Unit4:-

1. Explain exploring of Twitter's API
2. Analysing tweets and tweet entities with frequency analysis
3. Discuss how to understand social graph APIs.
4. Apply graph API to examine the friendship from your own social n/w
5. Explain the fb's open graph protocol
6. Explain Details with necessary examples following terms:-

   i) Creating a twitter API Connection
   ii) Extracting tweets entities
   iii) Computing lexical diversity of tweets
   iv) searching the tweets.

7. illustrate with example the exploring trending topics & trends & examining patterns in tweets in retweet for mining twitter
8. Define social graph

Scanned with CamScanner

Unit 4:- Twitter:-

Twitter is a real time, highly social micro-blogging service that allows people to communicate with short (140 characters) msg roughly correspons to thoughts or idea.

1 Creating twitter API Connection:-

Requirements:-

→ A python package twitter is required.
→ RESTful API

Steps:-

1) Create an appl^n at https://dev.twitter.com/apps

2) for read only access to the API use, OAuth credentials to gain authorization to query twitter's API.

3) Instead of username/password Combination a more secure & flexible standardized protocol called OAuth which allows user to authorize 3rd party appl^n to access their account data should be utilized

4) OAuth Consist of:-
1) Consumer key
2) Consumer secret
3) Access token

Access token is required for making twitter's API.

Outh_token
outh_token_secrets
api_obj = twitter. oauth. OAuth (Consumer key,
consumer secret
access key
access token)

② Exploring trending topics:-

i) Retriving trends:-

→ use WOE (where on Earth) id system
→ set of trends for the entire world.

api_obj. trends. place (_id = WORLD)
Returns top 10 trending topics as an array of 'trend' objects.

ii) Searching for tweets:-

s_r = api_obj. search. tweets (hastag, count=200)
print s_r['statuses']

③ Analyzing the 140 character:-

Let us assume that we extracted a single tweet from the search and ~~search~~ stored it in a variable name t.

t. keys() returns the top level field for the tweet (id, created at, text, user, entities, extended entities, source)

→ t[id]:- accesses the identifier of the tweet.
→ t['text']:- access the human readable text of the tweet
→ t['entities']:- the entities in the text are accessed.
→ t['retweet_count'] refers to the total no.

of times the Original tweet has been retwe

→ t['favouriate_Count'] :- reflects the no of times tweet being bookmarked

→ t['retweeted_status'] :- this field tells abo details of Original tweet and its author

→ t['retweeted'] denotes wheather or not the authenticated user has retweeted this particular tweet via an authorized appl'

④ <u>Extracting tweet entities</u>:-

→ Analyzing the entities in the text of the tweet.

→ extracting the status text, screen names & hastags

```
status_text = [status['text'] for status in
                       statuses]

screen_names = [user_mention['screen_name']
                 for status in statuses
                 for user_mention in
                 status['entities']['user_mention']
               ]

hastags:- [hashtags['text']
            for status in statuses
            for hashtags in status['entities']
                               ['hastags']]
```

(5) compute collection of all words of all tweets:-

```
status-texts = [status['text']
                 for status in statuses]

words = [w for t in status-texts
          for w in t.split()]
```

(6) Creating a basic frequency distribution from the words in tweets:-

```
for items in [words, screen-names, hastags]
        c = Counter(item)
        print c. most-common() [:10]
                                    ↳ top 10

        print
```

(7) Calculating lexical Diversity for tweets:-

```
    def lexical-diversity (tokens)
        return 10 * len(set(tokens)) / len(tokens)
    print ~~presnt~~ lexical-diversity( words)
    print lexical-diversity(screennames)
    print lexical-diversity (hastag)
```

(8) extracting top 5 popular tweets:-

```
pt = pretty table (field-names) =
             ['Count', 'screen-name', 'text']
[pt.add-row (row) for row in
             sorted (retweets, reverse = TRUE) [:5]
 print pt
```

(9) Social graph:-

→ it is a Data structure

→ A graphical representation of interconnecti

of R/P among people as an organisat

in an online social n/w

(10) Social graph API:-

Components are:-

- Access token:- requird for making request to twitter's API

- Node id's:- A node with an id Correspon-ding to a person name

- Connection Constraints:-
Modifying the original query

- likes Constraints:-
A further modification in Original query is to add likes connection for each of your friends.

- Debugging:-
It helps in troubleshooting query related to problem.

- Json response format:-
The result of a graph API query are in a JSON format

(11) Open Graph Protocol:-

→ it is a protocol which is used to integrate any webpage into the social graph.
→ Once the webpage is integrated it behaves like the object of the social graph.
→ through this you can tell fb how your contents should be displayed on fb

```
<html xmlns: og = "http://ogp.me/ns#">
<head>
<title> The Rock </title>
<meta property ="og: title" content="The Rock"/>
<meta property = 'og: type' content = 'movie'/>
<meta property = "og: url" content =
        "http://www.imdb.com"/">
</head>
</html>
```

(12) Analyzing Social Graph Connection:-

```
pip install facebook-sdk.
import facebook.
g = facebook. GraphAPI (ACCESS_TOKEN)
 print g.get_object('me')
 print g.get_connection('me', 'friends')
```

steps:-
(i) Creating an objects for graph API.
(a) ACCESS_TOKEN will be based on OAuth

(ii) Retreving from Social graph API
 id, middle name etc

(iii) Details about friends
(iv) request info.

(13) Analyzing fb Pages:-

- Get an instance of Mining the web
- Using the page name also works
- eg:- 'Mining the social web' or 'crossfit'
- mtsw_id = '146803958708175'
- pp(g-get_object(mtsw-id))

(14) Analyzing coke v/s Pepsi:

```
# find pepsi & coke in search page

print g.request('search', 'q': 'pepsi',
                type : 'page', 'limit' :5)

print "pepsi likes", int_format(g.get_object(
                (p_id)['likes'])
```

for coke:-

```
print "coke likes", int_format(g.get_object
                (c_id)['likes'])
```

(15) Query for all friends likes:-

```
fr = g.get_connections ("me", "friends")['data'
ls = friend['name']: g.get_connections
                            (f['id'], "likes")
         ['data'] for friends in fr ]
print ls
```