader(f)

for row in reader:

my\_list=row

word=word.lower()

if (word == my\_list[0] and len(word)>=1):

ans\_list=[]

count\_adj+=1

#print(count)

ans\_list.append(word)

ans\_list.append(my\_list[1])

ans\_list.append(my\_list[3])

ans\_list.append(my\_list[5])

ans\_list.append(my\_list[7])

ans\_list.append(my\_list[9])

adj\_ans[0]+=float(my\_list[1])

adj\_ans[1]+=float(my\_list[3])

adj\_ans[2]+=float(my\_list[5])

adj\_ans[3]+=float(my\_list[7])

adj\_ans[4]+=float(my\_list[9])

#final\_list.append(ans\_list)

#print(ans\_list)

break

with open('adverb.csv',encoding="ISO-8859-1",newline='') as a:

reader=csv.reader(a)

for row in reader:

list\_adverb=row

#print(list\_adverb)

word=word.lower()

if (word==list\_adverb[0]):

count+=1

ans\_list=[]

ans\_list.append(word)

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

ans\_list.append(list\_adverb[1])

adv\_ans[0]+=float(list\_adverb[1])

adv\_ans[1]+=float(list\_adverb[1])

adv\_ans[2]+=float(list\_adverb[1])

adv\_ans[3]+=float(list\_adverb[1])

adv\_ans[4]+=float(list\_adverb[1])

#print(ans\_list)

break

with open('verb.csv',encoding="ISO-8859-1",newline='') as v:

reader=csv.reader(v)

for row in reader:

list\_verb=row

word=word.lower()

#print(list\_adverb)

if (word==list\_verb[0]):

count+=1

ans\_list=[]

ans\_list.append(word)

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

ans\_list.append(list\_verb[1])

adv\_ans[0]+=float(list\_verb[1])

adv\_ans[1]+=float(list\_verb[1])

adv\_ans[2]+=float(list\_verb[1])

adv\_ans[3]+=float(list\_verb[1])

adv\_ans[4]+=float(list\_verb[1])

#print(ans\_list)

break

if adj\_ans[0]/count\_adj >3:

ans[0]=((adj\_ans[0]/3.725)+adv\_ans[0])/(count+count\_adj)

else:

ans[0]=((adj\_ans[0]/3.725)-adv\_ans[0])/(count+count\_adj)

if adj\_ans[2]/count\_adj >3:

ans[2]=((adj\_ans[2]/3.4875)+adv\_ans[2])/(count+count\_adj)

else:

ans[2]=((adj\_ans[2]/3.4875)-adv\_ans[2])/(count+count\_adj)

if adj\_ans[4]/count\_adj >3:

ans[4]=((adj\_ans[4]/2.665)+adv\_ans[4])/(count+count\_adj)

else:

ans[4]=((adj\_ans[4]/2.665)-adv\_ans[4])/(count+count\_adj)

if adj\_ans[1]/count\_adj >3:

ans[1]=((adj\_ans[1]/2.5)+adv\_ans[1])/(count+count\_adj)

else:

ans[1]=((adj\_ans[1]/2.5)-adv\_ans[1])/(count+count\_adj)

if adj\_ans[3]/count\_adj >3:

ans[3]=((adj\_ans[3]/2.5)+adv\_ans[3])/(count+count\_adj)

else:

ans[3]=((adj\_ans[3]/2.5)-adv\_ans[3])/(count+count\_adj)

print(ans)

return ans

**POS Tagger Function used to identify the adjectives,verbs,adverbs.**

def POS\_tagger(tweets,username):

x=[]

#for each line in tweets list

for line in tweets:

tokenized=nltk.sent\_tokenize(line)

t=""

#for each sentence in the line

for sent in tokenized:

#tokenize this sentence

text=nltk.word\_tokenize(sent)

k=nltk.pos\_tag(text)

for i in k:

if (i[1][:2]=="VB" or i[1][:2]=="JJ") or i[1][:2]=="RB":

t=t+i[0]+' '

x.append(t)

filename="pos\_tagged\_"+username+".txt"

handle=open(filename,"w")

for i in x:

handle.write(i+'\n')

**Function to expand the dictionary:**

ef learn(name):

list = []

filename = 'data\_emotions\_words\_list.csv'

fname = 'new\_dict.csv'

dict\_list = []

new\_list = []

l = 0

oh = []

oa = []

os = []

od = []

of = []

oc = []

with open(fname,encoding="ISO-8859-1",newline='') as n: #new dictionary

new\_reader = csv.reader(n)

for new\_row in new\_reader:

my\_new\_list = new\_row

#new\_list.append(my\_new\_list[0])

dict\_list.append(my\_new\_list[0])

oh.append(float(my\_new\_list[1]))

oa.append(float(my\_new\_list[2]))

os.append(float(my\_new\_list[3]))

of.append(float(my\_new\_list[4]))

od.append(float(my\_new\_list[5]))

oc.append(float(my\_new\_list[6]))

print(dict\_list[l]+","+str(oh[l]))

l += 1

list\_evalue = []

my\_new\_list = []

b = []

j = 0

words = []

#ans = [0.52,0.32,0.3,0.36,0.28]

'''

ans[0] = 0.48

ans[1] = 0.32

ans[2] = 0.3

ans[3] = 0.36

ans[4] = 0.28

'''

fi = open(name,'r')

tweets = fi.readlines()

for tweet in tweets:

if(tweet == '\n'):

continue

words[:] = []

words = tweet.split(' ')

for word in words:

if(word == '\n'):

continue

#print(word)

word=word.lower()

with open(filename,encoding="ISO-8859-1",newline='') as di: #original dictionary

reader = csv.reader(di)

for row in reader:

my\_list=row

if(word != my\_list[0]): # word not in original dictionary

if(word in new\_list):

if(word in dict\_list): # word in new dictionary and in tweet

continue

else:

j = new\_list.index(word) # repeated words

c[j] += 1

else:

l += 1

new\_list.append(word) # word not in new dictionary

h.append(ans[0])

a.append(ans[1])

s.append(ans[2])

f.append(ans[3])

d.append(ans[4])

c.append(1)

#print(new\_list[l-1]+","+str(h[l-1])+","+str(a[l-1])+","+str(s[l-1])+","+str(f[l-1])+","+str(d[l-1])+","+str(c[l-1]))

total = l

z = 0

'''

cnt = 0

with open(fname,encoding="ISO-8859-1",newline='') as n: #new dictionary

new\_reader = csv.reader(n)

for new\_row in new\_reader:

cnt += 1

'''

while(z<len(new\_list)):

if(c[z]>=1000):

c[z] = round((c[z]/1036),1)

z = z + 1

print("\n"+str(len(dict\_list))+"\n")

print(len(new\_list))

print("\n")

print(l)

i = 0

k = 0

z = 0

'''

flag2 = []

while(i<len(dict\_list)):

flag2[i] = 0

i += 1

'''

flag2 = [0]\*len(dict\_list)

flag3 = [0]\*len(new\_list)

while(z<len(new\_list)):

i = 0

flag = 0

while(i<len(dict\_list) and flag == 0):

if(new\_list[z] == dict\_list[i] and flag3[z] == 0): # in tweet and in dictionary

flag = 1

flag2[i] = 1

flag3[z] = 1

list.append(new\_list[z])

dh.append((h[z]\*c[z] + oh[i]\*oc[i])/(c[z]+oc[i]))

da.append((a[z]\*c[z] + oa[i]\*oc[i])/(c[z]+oc[i]))

ds.append((s[z]\*c[z] + os[i]\*oc[i])/(c[z]+oc[i]))

df.append((f[z]\*c[z] + of[i]\*oc[i])/(c[z]+oc[i]))

dd.append((d[z]\*c[z] + od[i]\*oc[i])/(c[z]+oc[i]))

dc.append(c[z]+oc[i])

else:

if(flag2[i] == 0): # only in dictionary

list.append(dict\_list[i])

dh.append(oh[i])

da.append(oa[i])

ds.append(os[i])

df.append(of[i])

dd.append(od[i])

dc.append(oc[i])

flag2[i] = 1

i += 1

if(flag == 0 and flag3[z] == 0):

list.append(new\_list[z])

dh.append(h[z]) # in tweet only

da.append(a[z])

ds.append(s[z])

df.append(f[z])

dd.append(d[z])

dc.append(c[z])

#print(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z]))

z += 1

z = 0

y = open("new\_dict.csv",'w')

while(z<len(list)):

print(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z]))

y.write(list[z]+","+str(dh[z])+","+str(da[z])+","+str(ds[z])+","+str(df[z])+","+str(dd[z])+","+str(dc[z]))

y.write("\n")

z = z + 1

y.close()

**Main Program:**

def run(username, master):

column0\_padx = 24

row\_pady = 36

tweets = []

print (username)

if(username[0] == "@"):

tweets = get\_all\_tweets(username)

print("Downloading of tweets of user has started !!")

if(username[0] == "#"):

tweets = get\_all\_hash(username)

print("Downloading of tweets of hashtag has started !!")

print("Tweets have been downloaded !!")

print("Now Cleaning of tweets starts !!")

#time.sleep(1)

'''

filename = username+"tweets.txt"

with open(filename) as f:

for line in f:

tweets.append(line)'''

tweets=rem\_substring(tweets,'#')

tweets=rem\_substring(tweets,'http')

tweets=rem\_substring(tweets,'@')

tweets=rem\_substring(tweets,'RT')

tweets=rem\_punctuation(tweets,'\"')

tweets=rem\_punctuation(tweets,'-')

tweets=rem\_punctuation(tweets,'!')

tweets=rem\_punctuation(tweets,':')

tweets=removeNonEnglish(tweets)

#tweets.replace("."," ")

for tweet in tweets:

tweet=tweet.replace("."," ")

z=open('cleaned\_'+username+'.txt','w')

for tweet in tweets:

tweet=tweet.encode('ascii','ignore')

z.write("\n")

z.write(tweet.decode())

z.close()

#filename = username+"\_cleaned\_tweets.txt"

#x = open(filename,"a")

'''for i in tweets:

x.write(i+'\n')

#time.sleep(2)'''

POS\_tagger(tweets,username)

print("Tweets have now been cleaned !!")

evalue = score(tweets)

learn("pos\_tagged\_"+username+".txt")

L3 = Label(master, text="Happiness : ", wraplength=150, justify='left', pady=row\_pady)

L3.grid(row=5, column=0, sticky='w', padx=column0\_padx)

L8 = Label(master, text=str(evalue[0]))

L8.grid(row=5, column=1, sticky='w')

L4 = Label(master, text="Anger : ", wraplength=150, justify='left', pady=row\_pady)

L4.grid(row=6, column=0, sticky='w', padx=column0\_padx)

L9 = Label(master, text=str(evalue[1]))

L9.grid(row=6, column=1, sticky='w')

L5 = Label(master, text="Sadness : ", wraplength=150, justify='left', pady=row\_pady)

L5.grid(row=7, column=0, sticky='w', padx=column0\_padx)

L10 = Label(master, text=str(evalue[2]))

L10.grid(row=7, column=1, sticky='w')

L6 = Label(master, text="Fear : ", wraplength=150, justify='left', pady=row\_pady)

L6.grid(row=8, column=0, sticky='w', padx=column0\_padx)

L11 = Label(master, text=str(evalue[3]))

L11.grid(row=8, column=1, sticky='w')

L7 = Label(master, text="Disgust : ", wraplength=150, justify='left', pady=row\_pady)

L7.grid(row=9, column=0, sticky='w', padx=column0\_padx)

L12 = Label(master, text=str(evalue[4]))

L12.grid(row=9, column=1, sticky='w')

bottom\_fram = Frame(master)

bottom\_fram.grid(row=10, column=0, columnspan=2, sticky='w')

btn\_start = ttk.Button(bottom\_fram, text = "Show Graph", width=20, command= lambda: new\_window())

btn\_start.pack(side='left', padx=145)

print("Tweets have now been cleaned !!")

def new\_window():

id = "Graph"

window = Toplevel(master)

label = ttk.Label(window, text=id)

label.pack(side="top", fill="both", padx=10, pady=10)

f = Figure(figsize=(5,5), dpi=100)

a = f.add\_subplot(111)

#t = arange(0.0,3.0,0.01)

#s = sin(2\*pi\*t)

#a.plot(t,s)

a.plot([1,2,3,4,5],[evalue[0], evalue[1], evalue[2], evalue[3], evalue[4]])

a.set\_title('Emotion Scores')

a.set\_xlabel('1 - > Happiness , 2 - > Anger , 3 - > Sadness , 4 - > Fear , 5 - > Disgust ')

a.set\_ylabel('Score')

# a tk.DrawingArea

canvas = FigureCanvasTkAgg(f, master=window)

canvas.show()

canvas.get\_tk\_widget().pack(side=TOP, fill=BOTH, expand=1)

toolbar = NavigationToolbar2TkAgg( canvas, window )

toolbar.update()

canvas.\_tkcanvas.pack(side=TOP, fill=BOTH, expand=1)

def on\_key\_event(event):

print('you pressed %s'%event.key)

key\_press\_handler(event, canvas, toolbar)

canvas.mpl\_connect('key\_press\_event', on\_key\_event)

def \_quit():

window.quit() # stops mainloop

window.destroy() # this is necessary on Windows to prevent

# Fatal Python Error: PyEval\_RestoreThread: NULL tstate

button = ttk.Button(master=window, text='Quit', command=\_quit)

button.pack(side=BOTTOM)

class App:

def \_\_init\_\_(self, master):

self.root = Frame(master)

column0\_padx = 24

row\_pady = 36

#Label entry

userart = Label(

master, text="Input User Name -> ",

wraplength=150, justify='left', pady=row\_pady)

entry\_point = Entry(master, width=30)

userart.grid(row=1, column=0, sticky='w', padx=column0\_padx)

entry\_point.grid(row=1, column=1, sticky='w')

# version

lbl\_version = ttk.Label(master, text="Beta-Version @TechnoDesign")

version = ttk.Label(master, text="ver. 1.004")

lbl\_version.grid(row=4, column=0, sticky='w', padx=column0\_padx)

version.grid(row=4, column=1, sticky='w')

sep = ttk.Label(master)

sep.grid(row=3, column=0, sticky='w')

#progress\_bar

#progressbar = ttk.Progressbar(orient='horizontal', length=200, mode='determinate')

#progressbar.grid(row=5, column=0, sticky='w', padx=column0\_padx)

#progressbar.start()

# buttons

bottom\_frame = Frame(master)

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='w')

btn\_start = ttk.Button(bottom\_frame, text = "Run", width=7, command=lambda: run(entry\_point.get(), master))

btn\_start.pack(side='left', padx=100)

btn\_exit = ttk.Button(bottom\_frame, text="Exit", width=7, command=self.root.quit)

btn\_exit.pack(side='left', padx=10)

root = Tk()

root.title("Emotion Calculator of TWITTER Data")

root.minsize(500, 700)

app = App(root)

root.mainloop()