Introduction

Social Network Analysis is the study of network behavior in social structures by leveraging the concepts in Graph Theory and Psychology. It is becoming popular in the recent times due to increasing social media usage and its applications in targeted advertisements, truth index of social posts, friendship recommendation, identification of key people and even in preventing epidemics!

In this tutorial, we will see the social network analysis on GitHub connections between people and the repositories.

Before getting into the tutorial, get motivated by this SNA 101 video https://www.youtube.com/watch?v=tbRF_ELh0Nc (https://www.youtube.com/watch?v=tbRF_ELh0Nc) by Prof. Sudarshan.

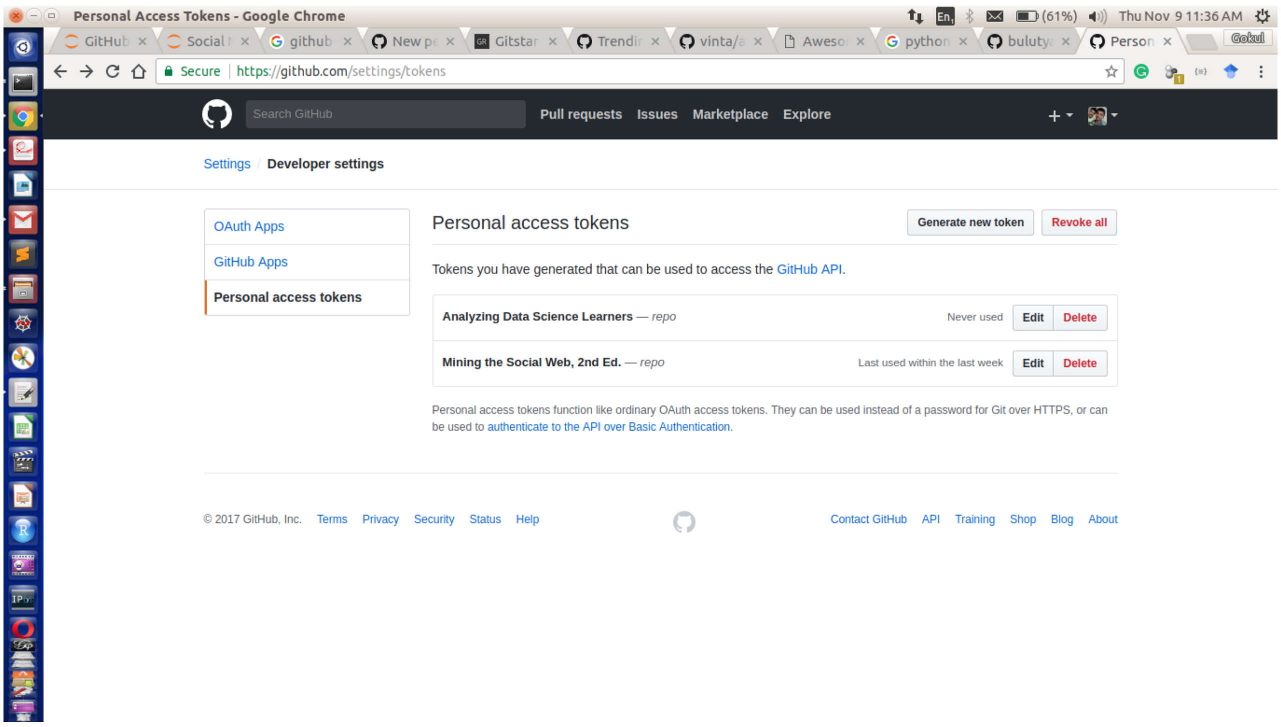
Resources: "Mining the Social Web", "Crowds, Networks and Markets", "NPTEL: Social Networks"

Table of Contents

# 1. Generating GitHub Access Key

```
Like other social networking sites Twitter and Facebook, GitHub also has **GitHub
API v3** which could be used to used to extract much useful information.
Information like followers, starred repositories, the programming language of the
repositories and much more can be fetched using this API.

We need an access key to use this API. Follow the steps to get the access key.
```

1. Login github.com
2. Go to Settings -> Developer settings -> Personal access tokens



Personal access tokens

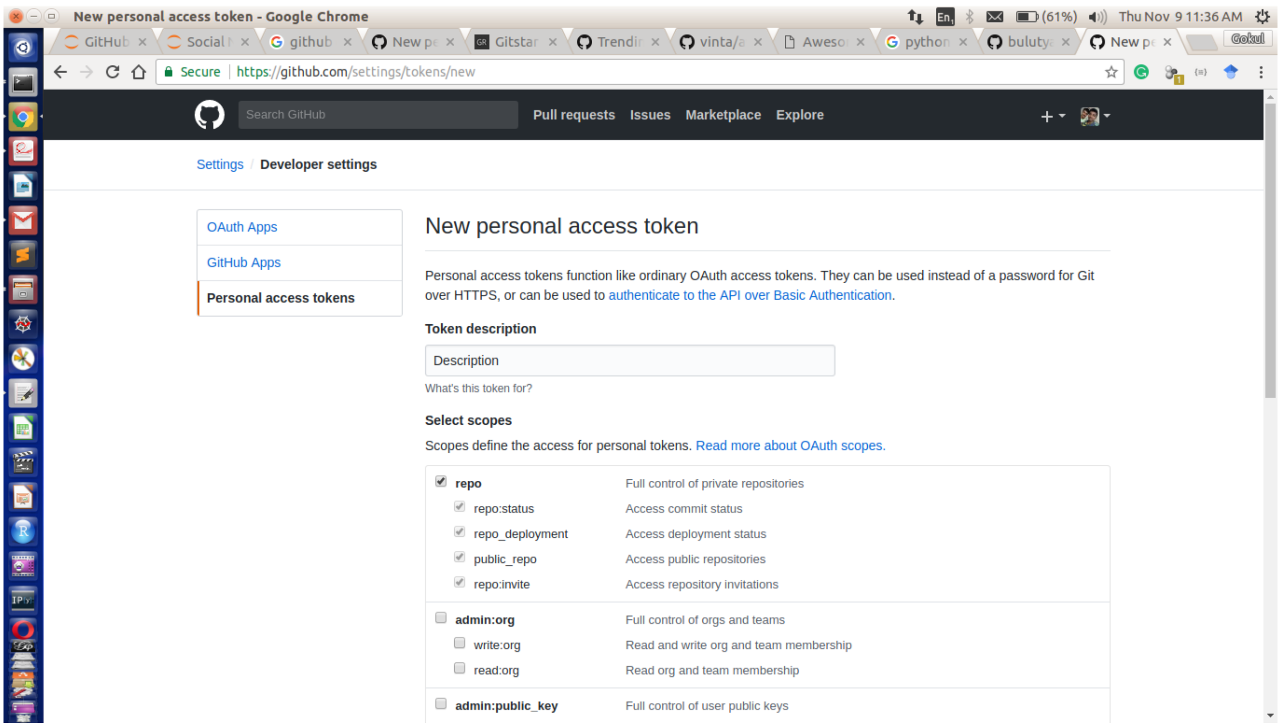
Tokens you have generated that can be used to access the GitHub API.

Analyzing Data Science Learners — *repo*	Never used	Edit Delete
Mining the Social Web, 2nd Ed. — *repo*	Last used within the last week	Edit Delete

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

3. Click on Generate New Token


New personal access token - Google Chrome
Secure | https://github.com/settings/tokens/new
Search GitHub
Pull requests
Issues
Marketplace
Explore
Settings / Developer settings
OAuth Apps
GitHub Apps
Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Token description

Description

What's this token for?

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ **repo**	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☐ **admin:org**	Full control of orgs and teams
☐ write:org	Read and write org and team membership
☐ read:org	Read org and team membership
☐ **admin:public_key**	Full control of user public keys

4. Check the repo box and fill the description.
5. Generate token

Don't forget to note down the token locally as it will not be accessible via the website next time.

2. Getting Started With PyGitHub

PyGitHub is the wrapper library over GitHub API v3 and it makes our job easier!

Install it using PIP.

In [ ]:

```
#pip install PyGitHub
```

```
It's simple to work with PyGitHub. Just create the instance and play with the
objects!
Let's see a quick code:
```

In [1]:

```
from github import Github

USER = "GokulKarthik"
PASSWD = "gh.gokul98"
g = Github(USER, PASSWD)

for repo in g.get_user().get_repos():
    print repo.name
```

```
Applied-Text-Mining-in-Python
Applied_Data_Science_with_Python_UMich
coursera-ml
DataEngineering-SIG
datasciencecoursera
datasharing
Infinity-17-NLP_Workshop
interview-guide
Mini-Projects
NPTEL-SocialNetworks
python-for-competitive-programming
Python-for-Data-Science-and-Machine-Learning-Bootcamp---Udemy
R-PAD
rosetta
Social-Networks
TCE-NSS-Website
```

Creating the instance with the user credentials allow us to access his/her information as above.

3. Graph Collection

3.1. Case Study :

Throughout this tutorial, I am going to analyze the social network of data science learners in GitHub.

But how do we collect the graph? In this NPTEL video, Prof. Sudarshan beautifully explains the concept of "Collecting Web Graph by random walk".

GitHub is a very large network and we need GPUs and graph database to work with such a large dataset. So we are going to take the sample network data of data science learners community.

Then comes another problem! From which node I have to start? The answer is 'any'. But if we start from the popular data science resource/tutorial repository, we can easily capture more information about the data science learning community. Google page rank suggested me to start with Awesome Datascience github repository by Bulut Yazılım.

3.2. Getting Dirty With Networkx

NetworkX is the graph computing library in python. We can easily create the graph using networkX by reading graph files like gexf, gml, graphml, pajek net or by simply adding nodes and edges.

Install it using pip

In [ ]:

```
#pip install networkx
```

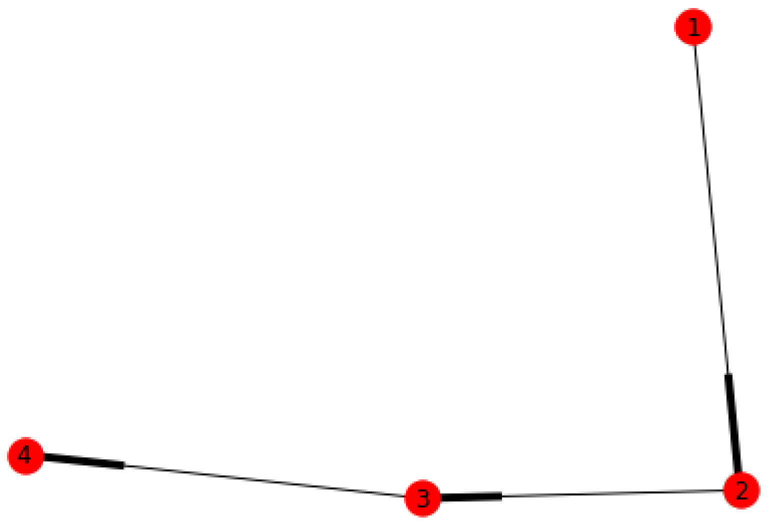
In [5]:

```
#importing the package
import networkx as nx

#Creating the NULL graph
G1 = nx.Graph()

#Adding the edges will automatically add the nodes
G1.add_edges_from([(1,2), (2,3), (3,4)])

#Visulazing using matplotlib
import matplotlib.pyplot as plt
nx.draw(G1, with_labels=True)
plt.show()
```

```
----------------------------------------------------------------------
-----
ImportError                               Traceback (most recent call
 last)
<ipython-input-5-c1076c757253> in <module>()
      1 #importing the package
----> 2 import networkx as nx
      3 
      4 #Creating the NULL graph
      5 G1 = nx.Graph()

/home/gokul/anaconda2/lib/python2.7/site-packages/networkx/__init__.py
 in <module>()
     72 
     73 # Release data
---> 74 from networkx import release
     75 
     76 __author__ = '%s <%s>\n%s <%s>\n%s <%s>' % \

ImportError: cannot import name release
```

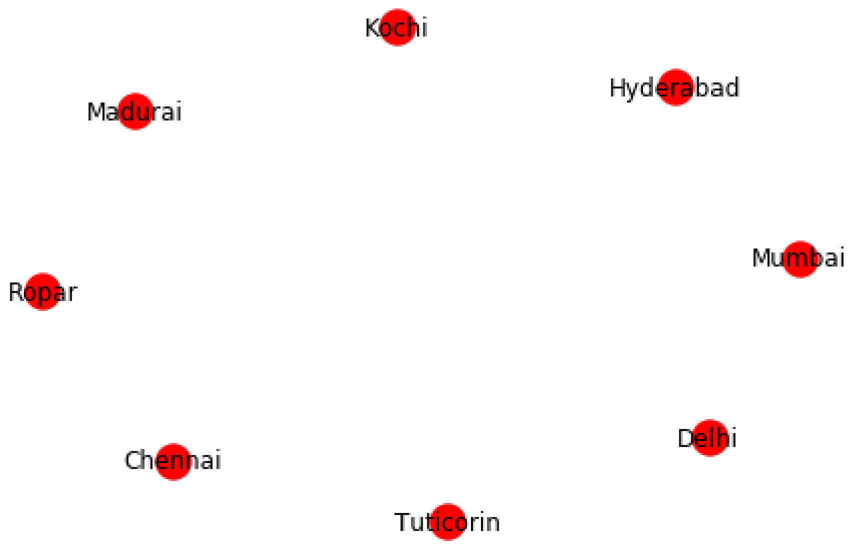
In [14]:

```
#Creating the directed graph
G2 = nx.DiGraph([(1,2), (2,3), (3,4)])
nx.draw(G2, with_labels=True)
plt.show()
```



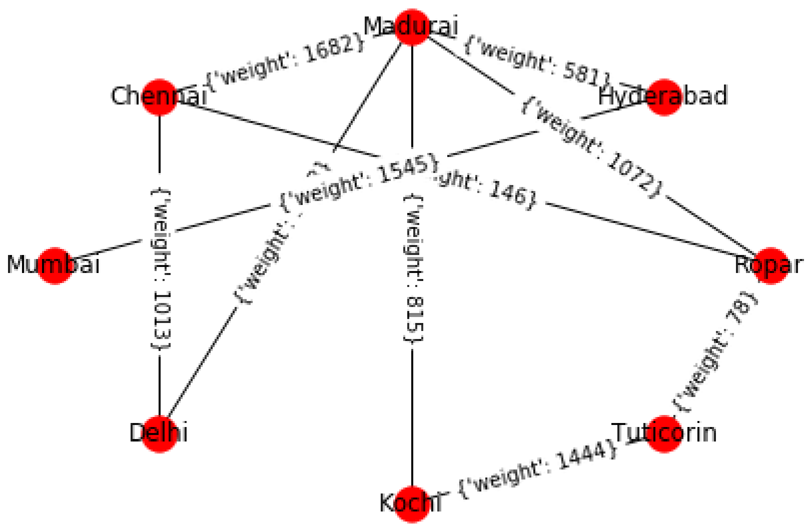
In [15]:

```
#modelling road network
G3 = nx.Graph()
cities = ['Delhi', 'Madurai', 'Mumbai', 'Tuticorin', 'Chennai', 'Kochi', 'Hyderabad
for city in cities:
    G3.add_node(city)
nx.draw(G3, with_labels=1)
plt.show()
```



In [20]:

```
import random as rd

#Addding 20 edges to the graph
while(G3.number_of_edges() != 10):
    city1 = rd.choice(G3.nodes())
    city2 = rd.choice(G3.nodes())
    #selecting random costs
    wt = rd.randint(20, 2000)
    if city1 != city2 and G3.has_edge(city1, city2) == 0:
        G3.add_edge(city1, city2, weight = wt)

pos = nx.circular_layout(G3)
nx.draw(G3, pos, with_labels=1)
nx.draw_networkx_edge_labels(G3, pos)
plt.show()
```



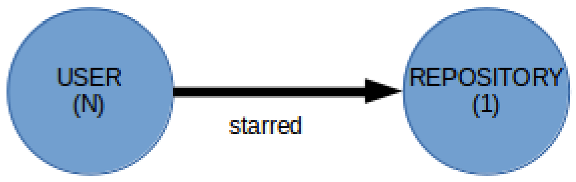
In [23]:

```
# Getting the shortest path
print nx.dijkstra_path(G3, 'Mumbai', 'Tuticorin')
```

```
['Mumbai', 'Hyderabad', 'Madurai', 'Ropar', 'Tuticorin']
```

3.3. Generating Interest Graph

The interest graph is generated by making connections to the object which people are interested in. Here objects are the repositories. As I already stated let us build a star graph by making edges to the repository "AwesomeDataScience" with the people who starred it. Its schema is displayed below.



Graph Schema 1

In [1]:

```
# Finding the people who followed the "awesome-datascience" repository
from github import Github

ACCESS_TOKEN = 'cfda794b26e8ab9ec9c28fca61cce08c76417d7a'
gh = Github(ACCESS_TOKEN, per_page=100)
user = gh.get_user('bulutyazilim')
repo = user.get_repo('awesome-datascience')

stargazers = [stargazer for stargazer in repo.get_stargazers()]
print "Number of stargazers :", len(stargazers)
```

```
Number of stargazers : 4747
```

In [2]:

```
print stargazers[:15]
```

```
[NamedUser(login="sakkas45"), NamedUser(login="ogta"), NamedUser(login
="necatikartal"), NamedUser(login="gwn"), NamedUser(login="musaulke
r"), NamedUser(login="bahattincinic"), NamedUser(login="mahammad"), Na
medUser(login="kurtulusahmet"), NamedUser(login="onuryurtturk"), Named
User(login="nicolasramy"), NamedUser(login="GarethLewin"), NamedUser(l
ogin="qs"), NamedUser(login="Jberlinsky"), NamedUser(login="jt6211"),
 NamedUser(login="erolrecep")]
```

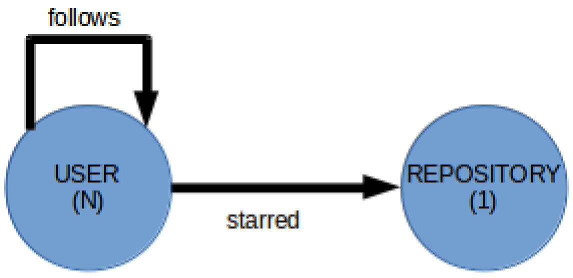
In [3]:

```
# Creating an interest graph with stargazers
import networkx as nx

G = nx.DiGraph()
G.add_node(repo.name, type='repo', lang=repo.language, owner=user.login)

for stargazer in stargazers:
    G.add_node(stargazer.login, type='user')
    G.add_edge(stargazer.login, repo.name , type='starred')

print nx.info(G)
```

```
Name:
Type: DiGraph
Number of nodes: 4748
Number of edges: 4747
Average in degree:   0.9998
Average out degree:   0.9998
```

3.4. Adding Social Links

The connections between the people in the graph can be established by extracting the followers. If a person follows another person within this group, we add an edge.(Similar to a random walk) The schema for this graph is:



Graph Schema 2

In [4]:

```
for i, stargazer in enumerate(stargazers[:500]):
    try:
        for follower in stargazer.get_followers():
            if follower.login in G:
                G.add_node(follower.login, type='user')
                G.add_edge(follower.login, stargazer.login, type='follows')
    except Exception:
        print "Error in getting the followers of", stargazer.login

    print "Processed", i+1, " stargazers"
    print "Number of nodes and edges in the graph",G.number_of_nodes(), "and", G.nu
```

```
Number of nodes and edges in the graph 4748 and 4997
Processed 38  stargazers
Number of nodes and edges in the graph 4748 and 4998
Processed 39  stargazers
Number of nodes and edges in the graph 4748 and 4998
Processed 40  stargazers
Number of nodes and edges in the graph 4748 and 5000
Processed 41  stargazers
Number of nodes and edges in the graph 4748 and 5001
Processed 42  stargazers
Number of nodes and edges in the graph 4748 and 5002
Processed 43  stargazers
Number of nodes and edges in the graph 4748 and 5003
Processed 44  stargazers
Number of nodes and edges in the graph 4748 and 5010
Processed 45  stargazers
Number of nodes and edges in the graph 4748 and 5011
Error in getting the followers of ktaranov
Processed 46  stargazers
Number of nodes and edges in the graph 4748 and 5011
```

```
Note: As it is a very time consuming process, I have limited the number of
stargazers to process.
```

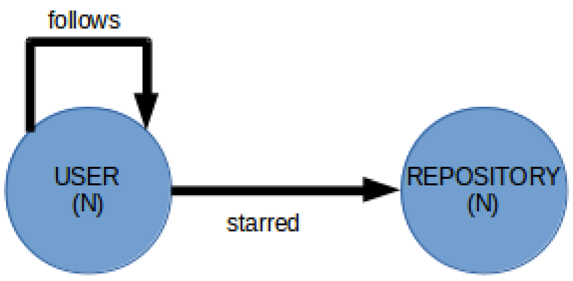
In [5]:

```
print nx.info(G)
```

```
Name:
Type: DiGraph
Number of nodes: 4748
Number of edges: 6486
Average in degree:   1.3660
Average out degree:   1.3660
```

3.5. Adding More Repositories

We are going to analyze the behavior of the people in this network.(What are the non data science languages that are popular among the stargazers of "awesome data science" repository?) For that, let us add information about the repositories that people in this group follow. This will increase the nodes and edges in the graph.



Graph Schema 3

In [6]:

```
print G.nodes()
```

```
u'gianlucadonato', u'DocSavage', u'beginnerSC', u'suryanarayadev',
 u'andrewviren', u'mimanshujain', u'tom031', u'vimalanandans', u'Cam
iloMorales', u'psukys', u'shencoop', u'myberg', u'yujingma45', u'lau
razh', u'qutus', u'dccooper', u'FeynmanHung', u'dmarkos', u'kujon',
 u'rgalbo', u'mightymarc', u'stczztop', u'sld', u'gcameo', u'cadre
v', u'fcschmidt', u'Yikun', u'gwulfs', u'maman', u'TheikChan', u'nur

itnt', u'giranntu', u'omoju', u'antaraB', u'GarethLewin', u'snorik',
u'snowprint', u'iamnut', u'MatejLach', u'heluvaguy', u'BrightFeathe
r', u'RahadianArthapati', u'thanosgatos', u'hoker', u'kianho', u'sru
thikesh-MU', u'adampush', u'DulajChathuranga', u'Acen1991', u'artica
l', u'brunohenrique', u'askming', u'henfiber', u'bittergrass', u'haj
imezhao', u'mjgil', u'jrminz', u'adyomin', u'RichardLitt', u'VibhaBe
lavadi', u'shuaiyuancn', u'enochko', u'khajavi', u'halbtuerke', u'fe
derivo', u'ledezhu', u'Awesomecase', u'aalhour', u'JackJonesIndian
a', u'ClementParis016', u'avy-the-one', u'johanherman', u's4kibs4m
i', u'ooooogllllleoooooooooooooo', u'manishdwibedy', u'ramirobente
s', u'AndreiTuta', u'aishangzoulu', u'krcgk', u'JosmanPS', u'giangst
rider', u'jeffm852', u'subratrout', u'bladereo', u'codegritty', u'Ma
rioCatuogno', u'covertfunction', u'rahul2u', u'YabinFan', u'halojett
```

In [7]:

```
for i, stargazer in enumerate(stargazers[:200]):
    try:
        for starred in stargazer.get_starred()[:200]: #Limiting
            if starred.name not in G.nodes():
                G.add_node(starred.name, type='repo', lang=starred.language, owner=
            G.add_edge(stargazer.login, starred.name, type='gazes')
    except Exception:
        print "Error in getting the starred repositories of", stargazer.login

    print "Processed", i+1, " stargazers"
    print "Number of nodes and edges in the graph",G.number_of_nodes(), "and", G.nu
    print "-"*100
```

```
Number of nodes and edges in the graph 5838 and 7638
--------------------------------------------------------------------
--------------------------------
Processed 8  stargazers
Number of nodes and edges in the graph 6022 and 7838
--------------------------------------------------------------------
--------------------------------
Processed 9  stargazers
Number of nodes and edges in the graph 6202 and 8037
--------------------------------------------------------------------
--------------------------------
Processed 10  stargazers
Number of nodes and edges in the graph 6382 and 8237
--------------------------------------------------------------------
--------------------------------
Processed 11  stargazers
Number of nodes and edges in the graph 6497 and 8370
--------------------------------------------------------------------
--------------------------------
Processed 12  stargazers
```

```
Note: As it is a very time consuming process, I have limited the number of
stargazers to process.
```

In [8]:

```
print(nx.info(G))
```

```
Name:
Type: DiGraph
Number of nodes: 23768
Number of edges: 35843
Average in degree:   1.5080
Average out degree:   1.5080
```

3.6. Snapshoting The Graph

Snapshoting the graph can help us to reuse it and we don't need to make API calls through PyGitHub everytime.

It can be done either by saving the graph as graphical data formats like graphml, gml, gexf or by pickling. Pickling is the process of deserializing the python object into a byte stream, which can be deserialized to retrieve the graph.

In [11]:

```
# Pickling
nx.write_gpickle(G, "github_1.pickle")
# G1 = nx.read_gpickle("github_1.pickle")

# Exercise: saving as graphml (Modify the Node type with 'None' as it cannot be sav
# nx.write_gml(G, "github_1.gml")
# G1 = nx.read_gml("github_1.gml")
```

In [10]:

```
import networkx as nx
G1 = nx.read_gpickle("github_1.pickle")
print nx.info(G1)
```

```
Name:
Type: DiGraph
Number of nodes: 23768
Number of edges: 35843
Average in degree:   1.5080
Average out degree:   1.5080
```

```
The snapshot dataset is available at <a href="">this</a> link.
```

4. Graph Exploration

4.1. Popular Repositories

Idea: The repository nodes with many high indegree are the popular ones. As we started with the 'awesome-datascience' repository and added only some other repositories, there is a great probability that this repository top this list. Let us find!

In [12]:

```
print "Most popular repositories"
repos_degree = [(n, G1.in_degree(n)) for n in G1.nodes() if G1.node[n]['type'] == '
sorted_repos_degree = sorted(repos_degree, key = lambda x:x[1], reverse=True)
print sorted_repos_degree[:20]
```

```
Most popular repositories
[(u'awesome-datascience', 4747), (u'awesome-machine-learning', 34),
 (u'go', 34), (u'every-programmer-should-know', 34), (u'awesome', 33),
(u'tensorflow', 33), (u'dotfiles', 32), (u'coding-interview-universit
y', 29), (u'awesome-python', 28), (u'free-programming-books', 27),
 (u'system-design-primer', 27), (u'awesome-awesomeness', 25), (u'aweso
me-bigdata', 23), (u'papers-we-love', 22), (u'awesome-public-dataset
s', 22), (u'the-art-of-command-line', 21), (u'machine-learning-for-sof
tware-engineers', 21), (u'public-apis', 21), (u'models', 20), (u'thefu
ck', 20)]
```

4.2. Popular Languages

Idea: The weights of the repository nodes can be used to find the popular languages. The repository node attribute 'lang' is used and the in degree of that node is used for adding weight. (Each in-edge incerments the weight by one)

In [13]:

```
languages_score = {}
languages_score = {G1.node[n]['lang']:0 for n in G1.nodes() \
                   if G1.node[n]['type'] == 'repo' and G1.node[n]['lang'] not in la
print languages_score
```

```
{u'TypeScript': 0, u'Objective-C++': 0, u'Lean': 0, u'Fortran': 0, u'J
upyter Notebook': 0, u'Groff': 0, u'Shell': 0, u'Web Ontology Languag
e': 0, u'AppleScript': 0, u'Elm': 0, u'SuperCollider': 0, u'Nginx': 0,
u'Elixir': 0, u'PostScript': 0, u'Thrift': 0, u'Logos': 0, u'D': 0,
 u'Kotlin': 0, u'Crystal': 0, u'Objective-C': 0, u'Batchfile': 0, u'Ob
jective-J': 0, u'Roff': 0, u'FreeMarker': 0, u'Swift': 0, u'Smarty':
 0, u'Arc': 0, u'Go': 0, u'Visual Basic': 0, u'PHP': 0, u'Coq': 0, u'J
ava': 0, u'Scala': 0, u'OpenSCAD': 0, u'ApacheConf': 0, u'Makefile':
 0, u'Perl': 0, u'Lua': 0, u'GDScript': 0, u'Verilog': 0, u'Haxe': 0,
 u'Red': 0, u'WebAssembly': 0, u'CoffeeScript': 0, u'HTML': 0, u'Awk':
0, u'Lex': 0, u'Idris': 0, u'Ruby': 0, u'PLpgSQL': 0, u'C': 0, u'AutoH
otkey': 0, u'P4': 0, u'GCC Machine Description': 0, u'Clojure': 0, u'H
CL': 0, u'Prolog': 0, u'CMake': 0, u'Tcl': 0, u'Xtend': 0, u'PigLati
n': 0, u'GLSL': 0, u'VHDL': 0, u'Arduino': 0, u'IDL': 0, u'SQLPL': 0,
 u'ASP': 0, u'Assembly': 0, None: 0, u'C#': 0, u'Vala': 0, u'Processin
g': 0, u'NSIS': 0, u'CSS': 0, u'LiveScript': 0, u'Stan': 0, u'SaltStac
k': 0, u'Protocol Buffer': 0, u'QML': 0, u'Mathematica': 0, u'XSLT':
 0, u'Mask': 0, u'WebIDL': 0, u'TeX': 0, u'R': 0, u'Cuda': 0, u'Vim sc
ript': 0, u'Pony': 0, u'Scilab': 0, u'Smali': 0, u'OpenEdge ABL': 0,
 u'Perl6': 0, u'M4': 0, u'OCaml': 0, u'Gherkin': 0, u'Pascal': 0, u'F
#': 0, u'Puppet': 0, u'ActionScript': 0, u'Emacs Lisp': 0, u'JavaScrip
t': 0, u'VimL': 0, u'Rust': 0, u'Matlab': 0, u'Erlang': 0, u'Eagle':
 0, u'Scheme': 0, u'PLSQL': 0, u'Nim': 0, u'Python': 0, u'Common Lis
p': 0, u'Dart': 0, u'XQuery': 0, u'Nimrod': 0, u'SystemVerilog': 0,
 u'Groovy': 0, u'Vue': 0, u'M': 0, u'C++': 0, u'FORTRAN': 0, u'ShaderL
ab': 0, u'MoonScript': 0, u'Standard ML': 0, u'PureScript': 0, u'Juli
a': 0, u'PowerShell': 0, u'Bro': 0, u'Haskell': 0, u'Racket': 0, u'AGS
Script': 0, u'nesC': 0}
```

In [14]:

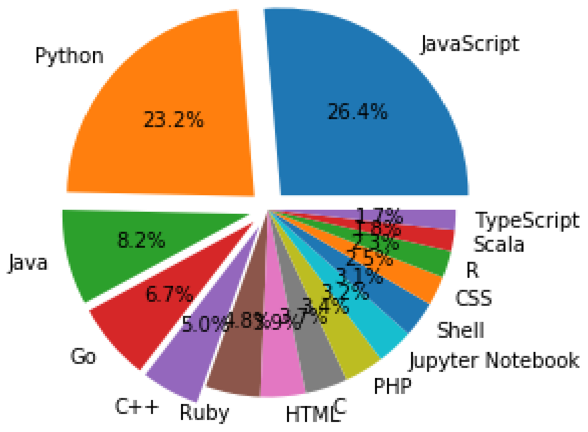
```
for n in G1.nodes():
    if G1.node[n]['type'] == 'repo':
        languages_score[G1.node[n]['lang']] += G1.in_degree(n)

languages_score_sorted = sorted(languages_score.items(), key = lambda x:x[1], rever
print languages_score_sorted[1:20] #The 0th item is the None (For the repos with un
```

```
[(u'JavaScript', 6170), (u'Python', 5416), (u'Java', 1920), (u'Go', 15
69), (u'C++', 1169), (u'Ruby', 1110), (u'HTML', 910), (u'C', 853),
 (u'PHP', 795), (u'Jupyter Notebook', 744), (u'Shell', 727), (u'CSS',
 593), (u'R', 540), (u'Scala', 423), (u'TypeScript', 405), (u'C#', 31
8), (u'Swift', 294), (u'Objective-C', 283), (u'Lua', 189)]
```

Visualizing the Top 15 languages of data science learners

In [79]:

```
import matplotlib.pyplot as plt

lang = [x[0] for x in languages_score_sorted[1:16]]
freq = [x[1] for x in languages_score_sorted[1:16]]
explode = [0]*len(languages_score_sorted[1:16])
for i in range(5):# only "explode" the first 5 slices
    explode[i] = 0.1

fig1, ax1 = plt.subplots()
ax1.pie(freq, explode=explode, labels=lang, autopct='%1.1f%%')
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



4.3. Connectivity

A directed graph is said to be **weakly** connected if there is a **path** between every pair of nodes in a graph.
A directed graph is said to be **strongly** connected if there is a **directed path** between every pair of nodes in a graph.

In [33]:

```
print nx.is_weakly_connected(G1)
print len(list(nx.weakly_connected_component_subgraphs(G1)))
print nx.is_strongly_connected(G1)
```

```
True
1
False
```

```
## 4.4. Popular Users
Idea: Centrality measures identify the important nodes in a network. Each
centrality measure quantifies the importance in a different way

As I have processed only a few stargazers in the network, let us make subgraph
with only the user nodes that have degree > 1.
```

In [15]:

```
user_nodes = [node for node in G1.nodes() if G1.node[node]['type'] == 'user' and G1
H = G1.subgraph(user_nodes)
print nx.info(H)
```

```
Name: 
Type: SubDiGraph
Number of nodes: 813
Number of edges: 1308
Average in degree:   1.6089
Average out degree:   1.6089
```

4.4.1. Degree Centrality

The assumption of important nodes have many connections is made here.

In [16]:

```
dc_nodes = nx.degree_centrality(H)
dc_nodes_sorted = sorted(dc_nodes.items(), key = lambda x:x[1], reverse=True)
print dc_nodes_sorted[:10]
```

```
[(u'angusshire', 0.26108374384236455), (u'josephmisiti', 0.13423645320
197045), (u'clarecorthell', 0.08374384236453201), (u'sdogruyol', 0.064
03940886699508), (u'ibrahimgunduz34', 0.04926108374384237), (u'erolrec
ep', 0.04926108374384237), (u'jonathan-bower', 0.04926108374384237),
 (u'hmert', 0.04926108374384237), (u'RichardLitt', 0.0418719211822660
1), (u'Ardakilic', 0.04187192118226601)]
```

4.4.2. Betweeness Centrality

The assumption of important nodes have connect other nodes is made here.

In [17]:

```
bc_nodes = nx.betweenness_centrality(H)
bc_nodes_sorted = sorted(bc_nodes.items(), key = lambda x:x[1], reverse=True)
print bc_nodes_sorted[:10]
```

```
[(u'bahattincinic', 0.003715609895757557), (u'sly01', 0.00279984750549
58256), (u'muraty', 0.0025828963716568246), (u'sdogruyol', 0.002299581
0244569605), (u'erolrecep', 0.0018606369882010257), (u'smtaydemir', 0.
0018147539359682438), (u'kurtulusahmet', 0.001704583554725233), (u'hal
idaltuner', 0.0016112726536821346), (u'ibrahimgunduz34', 0.00152195553
34538362), (u'Ardakilic', 0.0012694680524224118)]
```

4.4.3. Closeness Centrality

The assumption of important nodes close to each other is made here.

In [18]:

```
cc_nodes = nx.closeness_centrality(H)
cc_nodes_sorted = sorted(cc_nodes.items(), key = lambda x:x[1], reverse=True)
print cc_nodes_sorted[:10]
```

```
[(u'josephmisiti', 0.1602312119553499), (u'ionelmc', 0.115931034482758
63), (u'mher', 0.09075620234085945), (u'clarecorthell', 0.084772881175
8926), (u'sdogruyol', 0.0828735051959129), (u'bahattincinic', 0.071503
66454403459), (u'ibrahimgunduz34', 0.07085559507686813), (u'Ardakili
c', 0.07064217460976913), (u'smtaydemir', 0.0677838207238247), (u'hkul
ekci', 0.06407978680449002)]
```

4.5. Community Detection

Community in a social network is the sub network with more intra connectivity and less inter connectivty with other communities.
Girvan Newman algorithm is used to detect the communities in a network. It is based on *edge betweenness*. The edges which are connecting two communities tend to have high betweenness. The algorithms detect communities by repeatedly removing the edges with high betweenness.

Let us use this community detection concept to form more than 200 groups.

In [46]:

```
def edge_to_remove(G):
    eb_cent = nx.edge_betweenness_centrality(G) #Key-edges as tuple ;Value-edge bet
    eb_cent_items = eb_cent.items()#list has tuples, tuple has the above data
    eb_cent_items.sort(key = lambda x:x[1], reverse = True) #Descending order
    return eb_cent_items[0][0] #(a, b) #edge with max edge betweenness

def girvan(G):
    Gc = G.copy()
    # Returns connected components as subgrahs
    components = list(nx.weakly_connected_component_subgraphs(Gc))
    n_components = len(components)
    print 'The number of connected components are ', n_components
    while n_components < 200:
        Gc.remove_edge(*edge_to_remove(Gc)) # ((a, b)) -> (a, b)
        components = list(nx.weakly_connected_component_subgraphs(Gc))
        n_components = len(components)
        print 'The number of connected components are ', n_components
    return components
```

In [47]:

```
communities = girvan(H)
for community in communities: #i is a graph
    print community.nodes()
    print "Number of users : ", community.number_of_nodes()
    print "-"*100
```

```
mn1k', u'nmurthy', u'FreddieFO', u'MadFiend', u'arunslb123', u'RayOf
Creation', u'wohfab', u'AnalogJ', u'juandecarrion', u'Ardakilic',
 u'harrisony', u'mcolak', u'Matthimatiker', u'faruktemur', u'mehmetd
ik', u'The-Gupta', u'blvsaditya', u'eneskemalergin', u'hamdikavak',
 u'Roon', u'mmngreco', u'tralphy', u'bahattincinic', u'augustopedr
o', u'arrmac', u'dafukua', u'nabilov', u'glenux', u'perfettiful',
 u'Baltazore', u'vikramlance', u'rakhmad', u'onurdegerli', u'bruno7
8', u'torosgo', u'HGebhardt', u'AakashRaina', u'vkoul', u'lbali',
 u'tubanaz', u'setkyar', u'seyyah', u'scrivna', u'laranea', u'Mustaf
aFerhan', u'Jeremiearnaud', u'rishikksh20', u'juanshishido', u'Victo
rTomaili', u'arthuralvim', u'onlyice', u'unicod3', u'ercanpinar',
 u'KarthiAru', u'jonathan-bower', u'inferjay', u'fangcode', u'sluolc
t', u'onuryurtturk', u'KatherineMichel', u'fly51fly', u'ghaseminya',
u'knightelvis', u'di1eep', u'vdt', u'fabiocerqueira', u'vinta', u'ph
reakinggeek', u'Motun', u'halil', u'gauravssnl', u'aburan28', u'will
emdh', u'HQarroum', u'VikramTiwari', u'tscholak', u'EugeneBakin',

 u'great-thoughts', u'darul75', u'osonwanne', u'NDuma', u'Merker56',
u'tncyysl', u'4aficiona2', u'ThomasGHenry', u'data-catalysis', u'xpg
eng', u'mburakergenc', u'pugang', u'Burakhan', u'trivektor', u'ysh32
```

In [64]:

```
from collections import Counter
sizes = [community.number_of_nodes() for community in communities]
for (size, count) in Counter(sizes).items():
    print "{} group(s) have {} user(s)".format(count, size)
```

```
196 group(s) have 1 user(s)
1 group(s) have 610 user(s)
1 group(s) have 3 user(s)
2 group(s) have 2 user(s)
```

As the data we used is a small sample and because of the process limiting due to computational dependencies, many of these 200 groups has only 1 user. This may not be the case when we work with the entire data.

4.6 XXX

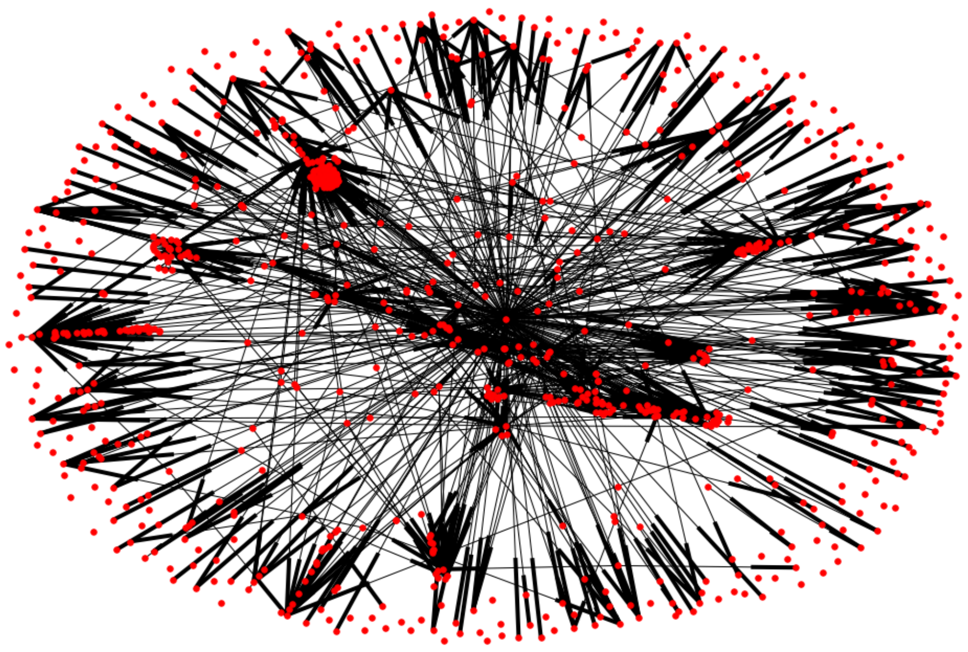
This space is for you. Analyze and write on the comment section. Enjoy exploring!

Bonus: Graph Visualization

NetworkX provides basic functionality for visualizing graphs. I highly recommend you to explore the fully featured graph visualization tools like Cytospace, Gephi and GraphViz.To use other tools, export the graph into the suitable format(like we did in snapshotting the graph) and visualize it.

In [56]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
nx.draw(H, with_labels=False, node_size=30)
plt.show()
```



End Notes

So this is all about collecting data and deriving insights from the GitHub social network. This is just a sample application and we can do a lot of exploration from this data. If you feel that I have missed any important part, please feel free to comment it.

Yes, you've learned a new thing today. It is the time to give feedback. Enjoy learning! :)