Mining Facebook, Analyzing pages, Examining friendships, and more

Introduction

- Facebook is the heart of the social web, with more than half if its 1 billion users active each day updating statuses, posting photos, exchanging messages, etc.
- Facebook's API provides incredible opportunities to synthesize data into information and get valuable insights.

Main Concepts

- Facebook's social graph API and how to make API requests.
- Open Graph protocol and its relationship to Facebook's Social Graph.
- Analysing likes from Facebook pages and from facebook friends.
- Techniques such as clique analysis for analysing the social graphs.
- Visualising social graphs with the D3 Javascript library.

Exploring Facebook's Social Graph API

- What is API?

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

Ex: Kitchen-waiter-customer/ the booking website.

- Facebook has its own API-Social Graph API.

SOCIAL GRAPH API

- As a social web miner, the only way that you can access a Facebook user's account data is by registering an application and using that application as the entry point into the Facebook developer platform(developers.facebook.com). Moreover, the only data that's available to an application is whatever the user has explicitly authorized it to access.
- For example, as a developer writing a Facebook application, you'll be the user who's logging into the application, and the application will be able to access any data that you explicitly authorize it to access.

TERMS

- **Access token:**

  When someone connects with an app using Facebook Login, the app will be able to obtain an **access token** which provides temporary, secure **access** to Facebook APIs. An **access token(OAuth token-**commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords**)** is an opaque string that identifies a user, app, or Page and can be used by the app to make graph API calls.

  /{node-id}? fields=<first-level>{<second-level>}

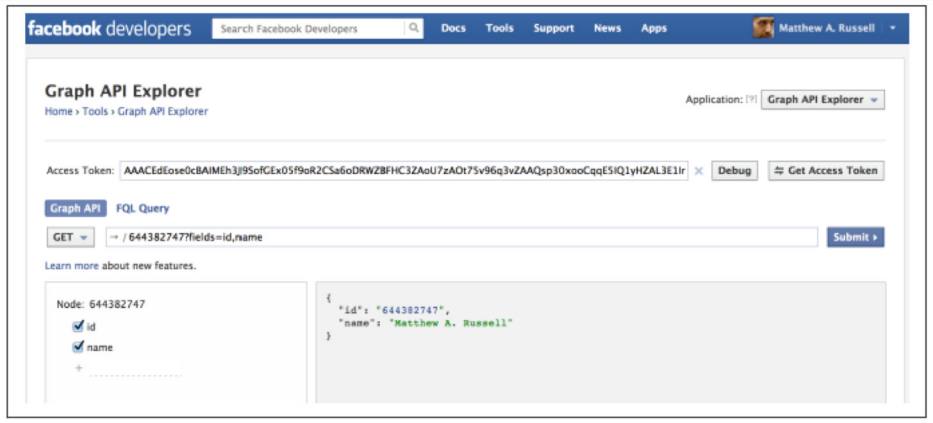


Figure 2-1. Using the Graph API Explorer application to progressively build up a query for friends' interests: a query for a node in the Social Graph

TERMS

- **Node IDs**

The basis of a query is a node with an ID corresponding to a user who is the logged in user of the graph explorer. The "id" and "name"values for a node are called fields.

- **Connection constraints**

You can modify the original query with a "friends", by clicking on the + and then scrolling to "friends" in the "connections" pop-up menu. The "friends" connections that appear in the console represent nodes that are connected to the original query node. At this point, you could click on any of the blue ID fields in these nodes and initiate a query with that particular node as the basis.

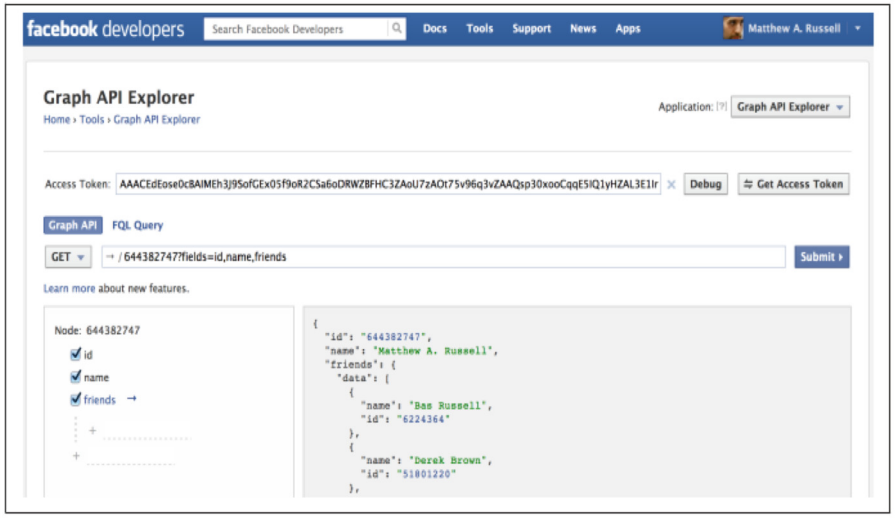


Figure 2-2. Using the Graph API Explorer application to progressively build up a query for friends' interests: a query for a node and connections to friends

TERMS

- **Likes constraints**

  A further modification to the original query is to add "likes" connections for each of your friends, as shown in Figure 2-3. Before you can retrieve likes connections for your friends, however, you must authorize the Graph API Explorer application to explicitly access your friends' likes by updating the access token that it uses and then approve this access, as shown in Figure 2-4.

- The Graph API Explorer allows you to easily authorize it by clicking on the Get Access Token button and checking the "friends_likes" box on the Friends Data Permissions tab.

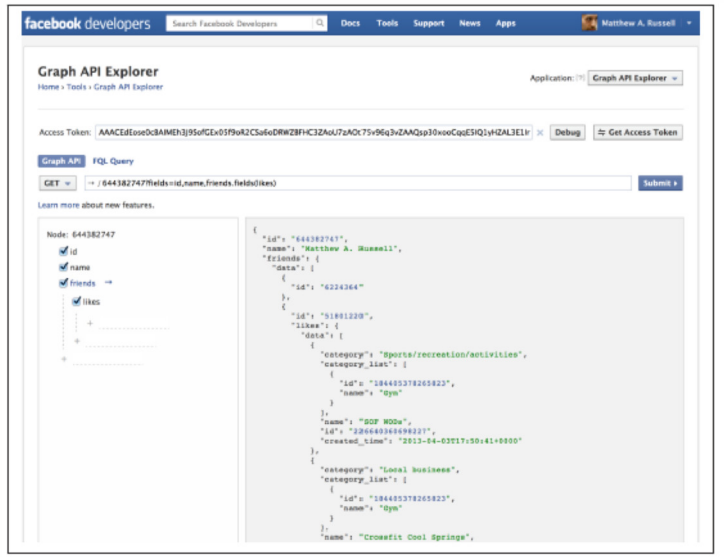


Figure 2-3. Using the Graph API Explorer application to progressively build up a query for friends' interests: a query for a node, connections to friends, and likes for those friends

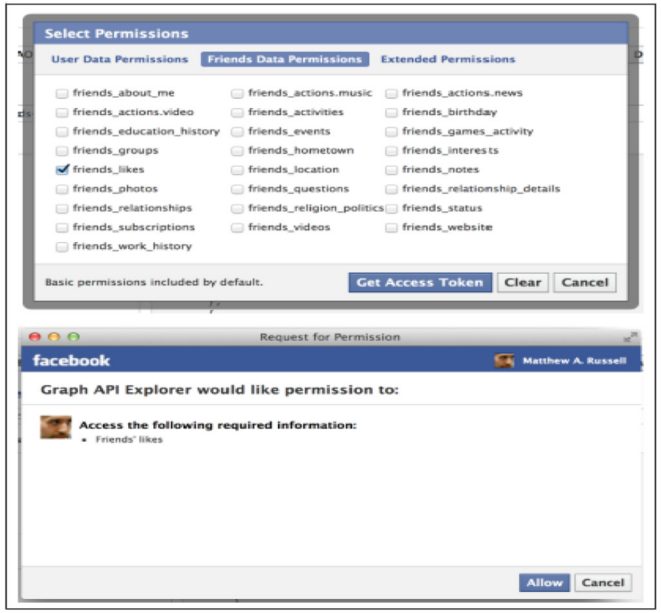


Figure 2-4. Facebook applications must explicitly request authorization to access a user's account data. Top: The Graph API Explorer permissions panel. Bottom: A Facebook dialog requesting authorization for the Graph API Explorer application to access friends' likes data.

Terms

- **Debugging**

The Debug button can be useful for troubleshooting queries that you think should

be returning data but aren't doing so based on the authorizations associated with the access token.

- **JSON response format**

The results of a Graph API query are returned in a convenient JSON format that can be easily manipulated and processed.

Ex-1:Making Graph API requests over HTTP

Example 2-1. Making Graph API requests over HTTP

```
import requests # pip install requests
import json

base_url = 'https://graph.facebook.com/me'

url = '%s?fields=%s&access_token=%s' % \
    (base_url, fields, ACCESS_TOKEN,)

# This API is HTTP-based and could be requested in the browser,
# with a command line utlity like curl, or using just about
# any programming language by making a request to the URL.
# Click the hyperlink that appears in your notebook output
# when you execute this code cell to see for yourself...
print url

# Interpret the response as JSON and convert back
# to Python data structures
content = requests.get(url).json()

# Pretty-print the JSON and display it
print json.dumps(content, indent=1)
```


Understanding the Open Graph Protocol

Introduction

- Facebook unveiled something called the Open Graph protocol (OGP) back in April 2010.

- OGP is a mechanism that enables developers to make any web page an object in Facebook's Social Graph by injecting some RDFa metadata into the page.

EXAMPLE

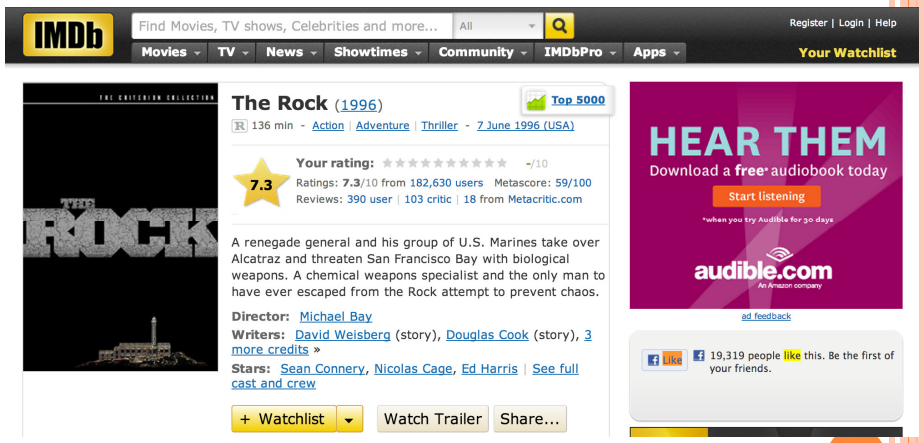


Figure 2-5. An IMDb page featuring an implementation of OGP for The Rock

IMPLEMENTATION OF OGP

XHTML document that uses namespaces looks something like this:

```
<html xmlns:og="http://ogp.me/ns#">
<head>
<title>The Rock (1996)</title>
<meta property="og:title" content="The Rock" />
<meta property="og:type" content="movie" />
<meta property="og:url"
  content="http://www.imdb.com/title/tt0117500/" />
<meta property="og:image" content="http://ia.media-
  imdb.com/images/rock.jpg" />
...
</head>
...
</html>
```


IMPLEMENTATION

At its core, querying the Graph API for Open Graph objects is incredibly simple:

For example, fetching the URL: http://graph.facebook.com/http://www.imdb.com/title/tt0117500 in your web browser would return this response:


```
{
"id": "114324145263104",
"name": "The Rock (1996)",
"picture": "http://profile.ak.fbcdn.net/hprofile-ak-
  snc4/hs344.snc4/...jpg",
"link": "http://www.imdb.com/title/tt0117500/",
"category": "Movie",
"description": "Directed by Michael Bay. With Sean Connery,
  ...",
"likes" : 3
}
```


IMPLEMENTATION

- you explicitly request additional metadata for an object in the page by appending the query string parameter metadata=1 to the request. Here is a sample response for the query:

  https://graph.facebook.com/114324145263104?metadata=1

Implementation

metadata": {
"connections": {
"feed": "http://graph.facebook.com/http://www.imdb.com/title/...",
"posts": "http://graph.facebook.com/http://www.imdb.com/title/...",
"tagged": "http://graph.facebook.com/http://www.imdb.com/title/...",
"statuses": "http://graph.facebook.com/http://www.imdb.com/title/...",
"links": "http://graph.facebook.com/http://www.imdb.com/title/...",
"notes": "http://graph.facebook.com/http://www.imdb.com/title/...",
"photos": "http://graph.facebook.com/http://www.imdb.com/title/...",
"albums": "http://graph.facebook.com/http://www.imdb.com/title/...",
"events": "http://graph.facebook.com/http://www.imdb.com/title/...",
"videos": "http://graph.facebook.com/http://www.imdb.com/title/...",
},

The items in metadata.connections are pointers to other nodes in the graph that you can crawl to get to other intriguing bits of data.

Analyzing Social Graph Connections

Introduction

- An official Python SDK for the Graph API is a community fork of that repository previously maintained by Facebook and can be installed using:

  pip install facebook-sdk.

Implementation Methods

Few key methods from the GraphAPI class that you need to know about in order to use the Graph API to fetch data:

1-get_object(self, id, **args)

Example usage: get_object("me", metadata=1)

2-get_objects(self, id, **args)

Example usage: get_objects(["me", "some_other_id"], metadata=1)

3-get_connections(self, id, connection_name, **args)

Example usage: get_connections("me", "friends", metadata=1)

4-request(self, path, args=None, post_args=None)

Example 2-2. Querying the Graph API with Python

```
import facebook # pip install facebook-sdk
import json

# A helper function to pretty-print Python objects as JSON
def pp(o):
print json.dumps(o, indent=1)

# Create a connection to the Graph API with your access
    token
g = facebook.GraphAPI(ACCESS_TOKEN)
```


Example

```
# Execute a few sample queries
print '----------'
print 'Me'
print '----------'
pp(g.get_object('me'))
print
print '----------'
print 'My Friends'
print '----------'
pp(g.get_connections('me', 'friends'))
print
print '----------'
print 'Social Web'
print '----------'
pp(g.request("search", {'q' : 'social web', 'type' : 'page'}))
```


to query for information about you, information about your friends, and the term social web.

Sample Output

```
----------
Me
----------
{
"last_name": "Russell",
"relationship_status": "Married",
"locale": "en_US",
"hometown": {
"id": "104012476300889",
"name": "Princeton, West Virginia"
},
"quotes": "The only easy day was yesterday.",
"favorite_athletes": [
{
"id": "112063562167357",
"name": "Rich Froning Jr. Fan Site"
}
],
```


Sample Output

```
My Friends
----------
{
"paging": {
"next": "https://graph.facebook.com/644382747/friends?...",
},
"data": [
{
"name": "Bas Russell",
"id": "6224364"
},
...
{
"name": "Jamie Lesnett",
"id": "100002388496252"
}
]
}
```

Sample Output

Social Web

{
"paging": {
"next": "https://graph.facebook.com/search?q=social+web&type=page...",
},
"data": [
{
"category": "Book",
"name": "Mining the Social Web“,
"id": "146803958708175"
},
{
"category": "Internet/software",
"name": "Social & Web Marketing",
"id": "172427156148334"
}

Analyzing Facebook Pages

Introduction

- Facebook started out as more of a pure social networking site.
- It quickly adapted to take advantage of the market needs.
- Fast-forward a few years, and now businesses, clubs, books, and many other kinds of nonperson entities have Facebook pages with a fan base.
- Facebook pages are a powerful tool for businesses to engage their customers, and Facebook has gone to some lengths to provide tools that allow Facebook page administrators to understand their fans.

ANALYSIS

- How popular is the page?
- How engaged are the page's fans?
- Are any of the fans for the page particularly outspoken and participatory?
- What are the most common topics being talked about on the page?

- Your imagination is the **only limitation** to what you can ask of the Graph API for a Facebook page when you are **mining its content for insights** and questions should get you headed in the right direction.

- For any of the items in the search results, we could use the ID as the basis of a graph query through get_object with an instance of facebook.GraphAPI. If you don't have a numeric string ID handy, just use the page name (such as "MiningTheSocialWeb") that appears in the URL bar of your browser when you visit the page.

QUERY

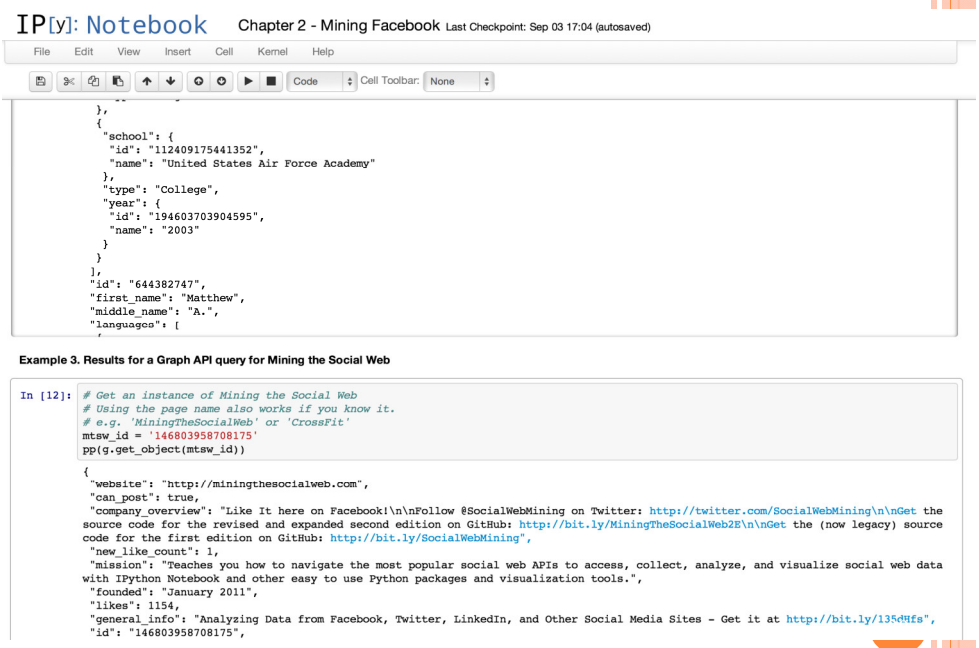
- *# Get an instance of Mining the Social Web*
- *# Using the page name also works if you know it.*
- *# e.g. 'MiningTheSocialWeb' or 'CrossFit'*
- mtsw_id = '146803958708175'
- pp(g.get_object(mtsw_id))

RESULT

- {
- **"category"**: "Book",
- **"username"**: "MiningTheSocialWeb",
- **"about"**: "Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social...",
- **"talking_about_count"**: 22,
- **"description"**: "Facebook, Twitter, and LinkedIn generate a tremendous ...",
- **"company_overview"**: "Like It here on Facebook!\n\nFollow @SocialWebMining...",
- **"release_date"**: "January 2011",
- **"can_post"**: **true**,
- **"cover"**: {
- **"source"**: "https://sphotos-b.xx.fbcdn.net/...",
- **"cover_id"**: 474206292634605,
- **"offset_x"**: -41,
- **"offset_y"**: 0
- },

- **"mission"**: "Teaches you how to...\n\n* Get a straightforward synopsis of ...",
- **"name"**: "Mining the Social Web",
- **"founded"**: "January 2011",
- **"website"**: "http://amzn.to/d1Ci8A",
- **"link"**: "http://www.facebook.com/MiningTheSocialWeb",
- **"likes"**: 911,
- **"were_here_count"**: 0,
- **"general_info"**: "Analyzing Data from Facebook, Twitter, LinkedIn, ...",
- **"id"**: "146803958708175",
- **"is_published"**: **true**
- }

- The interesting analytical results from the query response are the book's **talk**
- **ing_about_count** and **like_count.**



Example 3. Results for a Graph API query for Mining the Social Web

```
In [12]: # Get an instance of Mining the Social Web
         # Using the page name also works if you know it.
         # e.g. 'MiningTheSocialWeb' or 'CrossFit'
         mtsw_id = '146803958708175'
         pp(g.get_object(mtsw_id))
```


```
{
 "website": "http://miningthesocialweb.com",
 "can_post": true,
 "company_overview": "Like It here on Facebook!\n\nFollow @SocialWebMining on Twitter: http://twitter.com/SocialWebMining\n\nGet the source code for the revised and expanded second edition on GitHub: http://bit.ly/MiningTheSocialWeb2E\n\nGet the (now legacy) source code for the first edition on GitHub: http://bit.ly/SocialWebMining",
 "new_like_count": 1,
 "mission": "Teaches you how to navigate the most popular social web APIs to access, collect, analyze, and visualize social web data with IPython Notebook and other easy to use Python packages and visualization tools.",
 "founded": "January 2011",
 "likes": 1154,
 "general_info": "Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites - Get it at http://bit.ly/135dHfs",
 "id": "146803958708175",
```


Comparing Likes between Coke and Pepsi Fan Pages:

- *# Find Pepsi and Coke in search results*
- pp(g.request('search', {'q' : 'pepsi', 'type' : 'page', 'limit' : 5}))
- pp(g.request('search', {'q' : 'coke', 'type' : 'page', 'limit' : 5}))
- *# Use the ids to query for likes*
- pepsi_id = '56381779049' *# Could also use 'PepsiUS'*
- coke_id = '40796308305' *# Could also use 'CocaCola'*

- *# A quick way to format integers with commas every 3 digits*
- **def** int_format(n): **return** "{:,}".format(n)
- **print** "Pepsi likes:", int_format(g.get_object(pepsi_id)['likes'])
- **print** "Coke likes:", int_format(g.get_object(coke_id)['likes'])
- The results are somewhat striking:
- Pepsi likes: 9,677,881
- Coke likes: 62,735,664

QUERYING A PAGE FOR ITS "FEED" AND "LINKS" CONNECTIONS

- pp(g.get_connections(pepsi_id, 'feed'))
- pp(g.get_connections(pepsi_id, 'links'))
- pp(g.get_connections(coke_id, 'feed'))
- pp(g.get_connections(coke_id, 'links'))

- Are there any topics or special interests that are especially pronounced within your social network?
- Does your social network contain many mutual friendships or even larger cliques?
- How well connected are the people in your social network?
- Are any of your friends particularly outspoken or passionate about anything you might also be interested in learning more about?

QUERYING FOR ALL OF YOUR FRIENDS' LIKES

- *# First, let's query for all of the likes in your social*
- *# network and store them in a slightly more convenient*
- *# data structure as a dictionary keyed on each friend's*
- *# name. We'll use a dictionary comprehension to iterate*
- *# over the friends and build up the likes in an intuitive*
- *# way, although the new "field expansion" feature could*
- *# technically do the job in one fell swoop as follows:*
- *#*
- *# g.get_object('me', fields='id,name,friends.fields(id,name,likes)')*
- *#*
- *# See Appendix C for more information on Python tips such as*
- *# dictionary comprehensions*
- **friends = g.get_connections("me", "friends")['data']**
- **likes = { friend['name'] : g.get_connections(friend['id'], "likes")['data']**
- **for friend in friends }**
- **print likes**

Calculating the most popular likes among your friends

```
# Analyze all likes from friendships for frequency
# pip install prettytable
from prettytable import PrettyTable
from collections import Counter
friends_likes = Counter([like['name']
for friend in likes
for like in likes[friend]
if like.get('name')])
pt = PrettyTable(field_names=['Name', 'Freq'])
pt.align['Name'], pt.align['Freq'] = 'l', 'r'
[ pt.add_row(fl) for fl in
friends_likes.most_common(10) ]
print 'Top 10 likes amongst friends'
print pt
```


Result:

```
Top 10 likes amongst friends
+-------------------------------------+------+
| Name                                | Freq |
+-------------------------------------+------+
| Crossfit Cool Springs               | 14 |
| CrossFit                            | 13 |
| The Pittsburgh Steelers             | 13 |
| Working Out                         | 13 |
| The Bible                           | 13 |
| Skiing                              | 12 |
| Star Trek                           | 12 |
| Seinfeld                            | 12 |
| Jesus                               | 12 |
+-------------------------------------+------+
```


Calculating the most popular categories for likes among your friends

```
# Analyze all like categories by frequency
friends_likes_categories =
Counter([like['category']
for friend in likes
for like in likes[friend]])
pt = PrettyTable(field_names=['Category', 'Freq'])
pt.align['Category'], pt.align['Freq'] = 'l', 'r'
[ pt.add_row(flc) for flc in
friends_likes_categories.most_common(10) ]
print "Top 10 like categories for friends"
print pt
```



```
Top 10 like categories for friends
+-------------------------------+------+
| Category                      | Freq |
+-------------------------------+------+
| Musician/band                 | 62 |
| Book                          | 46 |
| Movie                         | 43 |
| Interest                      | 40 |
| Tv show                       | 31 |
| Public figure                 | 31 |
| Local business                | 25 |
| Community                     | 24 |
| Non-profit organization       | 21 |
| Product/service               | 17 |
+-------------------------------+------+
```

Calculating the number of likes for each friend and sorting by frequency

```
# Build a frequency distribution of number of likes by
# friend with a dictionary comprehension and sort it in
# descending order
from operator import itemgetter
num_likes_by_friend = { friend : len(likes[friend])
for friend in likes }
pt = PrettyTable(field_names=['Friend', 'Num Likes'])
pt.align['Friend'], pt.align['Num Likes'] = 'l', 'r'
[ pt.add_row(nlbf)
for nlbf in sorted(num_likes_by_friend.items(),
key=itemgetter(1),
reverse=True) ]
print "Number of likes per friend"
print pt
```



```
Number of likes per friend
+-------------------+-----------+
| Friend        | Num Likes |
+-------------------+-----------+
| Joshua        | 187 |
| Derek         | 146 |
| Heather       | 84 |
| Rick                  | 69 |
| Patrick       | 42 |
| Bryan         | 38 |
| Ray                   | 17 |
| Jamie         | 14 |
| ...           | ... |
| Bas                   | 0 |
+-------------------+-----------+
```


Finding common likes between an ego and its friendships in a social network

```
# Which of your likes are in common with which friends?
my_likes = [ like['name']
for like in g.get_connections("me", "likes")['data'] ]
pt = PrettyTable(field_names=["Name"])
pt.align = 'l'
[ pt.add_row((ml,)) for ml in my_likes ]
print "My likes"
print pt
# Use the set intersection as represented by the ampersand
# operator to find common likes.
common_likes = list(set(my_likes) & set(friends_likes))
pt = PrettyTable(field_names=["Name"])
pt.align = 'l'
[ pt.add_row((cl,)) for cl in common_likes ]
Print
print "My common likes with friends"
print pt
```



```
My likes
+-----------------------------+
| Name |
+-----------------------------+
| Snatch (weightlifting)     |
| First Blood                |
| Robinson Crusoe    |
| The Godfather      |
| The Godfather      |
| ...                |
| The Art of Manliness       |
| USA Triathlon              |
| CrossFit           |
| Mining the Social Web   |
+-----------------------------+
```

- My common likes with friends
- +--+
- | Name |
- +--+
- | www.SEALFIT.com |
- | Rich Froning Jr. Fan Site |
- | CrossFit |
- | The Great Courses |
- | The Art of Manliness |
- | Dan Carlin - Hardcore History |
- | Mining the Social Web |
- | Crossfit Cool Springs |
- +--+

- *# Which of your friends like things that you like?*
- similar_friends = [(friend, friend_like['name'])
- **for** friend, friend_likes **in** likes.items()
- **for** friend_like **in** friend_likes
- **if** friend_like.get('name') **in** common_likes]
- *# Filter out any possible duplicates that could occur*
- ranked_friends = Counter([friend **for** (friend, like) **in** list(set(similar_friends))])
- pt = PrettyTable(field_names=["Friend", "Common Likes"])
- pt.align["Friend"], pt.align["Common Likes"] = 'l', 'r'
- [pt.add_row(rf)
- **for** rf **in** sorted(ranked_friends.items(),
- key=itemgetter(1),
- reverse=True)]
- **print** "My similar friends (ranked)"
- **print** pt

Calculating the friends most similar to an ego in a social network

- *# Also keep in mind that you have the full range of plotting*
- *# capabilities available to you. A quick histogram that shows*
- *# how many friends.*
- plt.hist(ranked_friends.values())
- plt.xlabel('Bins (number of friends with shared likes)')
- plt.ylabel('Number of shared likes in each bin')
- *# Keep in mind that you can customize the binning*
- *# as desired. See http://matplotlib.org/api/pyplot_api.html*
- *# For example...*
- **plt.figure() # *Display the previous plot***
- **plt.hist(ranked_friends.values(),**
- **bins=arange(1,max(ranked_friends.values()),1))**
- **plt.xlabel('Bins (number of friends with shared likes)')**
- **plt.ylabel('Number of shared likes in each bin')**
- **plt.figure() # *Display the working plot***

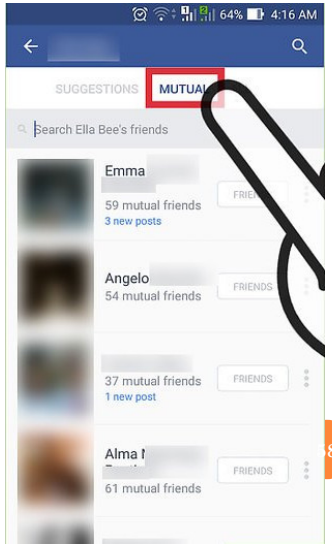
- My similar friends (ranked)
- +----------+-----------------------+
- | Friend | Common Likes |
- +----------+-----------------------+
- | Derek | 7 |
- | Jamie | 4 |
- | Joshua | 3 |
- | Heather | 3 |
- | ... | ... |
- | Patrick | 1 |
- +----------+-----------------------+
- As you are probably thinking, there is

Examining The FRiendship

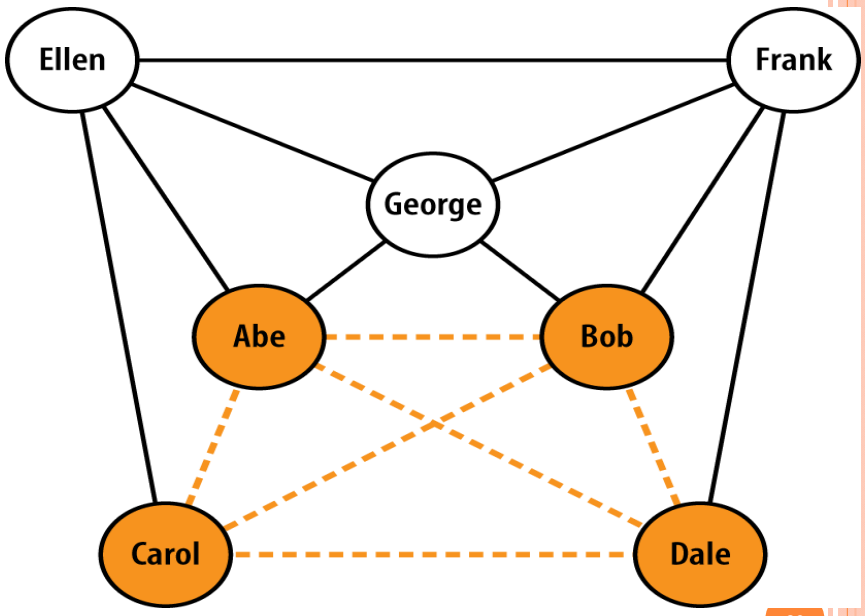
- Analyzing mutual friendships with directed graphs
- Visualizing directed graphs of mutual friendships
- Conclusion

Analyzing mutual friendships with directed graphs

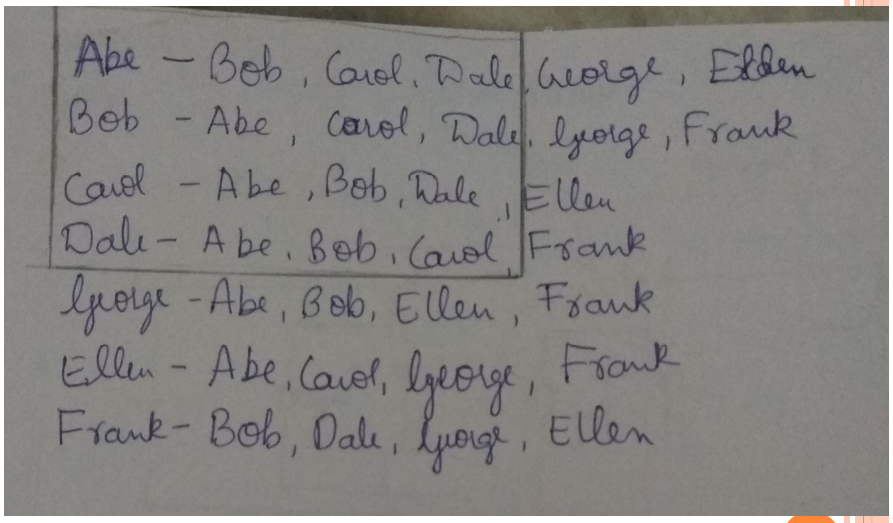
- **Mutual friends** are the people who are **Facebook friends** with both you and the person whose profile you're viewing.

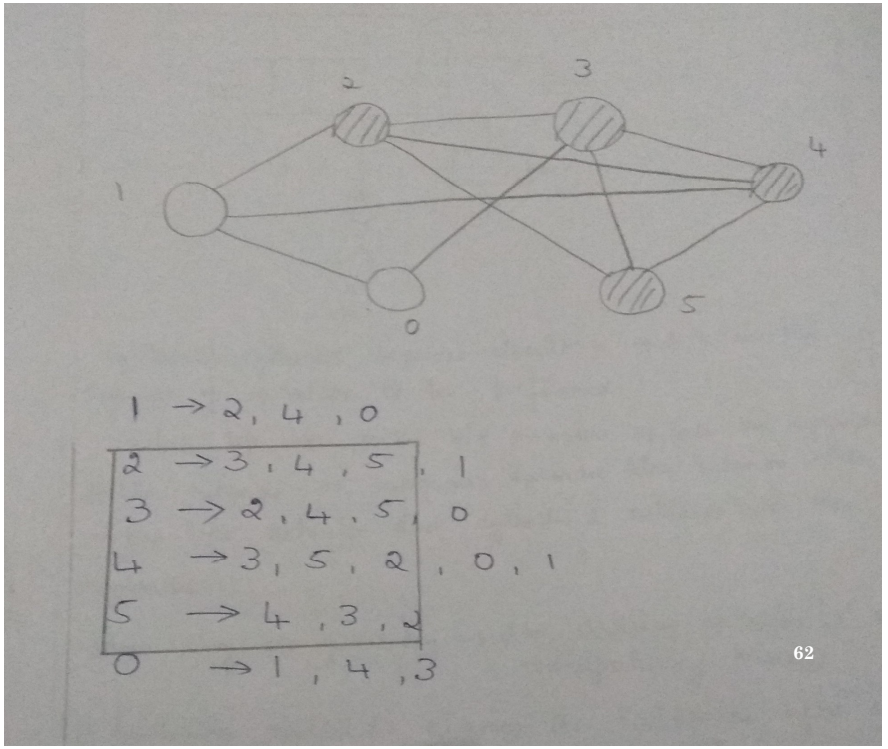


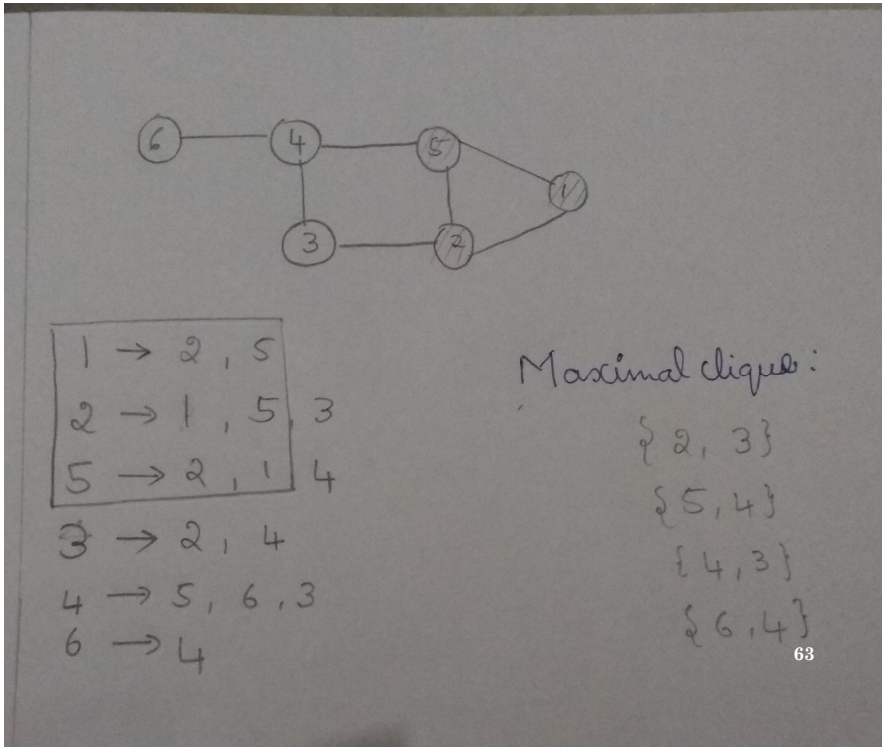
- Graph API is used to access data for the authenticating user and the authenticating user's friends.
- mutualfriends API
- Analysis of an ego graph for mutual friendships can be formulated using ***clique detection problem***
- *Clique represent subset of all people who know each other*



graph containing a maximum clique of size 4







- Given two social networks, comparing the sizes of the maximum friendship cliques provide a good starting point for analysis about various aspects of group dynamics.
- NP complete problem
- NetworkX package includes find_cliques method

Constructing a graph of mutual friendships :

```
import networkx as nx # pip install networkx
import requests # pip install requests

friends = [ (friend['id'], friend['name'],)
              for friend in g.get_connections('me', 'friends')['data'] ]

url = 'https://graph.facebook.com/me/mutualfriends/%s?access_token=%s'

mutual_friends = {}

# This loop spawns a separate request for each iteration, so
# it may take a while. Optimization with a thread pool or similar
# technique would be possible.
for friend_id, friend_name in friends:
    r = requests.get(url % (friend_id, ACCESS_TOKEN,) )
    response_data = json.loads(r.content)['data']
    mutual_friends[friend_name] = [ data['name']
                                    for data in response_data ]

nxg = nx.Graph()

[ nxg.add_edge('me', mf) for mf in mutual_friends ]

[ nxg.add_edge(f1, f2)
  for f1 in mutual_friends
      for f2 in mutual_friends[f1] ]

# Explore what's possible to do with the graph by
# typing nxg.<tab> or executing a new cell with
# the following value in it to see some pydoc on nxg
print nxg
```

Finding and analyzing cliques in a graph of mutual friendships:

```
# Finding cliques is a hard problem, so this could
# take a while for large graphs.
# See http://en.wikipedia.org/wiki/NP-complete and
# http://en.wikipedia.org/wiki/Clique_problem.

cliques = [c for c in nx.find_cliques(nxg)]

num_cliques = len(cliques)

clique_sizes = [len(c) for c in cliques]
max_clique_size = max(clique_sizes)
avg_clique_size = sum(clique_sizes) / num_cliques

max_cliques = [c for c in cliques if len(c) == max_clique_size]
```

```
num_max_cliques = len(max_cliques)

max_clique_sets = [set(c) for c in max_cliques]
friends_in_all_max_cliques = list(reduce(lambda x, y: x.intersection(
                                         max_clique_sets))

print 'Num cliques:', num_cliques
print 'Avg clique size:', avg_clique_size
print 'Max clique size:', max_clique_size
print 'Num max cliques:', num_max_cliques
print
print 'Friends in all max cliques:'
print json.dumps(friends_in_all_max_cliques, indent=1)
print
print 'Max cliques:'
print json.dumps(max_cliques, indent=1)
```

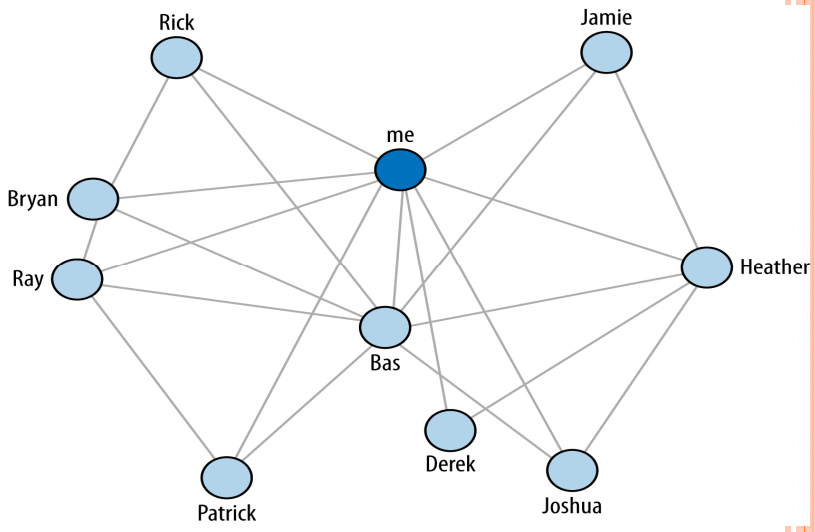
```
Num cliques: 6
Avg clique size: 3
Max clique size: 4
Num max cliques: 4

Friends in all max cliques:
[
 "me",
 "Bas"
]

Max cliques:
[
 [
  "me",
  "Bas",
  "Joshua",
  "Heather"
 ],
 [
  "me",
  "Bas",
  "Ray",
  "Patrick"
 ],
 [
  "me",
  "Bas",
  "Ray",
  "Rick"
 ],
 [
  "me",
  "Bas",
  "Jamie",
  "Heather"
 ]
]
```


VISUALIZING DIRECTED GRAPHS OF MUTUAL FRIENDSHIPS

- D3.js is a JavaScript toolkit that can render some beautiful visualizations in the browser.



A graph of mutual friendships within a Facebook social network

Example 2-15. Serializing a NetworkX graph to a file for consumption by D3

```
from networkx.readwrite import json_graph

nld = json_graph.node_link_data(nxg)

json.dump(nld, open('resources/ch02-facebook/viz/force.json','w'))
```


Visualizing a mutual friendship graph with D3:

```
from IPython.display import IFrame
from IPython.core.display import display


# IPython Notebook can serve files and display them into
# inline frames. Prepend the path with the 'files' prefix.

viz_file = 'files/resources/ch02-facebook/viz/force.html'

display(IFrame(viz_file, '100%', '600px'))
```


CONCLUSION

- To study about graph API
- How open graph protocol create connection
- How to programmatically query social graph