**Twitter data analysis using Python:**

**Objective:**

Analysis of tweets.csv which contains data of public tweets related to Infosys on dates18, 19 and 20th august.

Understand various subjects of interaction during these dates, group similar messages.

**Steps and Output:**

**1. Build Corpus.**

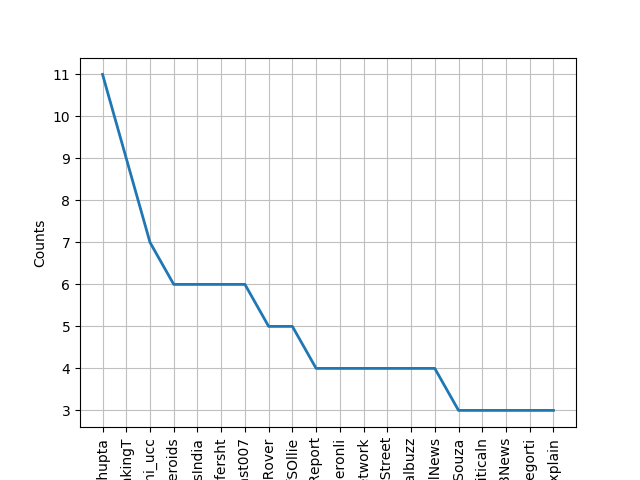
I already had twitter data containing stock tweets related to infy, dates 18,19 and 20 august.

Read the tweets.csv

#read data  
tweet\_df = pd.read\_csv("/home/nlp/aegis/tweets.csv", encoding = "ISO-8859-1")

**2. Find Top 20 tweet user**

users = tweet\_df['screenName'].tolist()  
fd = FreqDist(users)  
fd.plot(20)

****

**3. Text Pre-processing**

Remove non-english words, stopwords, links etc.

tweet\_content = tweet\_df["text"].tolist()  
stopwords = stopwords.words('english')  
english\_vocab = set(v.lower() for v in nltk.corpus.words.words())  
  
#Text Pre-processing  
def proc\_content(con):  
   if con.startswith('@null'):  
       return "[Invalid Tweet]"  
   con = re.sub(r'\$\w\*','',con)  
   con = re.sub(r'https?:\/\/.\*\/\w\*','',con)  
   con = re.sub(r'['+string.punctuation+']+', ' ',con)  
   twtok = TweetTokenizer(strip\_handles=True, reduce\_len=True)  
   tokens = twtok.tokenize(con)  
   tokens = [t.lower() for t in tokens if t not in stopwords and len(t) > 2 and t in english\_vocab]  
   return tokens  
   
words = []  
for tc in tweet\_content:  
    words += proc\_content(tc)

**Output:**

Word list generated is like-

['hell',  
 'handful',  
 'must',  
 'interest',  
 'minority',  
 'share',  
 'one',  
 'feel',  
 'sad',  
 'ongoing',  
 'needs',  
 'preserve',  
 'stood',  
 'one',…………..]

**4. Text Exploration**  
   
bigram\_measures = nltk.collocations.BigramAssocMeasures()  
finder = BigramCollocationFinder.from\_words(words, 10)  
finder.apply\_freq\_filter(10)  
print(finder.nbest(bigram\_measures.likelihood\_ratio, 10))

**Output:**

Most frequent words – bigrams are

[('affect', 'tech'), ('exit', 'affect'), ('tech', 'edge'), ('affect', 'edge'), ('ongoing', 'preserve'), ('feel', 'ongoing'), ('sad', 'ongoing'), ('preserve', 'stood'), ('needs', 'preserve'), ('ongoing', 'needs')]

**5. Clustering** is used to group similar tweets.

#Clustering  
contents\_clean= []  
for cc in tweet\_content:  
    words = proc\_content(cc)  
    content\_clean = " ".join(w for w in words if len(w) > 2 and w.isalpha())  
    contents\_clean.append(content\_clean)  
tweet\_df['CleanedTweet'] = contents\_clean  
   
vec\_tfidf = TfidfVectorizer(use\_idf=True, ngram\_range=(1,3))    
tfidfm = vec\_tfidf.fit\_transform(contents\_clean)    
feature\_names = vec\_tfidf.get\_feature\_names()   
distance = 1 - cosine\_similarity(tfidfm)    
print(distance)   
  
no\_clusters = 3    
km = KMeans(n\_clusters=no\_clusters)    
km.fit(tfidfm)    
clusters = km.labels\_.tolist()    
tweet\_df['ClusterID'] = clusters  
print(tweet\_df['ClusterID'].value\_counts())

**Output:**

Output shows 3 clusters. Clusters and no of tweets in the cluster as below,

**0    644  
1    233  
2    123**

Most of the tweets are found in cluster 0.

Mostly used words in cluster are-

prox\_cen = km.cluster\_centers\_.argsort()[:, ::-2]  
for i in range(no\_clusters):  
    print("Cluster {} : Words :".format(i))  
    for cent in prox\_cen[i, :10]:   
        print(' %s' % feature\_names[cent])

**Output:**

**Cluster 0 : Words :** via  tom  suit  resignation  class  suit recover   law mull class   class action suit  mull class action suit recover possible  
**Cluster 1 : Words :**  
 exit affect  exit  affect tech  edge  affect tech edge  tech edge technology  sex  sex exit affect getting reading last  zone  
**Cluster 2 : Words :**ongoing needs  preserve  feel sad ongoing  sad ongoing needs  one feel sad  
 needs preserve  feel  needs  stood  getting reading

**6. Topic modeling**

It finds main subjects in the set of tweets.

#topic modelling  
from gensim import corpora, models  
from nltk.corpus import stopwords  
from nltk.stem.wordnet import WordNetLemmatizer  
import string  
stop = set(stopwords.words('english'))  
exclude = set(string.punctuation)  
lemma = WordNetLemmatizer()  
def clean(dc):  
    stop\_free = " ".join([i for i in dc.lower().split() if i not in stop])  
    punc\_free = ''.join(ch for ch in stop\_free if ch not in exclude)  
    normalized = " ".join(lemma.lemmatize(word) for word in punc\_free.split())  
    return normalized  
texts = [text for text in contents\_clean if len(text) > 2]  
dc\_clean = [clean(dc).split() for dc in texts]  
dictionary = corpora.Dictionary(dc\_clean)  
dtm = [dictionary.doc2bow(dc) for dc in dc\_clean]  
ldamodel = models.ldamodel.LdaModel(dtm, num\_topics=3, id2word = dictionary, passes=5)  
for topic in ldamodel.show\_topics(num\_topics=6, formatted=False, num\_words=6):  
    print("Topic {}: Words: ".format(topic[0]))  
    topicwords = [w for (w, val) in topic[1]]  
    print(topicwords)

**Output:**

Topic 0: Words: ['exit', 'affect', 'tech', 'edge', 'board', 'retail']  
Topic 1: Words: ['resignation', 'short', 'via', 'fund', 'position', 'hedge']  
Topic 2: Words: ['need', 'one', 'feel', 'sad', 'preserve', 'ongoing']

**7. Doc2Vec , Kmeans**

#K-means and doc2vec

import gensim

from gensim.models.doc2vec import TaggedDocument  
tagdoc = []  
tagtweet = {}  
for index,i in enumerate(contents\_clean):  
    if len(i) > 2:   
         tag = u'SENT\_{:d}'.format(index)  
         sentence = TaggedDocument(words=gensim.utils.to\_unicode(i).split(),tags=[tag])  
         tagtweet[tag] = i  
         tagdoc.append(sentence)  
model = gensim.models.Doc2Vec(tagdoc, dm=0, alpha=0.05, size=20,   
min\_alpha=0.025, min\_count=0)  
for epoch in range(60):  
    if epoch % 20 == 0:  
        print('Training epoch %s' % epoch)  
    token\_count = sum([len(i) for i in tagdoc])       
    model.train(tagdoc,total\_examples = token\_count,epochs=model.iter)  
    model.alpha -= 0.002  # decrease the learning rate  
    model.min\_alpha = model.alpha  
  
  
data = model.wv.syn0  
cluster = KMeans(n\_clusters=6)  
centroid = cluster.fit\_predict(data)  
maptopic2word= {}  
for i, val in enumerate(data):  
    tag = model.docvecs.index\_to\_doctag(i)  
    topic = centroid[i]  
    if topic in maptopic2word.keys():  
        for w in (tagtweet[tag].split()):  
            maptopic2word[topic].append(w)  
    else:  
        maptopic2word[topic] = []  
for tw in maptopic2word:  
    words = maptopic2word[tw]  
    print("Topic {} has words {}".format(tw, words[:5]))

**Output:**

**Topics and mostly used words per list.**

Topic 0 has words ['lower', 'law', 'mull', 'class', 'action']  
Topic 1 has words ['enhanced', 'digital', 'experience', 'support', 'staff']  
Topic 2 has words ['unable', 'match', 'cultural', 'intellect', 'exit']  
Topic 3 has words ['one', 'feel', 'sad', 'ongoing', 'needs']  
Topic 4 has words ['one', 'feel', 'sad', 'ongoing', 'needs']  
Topic 5 has words ['enhanced', 'digital', 'experience', 'support', 'staff']