Analysis of Political Blogs

Aaron Palumbo | Partha Banerjee
Sept. 27, 2015

Assignment:

Week 4 Assignment

Centrality measures can be used to predict (positive or negative) outcomes for a node.

Your task in this week's assignment is to identify an interesting set of network data that is available on the web (either through web scraping or web APIs) that could be used for analyzing and comparing centrality measures across nodes. As an additional constraint, there should be at least one categorical variable available for each node (such as "Male" or "Female"; "Republican", "Democrat," or "Undecided", etc.)

In addition to identifying your data source, you should create a high level plan that describes how you would load the data for analysis, and describe a hypothetical outcome that could be predicted from comparing degree centrality across categorical groups.

For this week's assignment, you are not required to actually load or analyze the data. Please see also Project 1 below.

You may work in a small group on the assignment. You should post your document to GitHub by end of day on Sunday September 20th.

Data

Political blogs: A directed network of hyperlinks between weblogs on US politics, recorded in 2005 by Adamic and Glance. Please cite L. A. Adamic and N. Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005). Thanks to Lada Adamic for permission to post these data on this web site.

From http://www-personal.umich.edu/~mejn/netdata/ (http://www-personal.umich.edu/~mejn/netdata/)

From the data dictionary:

Political blogosphere Feb. 2005 Data compiled by Lada Adamic and Natalie Glance

Node "value" attributes indicate political leaning according to:

0 (left or liberal) 1 (right or conservative)

Data on political leaning comes from blog directories as indicated. Some blogs were labeled manually, based on incoming and outgoing links and posts around the time of the 2004 presidential election. Directory-derived labels are prone to error; manual labels even more so.

Links between blogs were automatically extracted from a crawl of the front page of the blog.

These data should be cited as Lada A. Adamic and Natalie Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005).

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import IPython.display as dis
import numpy as np
import pandas as pd

%matplotlib inline
```

In [2]:
```
dg = nx.read_gml("polblogs/polblogs.gml")

# We're going to turn this into an undirected graph
g = dg.to_undirected()
```

Analysis

First we check to see how large our graph is.

In [3]:
```
len(g)
```

Out[3]: 1490

1490 isn't too big. We can go ahead and calculate the in-degree measure of centrality on each node and take a look at the top 10.

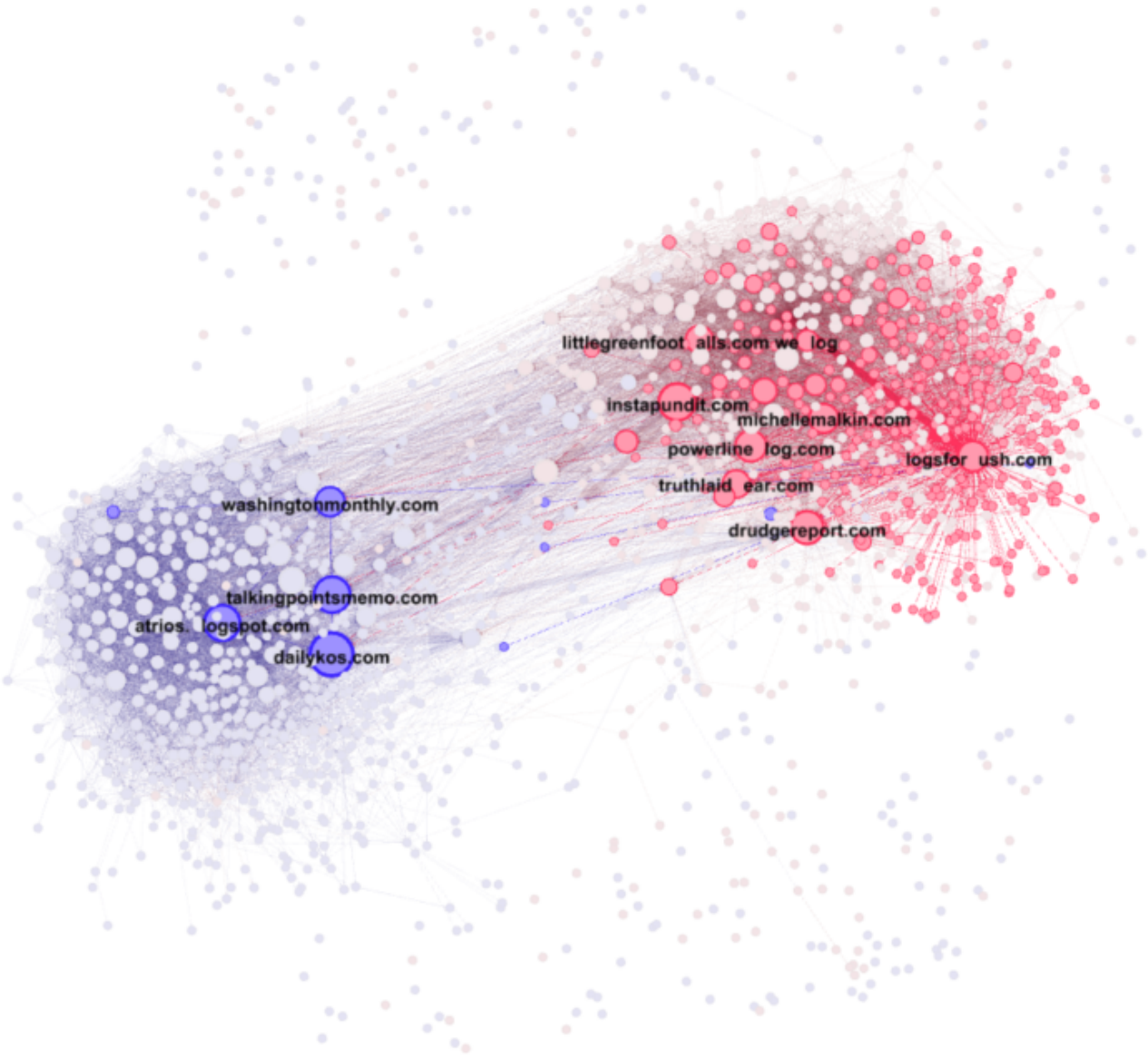
In [4]:
```
d = nx.degree(g)

def sorted_map(map):
    ms = sorted(map.iteritems(), key=lambda (k, v): (-v, k))
    return ms

ds = sorted_map(d)

k=0
for (i, j) in ds[0:9]:
    print g.node[i], ds[k]
    k+=1
```

```
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 155, 'value': 0,
'label': u'dailykos.com'} (155, 352)
{'source': u'BlogPulse', 'id': 1051, 'value': 1, 'label': u'instapundit.com'} (10
51, 307)
{'source': u'BlogPulse,CampaignLine', 'id': 855, 'value': 1, 'label': u'blogsforb
ush.com'} (855, 302)
{'source': u'BlogPulse,LeftyDirectory,CampaignLine', 'id': 55, 'value': 0, 'labe
l': u'atrios.blogspot.com'} (55, 278)
{'source': u'BlogPulse,LeftyDirectory', 'id': 641, 'value': 0, 'label': u'talking
pointsmemo.com'} (641, 275)
{'source': u'LabeledManually', 'id': 963, 'value': 1, 'label': u'drudgereport.co
m'} (963, 245)
{'source': u'BlogPulse', 'id': 1245, 'value': 1, 'label': u'powerlineblog.com'}
(1245, 224)
{'source': u'BlogPulse,LeftyDirectory', 'id': 729, 'value': 0, 'label': u'washing
tonmonthly.com'} (729, 218)
{'source': u'BlogPulse', 'id': 1153, 'value': 1, 'label': u'michellemalkin.com'}
(1153, 212)
```

We can also load this dataset into Gephi, which is an easy way to visualize what we have.

In [5]:
```
dis.Image('in_degree.png')
```

Out[5]:



We can see in the graph above that there appears to be one large component with two clusters, and many islands. Let's remove those islands before we move on.

ref: http://stackoverflow.com/questions/20012579/is-there-an-easy-way-to-prune-disconnected-networks-in-a-networkx-graph (http://stackoverflow.com/questions/20012579/is-there-an-easy-way-to-prune-disconnected-networks-in-a-networkx-graph)

In [6]:
```
def prune_islands(g):
    if not nx.is_connected(g):
        # get a list of unconnected networks
        sub_graphs = list(nx.connected_component_subgraphs(g))

        largest_sub = sub_graphs[0]

        # find the largest network in the list
        for sg in sub_graphs:
            if len(sg.nodes()) > len(largest_sub.nodes()):
                largest_sub = sg

        g = largest_sub
    return g
```

In [7]:
```
### This function returns a sorted degree list -- taken from our text book
def sorted_map(map):
    ms = sorted(map.iteritems(), key=lambda (k,v): (-v,k))
    return ms

# This function returns a new graph object that contains the network with pendant
and isolated nodes removed
def prn(g, l, mx=9):
    k=0
    for (i, j) in l[0:(mx-1)]:
        print g.node[i], l[k]
        k+=1
```

In [8]:
```
g = prune_islands(g)
d = nx.degree(g)
ds = sorted_map(d)

prn(g,ds,10)
```

```
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 155, 'value': 0,
'label': u'dailykos.com'} (155, 352)
{'source': u'BlogPulse', 'id': 1051, 'value': 1, 'label': u'instapundit.com'} (10
51, 307)
{'source': u'BlogPulse,CampaignLine', 'id': 855, 'value': 1, 'label': u'blogsforb
ush.com'} (855, 302)
{'source': u'BlogPulse,LeftyDirectory,CampaignLine', 'id': 55, 'value': 0, 'labe
l': u'atrios.blogspot.com'} (55, 278)
{'source': u'BlogPulse,LeftyDirectory', 'id': 641, 'value': 0, 'label': u'talking
pointsmemo.com'} (641, 275)
{'source': u'LabeledManually', 'id': 963, 'value': 1, 'label': u'drudgereport.co
m'} (963, 245)
{'source': u'BlogPulse', 'id': 1245, 'value': 1, 'label': u'powerlineblog.com'}
(1245, 224)
{'source': u'BlogPulse,LeftyDirectory', 'id': 729, 'value': 0, 'label': u'washing
tonmonthly.com'} (729, 218)
{'source': u'BlogPulse', 'id': 1153, 'value': 1, 'label': u'michellemalkin.com'}
(1153, 212)
```

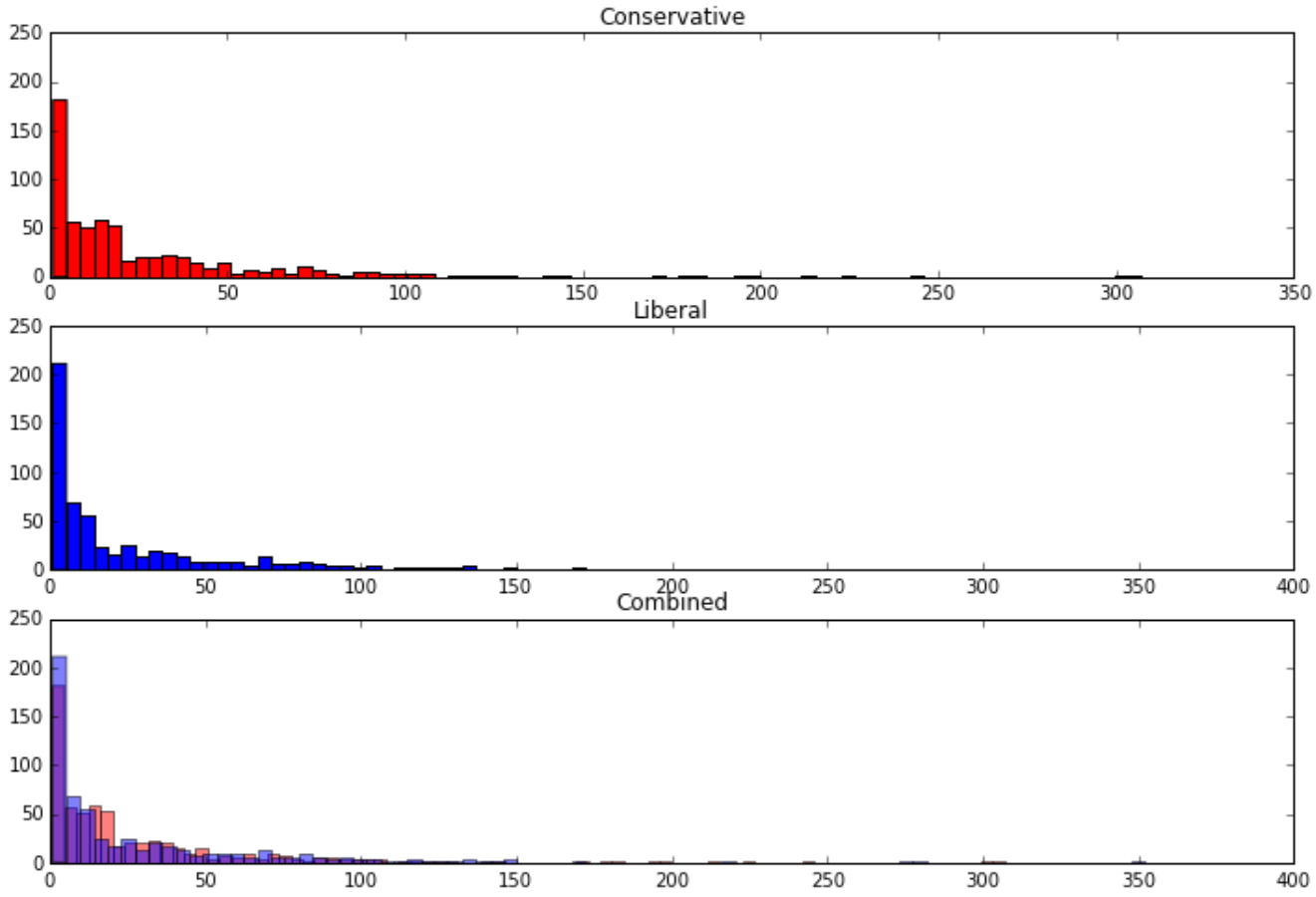
We don't really follow the blogging scene, but some Internet searches seem to indicate these are very popular blogs - except for blogsforbush.com, which we suspect was popular at the time of the data collection but now seems defunct.

Now we want to compare the centrality measure between left and right leaning blogs:

1 = conservative (red)
0 = liberal (blue)

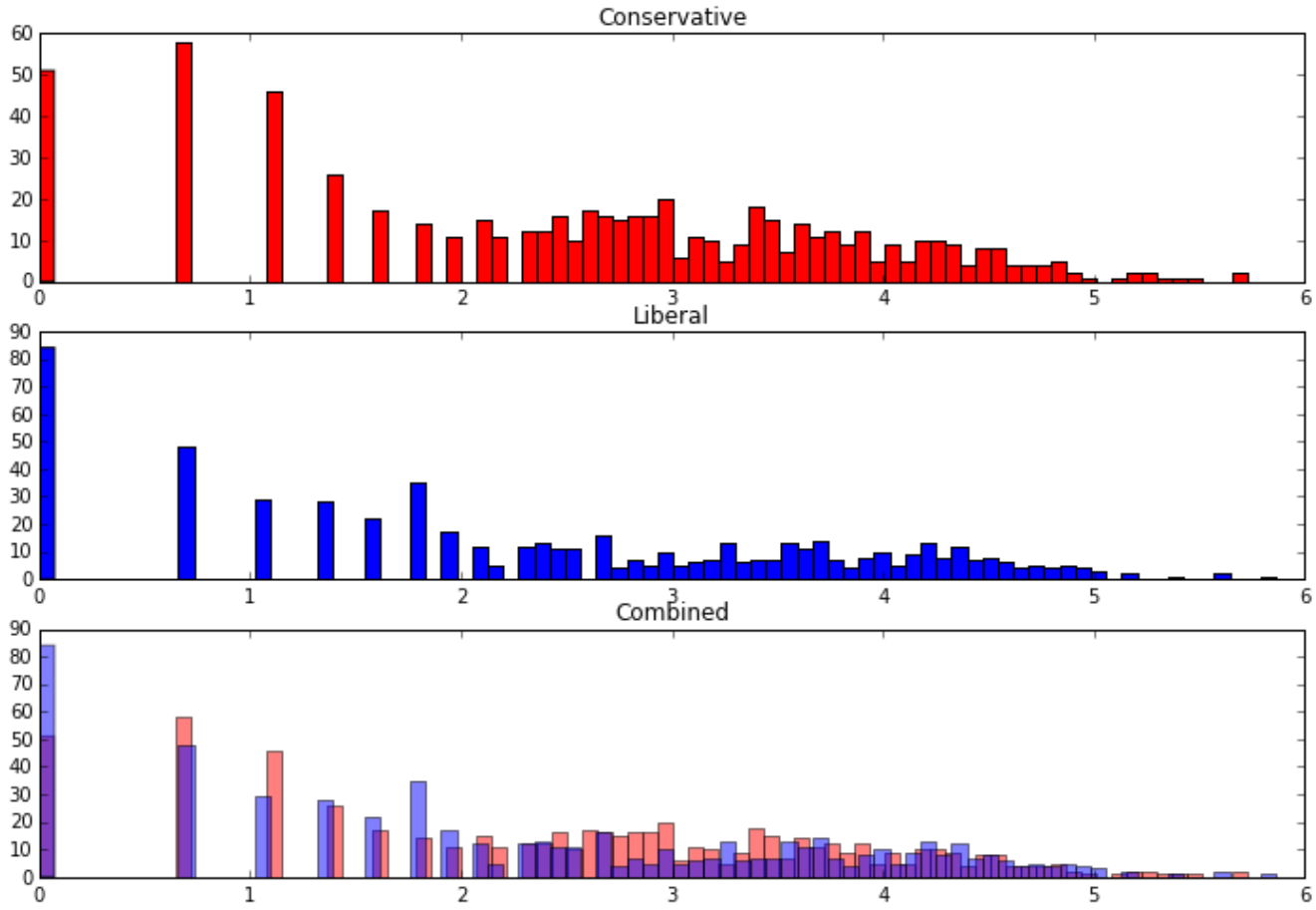
To do this we start by comparing a histogram of the two groups.

In [9]:
```
b = 80
maxy = 250
fig, ax = plt.subplots(3, 1, figsize=(12, 8))
ax[0].hist([j for (i, j) in ds if g.node[i]['value'] == 1],
           bins=b, color='red')
ax[0].set_ylim([0, maxy])
ax[0].set_title("Conservative")

ax[1].hist([j for (i, j) in ds if g.node[i]['value'] == 0],
           bins=b, color='blue')
ax[1].set_ylim([0, maxy])
ax[1].set_title("Liberal")

ax[2].hist([j for (i, j) in ds if g.node[i]['value'] == 1],
           bins=b, color='red', alpha=0.5)
ax[2].hist([j for (i, j) in ds if g.node[i]['value'] == 0],
           bins=b, color='blue', alpha=0.5)
ax[2].set_ylim([0, maxy])
ax[2].set_title("Combined")

plt.show()
```



The assignment suggests comparing the centrality of the two distributions with possibly a student t-test. We have to admit that we are not sure how to do this with the data we have. These are clearly not normally distributed. We can check to see if maybe the are log normal ...

In [10]:
```
b = 80
fig, ax = plt.subplots(3, 1, figsize=(12, 8))
ax[0].hist([np.log(j) for (i, j) in ds if g.node[i]['value'] == 1],
           bins=b, color='red')
# ax[0].set_ylim([0, 400])
ax[0].set_title("Conservative")

ax[1].hist([np.log(j) for (i, j) in ds if g.node[i]['value'] == 0],
           bins=b, color='blue')
# ax[1].set_ylim([0, 400])
ax[1].set_title("Liberal")

ax[2].hist([np.log(j) for (i, j) in ds if g.node[i]['value'] == 1],
           bins=b, color='red', alpha=0.5)
ax[2].hist([np.log(j) for (i, j) in ds if g.node[i]['value'] == 0],
           bins=b, color='blue', alpha=0.5)
# ax[2].set_ylim([0, 400])
ax[2].set_title("Combined")

plt.show()
```



These don't really look log normal either. We could calculate a mean and standard deviation of the two distributions, but we think comparing them with a t-test becomes more complicated when they are not normal.

We did a little research into this, but was unable to come to any conclusions.

From a qualitative perspective, they look to be very similar.

Now let us find the elit group in our data set using different central measures.

In [11]:
```
# Let us find the closeness
c=nx.closeness_centrality(g)
cs=sorted_map(c)

# Display top-10
prn(g, cs, 10)
```

```
{'source': u'BlogPulse', 'id': 1051, 'value': 1, 'label': u'instapundit.com'} (10
51, 0.5193534666099532)
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 155, 'value': 0,
'label': u'dailykos.com'} (155, 0.5186915887850467)
{'source': u'BlogPulse,LeftyDirectory', 'id': 641, 'value': 0, 'label': u'talking
pointsmemo.com'} (641, 0.5030902348578492)
{'source': u'BlogPulse,LeftyDirectory,CampaignLine', 'id': 55, 'value': 0, 'labe
l': u'atrios.blogspot.com'} (55, 0.4983673469387755)
{'source': u'LabeledManually', 'id': 1112, 'value': 1, 'label': u'littlegreenfoot
balls.com/weblog'} (1112, 0.4945321992709599)
{'source': u'BlogPulse,LeftyDirectory', 'id': 729, 'value': 0, 'label': u'washing
tonmonthly.com'} (729, 0.4901645925331192)
{'source': u'LabeledManually', 'id': 963, 'value': 1, 'label': u'drudgereport.co
m'} (963, 0.48510131108462456)
{'source': u'BlogPulse', 'id': 1153, 'value': 1, 'label': u'michellemalkin.com'}
(1153, 0.48413957176843775)
{'source': u'LeftyDirectory,LabeledManually', 'id': 1437, 'value': 1, 'label':
u'truthlaidbear.com'} (1437, 0.48108747044917255)
```

In [12]:
```
# Let us now calculate betweenness
b=nx.betweenness_centrality(g)
bs=sorted_map(b)

# Display top-10
prn(g, bs, 10)
```

```
{'source': u'BlogPulse,CampaignLine', 'id': 855, 'value': 1, 'label': u'blogsforb
ush.com'} (855, 0.09800883597157671)
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 155, 'value': 0,
'label': u'dailykos.com'} (155, 0.0883554502217737)
{'source': u'LabeledManually', 'id': 963, 'value': 1, 'label': u'drudgereport.co
m'} (963, 0.06824728427807385)
{'source': u'BlogPulse', 'id': 1051, 'value': 1, 'label': u'instapundit.com'} (10
51, 0.04959607210899005)
{'source': u'BlogPulse,LeftyDirectory', 'id': 641, 'value': 0, 'label': u'talking
pointsmemo.com'} (641, 0.04766945533811751)
{'source': u'LeftyDirectory,LabeledManually', 'id': 1437, 'value': 1, 'label':
u'truthlaidbear.com'} (1437, 0.042235191776918875)
{'source': u'BlogPulse,LeftyDirectory,CampaignLine', 'id': 55, 'value': 0, 'labe
l': u'atrios.blogspot.com'} (55, 0.03419353637453861)
{'source': u'Blogarama', 'id': 979, 'value': 1, 'label': u'etalkinghead.com'} (97
9, 0.028140975470049717)
{'source': u'BlogPulse', 'id': 1153, 'value': 1, 'label': u'michellemalkin.com'}
(1153, 0.027795027550421424)
```

Since our graph is a MultiGraph, we cannot derive eigenvector centrality directly. Let us create a weighted graph for our data.

In [13]:
```
wg = nx.Graph()
for u,v,data in g.edges_iter(data=True):
    if wg.has_edge(u,v):
        wg[u][v]['weight'] += 1
    else:
        wg.add_edge(u, v, weight=1)

# Now calculate eigenvector centrality
e=nx.eigenvector_centrality(wg)
es=sorted_map(e)

# Display top-10
prn(g, es, 10)
```

```
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 155, 'value': 0,
'label': u'dailykos.com'} (155, 0.1638534476041533)
{'source': u'BlogPulse,LeftyDirectory,CampaignLine', 'id': 55, 'value': 0, 'labe
l': u'atrios.blogspot.com'} (55, 0.16007548152992843)
{'source': u'BlogPulse,LeftyDirectory', 'id': 641, 'value': 0, 'label': u'talking
pointsmemo.com'} (641, 0.14909445001496544)
{'source': u'BlogPulse,LeftyDirectory', 'id': 729, 'value': 0, 'label': u'washing
tonmonthly.com'} (729, 0.1383576167605002)
{'source': u'LeftyDirectory,LabeledManually,CampaignLine', 'id': 363, 'value': 0,
'label': u'liberaloasis.com'} (363, 0.11790374467533808)
{'source': u'BlogPulse,LeftyDirectory', 'id': 180, 'value': 0, 'label': u'digbysb
log.blogspot.com'} (180, 0.11663737013363043)
{'source': u'BlogPulse', 'id': 1051, 'value': 1, 'label': u'instapundit.com'} (10
51, 0.11488348941044607)
{'source': u'LeftyDirectory,LabeledManually', 'id': 99, 'value': 0, 'label': u'bo
dyandsoul.typepad.com'} (99, 0.10983547747320058)
{'source': u'BlogPulse,LeftyDirectory,eTalkingHead,CampaignLine', 'id': 642, 'val
ue': 0, 'label': u'talkleft.com'} (642, 0.10751855386054236)
```

It is the time to present our data in a tabular form with all the central measures side-by-side.

In [14]:
```
## make a list of the elite group by merging top ten groups for 3 centrality metr
ics
names1=[x[0] for x in ds[:10]]
names2=[x[0] for x in cs[:10]]
names3=[x[0] for x in bs[:10]]
names4=[x[0] for x in es[:10]]

## use Python sets to compute a union of the sets
names=list(set(names1) | set(names2) | set (names3) | set(names4))

## build a table with centralities
table=[[name,g.node[name]['value'],d[name],c[name],b[name],e[name]] for name in n
ames]
df=pd.DataFrame(table,columns=['ID','Category', 'Degree','Closeness','Betweennes
s','Eigenvector'])

print 'Liberal: ', len(df[df['Category']==0]), ' + ', 'Conservative: ', len(df[d
f['Category']==1]), ' = ', 'Total: ', len(df)
```

```
Liberal:  9  +  Conservative:  8  =  Total:  17
```

So in our sample dataset of top 10 from each central measure, Liberals are narrowly ahead of Conservatives.

In [15]:
```
# Not let us see the data sorted by different measures - starting with Degree
df.sort('Degree', ascending=False)
```

Out[15]:

	ID	Category	Degree	Closeness	Betweenness	Eigenvector
10	155	0	352	0.518692	0.088355	0.163853
8	1051	1	307	0.519353	0.049596	0.114883
5	855	1	302	0.478448	0.098009	0.053419
16	55	0	278	0.498367	0.034194	0.160075
0	641	0	275	0.503090	0.047669	0.149094
2	963	1	245	0.485101	0.068247	0.055585
9	1245	1	224	0.474543	0.023734	0.075798
7	729	0	218	0.490165	0.027744	0.138358
3	1153	1	212	0.484140	0.027795	0.069879
13	1437	1	199	0.481087	0.042235	0.064435
6	1112	1	196	0.494532	0.024184	0.067420
11	363	0	171	0.440476	0.012325	0.117904
1	642	0	148	0.467816	0.012136	0.107519
15	180	0	147	0.458850	0.005039	0.116637
12	493	0	144	0.448567	0.007638	0.106894
14	99	0	133	0.430384	0.002473	0.109835
4	979	1	75	0.429627	0.028141	0.016857

Conclusion

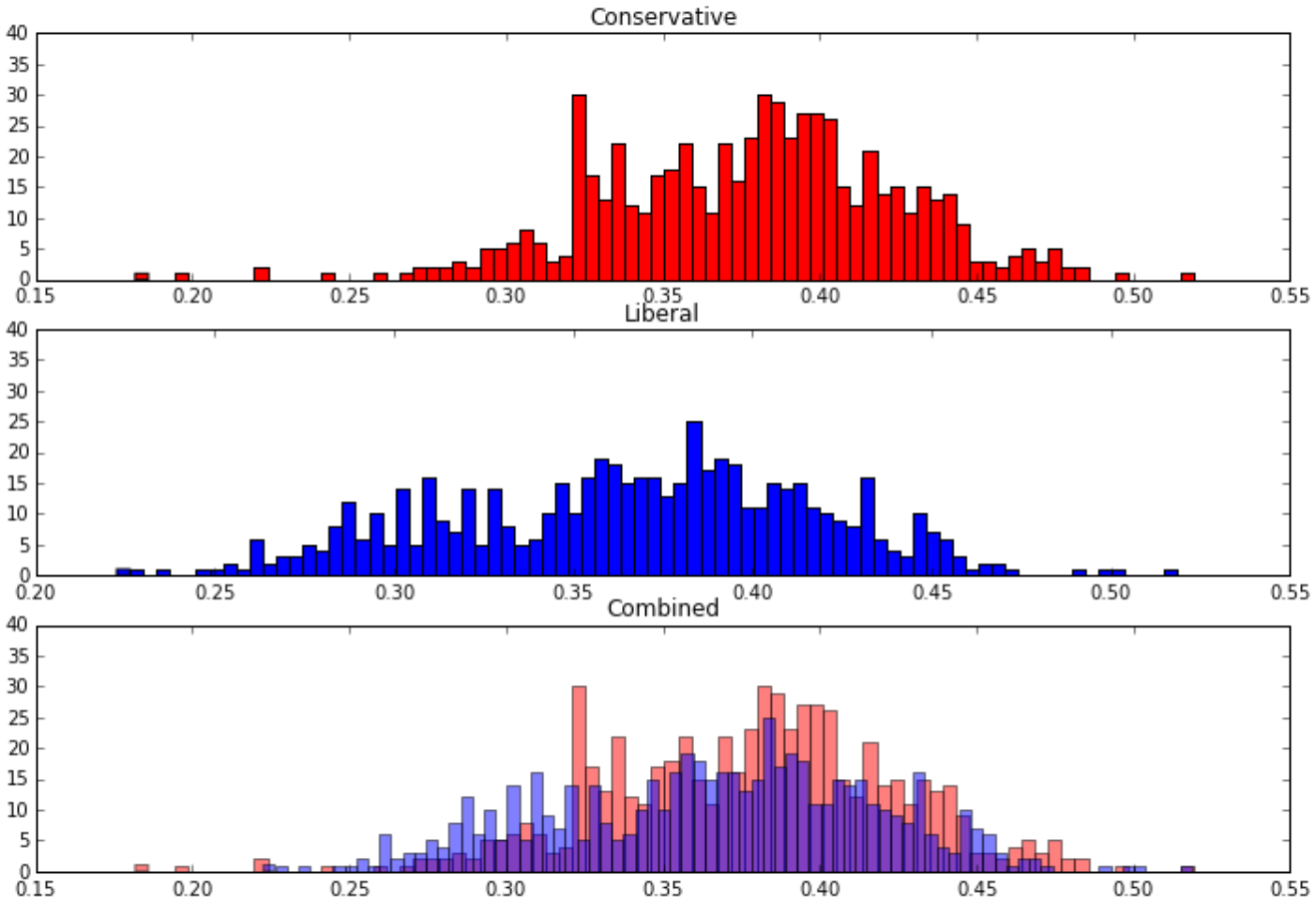
From the histograms at the begining, we have already seen that Liberals are better connected than Conservatives. The above sample dataset shows their rankings.

In [16]:
```
# Sort data by Closeness
df.sort('Closeness', ascending=False)
```

Out[16]:

	ID	Category	Degree	Closeness	Betweenness	Eigenvector
8	1051	1	307	0.519353	0.049596	0.114883
10	155	0	352	0.518692	0.088355	0.163853
0	641	0	275	0.503090	0.047669	0.149094
16	55	0	278	0.498367	0.034194	0.160075
6	1112	1	196	0.494532	0.024184	0.067420
7	729	0	218	0.490165	0.027744	0.138358
2	963	1	245	0.485101	0.068247	0.055585
3	1153	1	212	0.484140	0.027795	0.069879
13	1437	1	199	0.481087	0.042235	0.064435
5	855	1	302	0.478448	0.098009	0.053419
9	1245	1	224	0.474543	0.023734	0.075798
1	642	0	148	0.467816	0.012136	0.107519
15	180	0	147	0.458850	0.005039	0.116637
12	493	0	144	0.448567	0.007638	0.106894
11	363	0	171	0.440476	0.012325	0.117904
14	99	0	133	0.430384	0.002473	0.109835
4	979	1	75	0.429627	0.028141	0.016857

In [17]:
```
b = 80
maxy = 40
fig, ax = plt.subplots(3, 1, figsize=(12, 8))
ax[0].hist([j for (i, j) in cs if g.node[i]['value'] == 1],
           bins=b, color='red')
ax[0].set_ylim([0, maxy])
ax[0].set_title("Conservative")

ax[1].hist([j for (i, j) in cs if g.node[i]['value'] == 0],
           bins=b, color='blue')
ax[1].set_ylim([0, maxy])
ax[1].set_title("Liberal")

ax[2].hist([j for (i, j) in cs if g.node[i]['value'] == 1],
           bins=b, color='red', alpha=0.5)
ax[2].hist([j for (i, j) in cs if g.node[i]['value'] == 0],
           bins=b, color='blue', alpha=0.5)
ax[2].set_ylim([0, maxy])
ax[2].set_title("Combined")

plt.show()
```



Conclusion

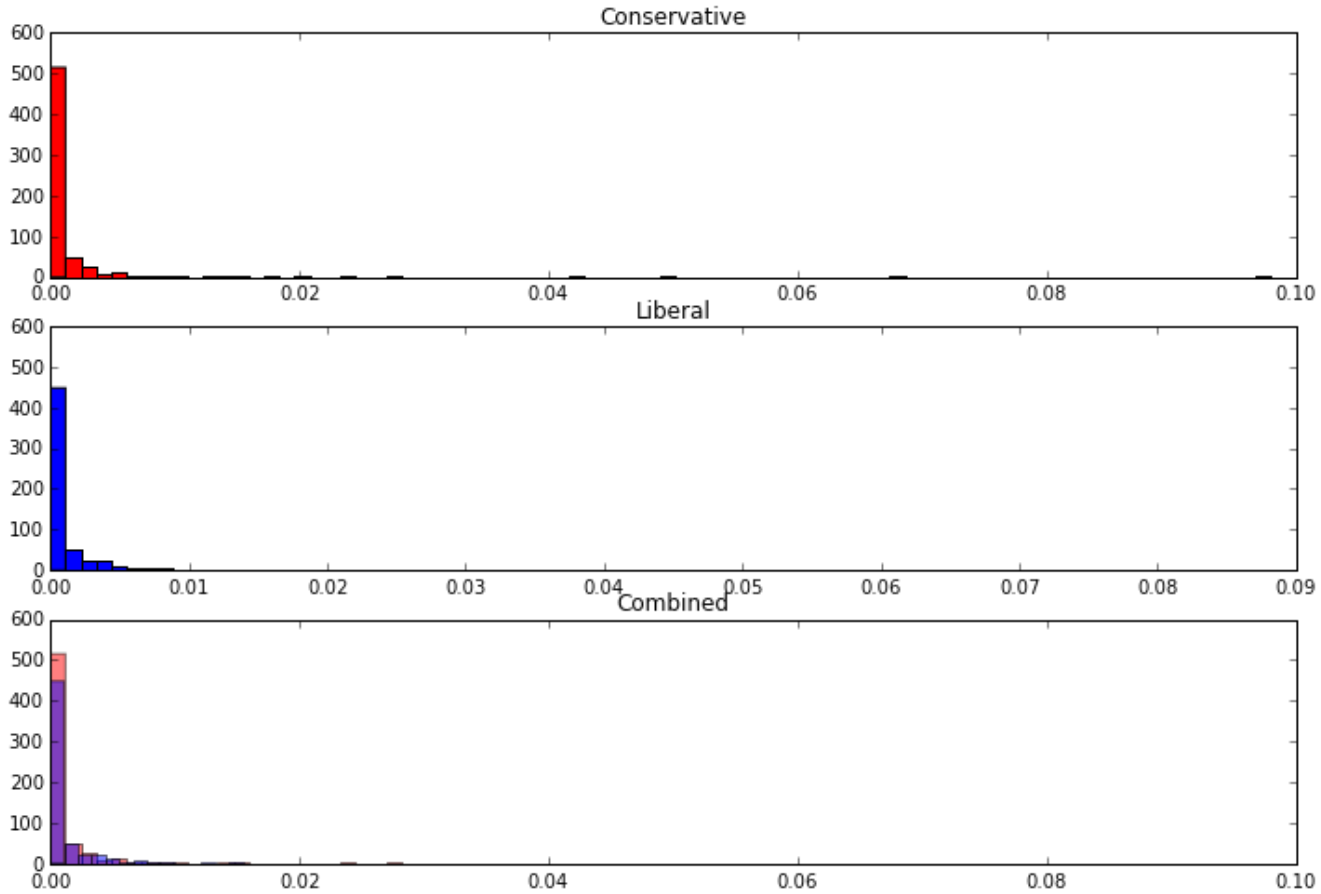
While a Conservative is at the top in close networking, Liberals are still ahead of the game.

In [18]:
```
# Sort data by Betweenness
df.sort('Betweenness', ascending=False)
```

Out[18]:

	ID	Category	Degree	Closeness	Betweenness	Eigenvector
5	855	1	302	0.478448	0.098009	0.053419
10	155	0	352	0.518692	0.088355	0.163853
2	963	1	245	0.485101	0.068247	0.055585
8	1051	1	307	0.519353	0.049596	0.114883
0	641	0	275	0.503090	0.047669	0.149094
13	1437	1	199	0.481087	0.042235	0.064435
16	55	0	278	0.498367	0.034194	0.160075
4	979	1	75	0.429627	0.028141	0.016857
3	1153	1	212	0.484140	0.027795	0.069879
7	729	0	218	0.490165	0.027744	0.138358
6	1112	1	196	0.494532	0.024184	0.067420
9	1245	1	224	0.474543	0.023734	0.075798
11	363	0	171	0.440476	0.012325	0.117904
1	642	0	148	0.467816	0.012136	0.107519
12	493	0	144	0.448567	0.007638	0.106894
15	180	0	147	0.458850	0.005039	0.116637
14	99	0	133	0.430384	0.002473	0.109835

In [19]:
```
b = 80
maxy = 600
fig, ax = plt.subplots(3, 1, figsize=(12, 8))
ax[0].hist([j for (i, j) in bs if g.node[i]['value'] == 1],
           bins=b, color='red')
ax[0].set_ylim([0, maxy])
ax[0].set_title("Conservative")

ax[1].hist([j for (i, j) in bs if g.node[i]['value'] == 0],
           bins=b, color='blue')
ax[1].set_ylim([0, maxy])
ax[1].set_title("Liberal")

ax[2].hist([j for (i, j) in bs if g.node[i]['value'] == 1],
           bins=b, color='red', alpha=0.5)
ax[2].hist([j for (i, j) in bs if g.node[i]['value'] == 0],
           bins=b, color='blue', alpha=0.5)
ax[2].set_ylim([0, maxy])
ax[2].set_title("Combined")

plt.show()
```



Conclusion

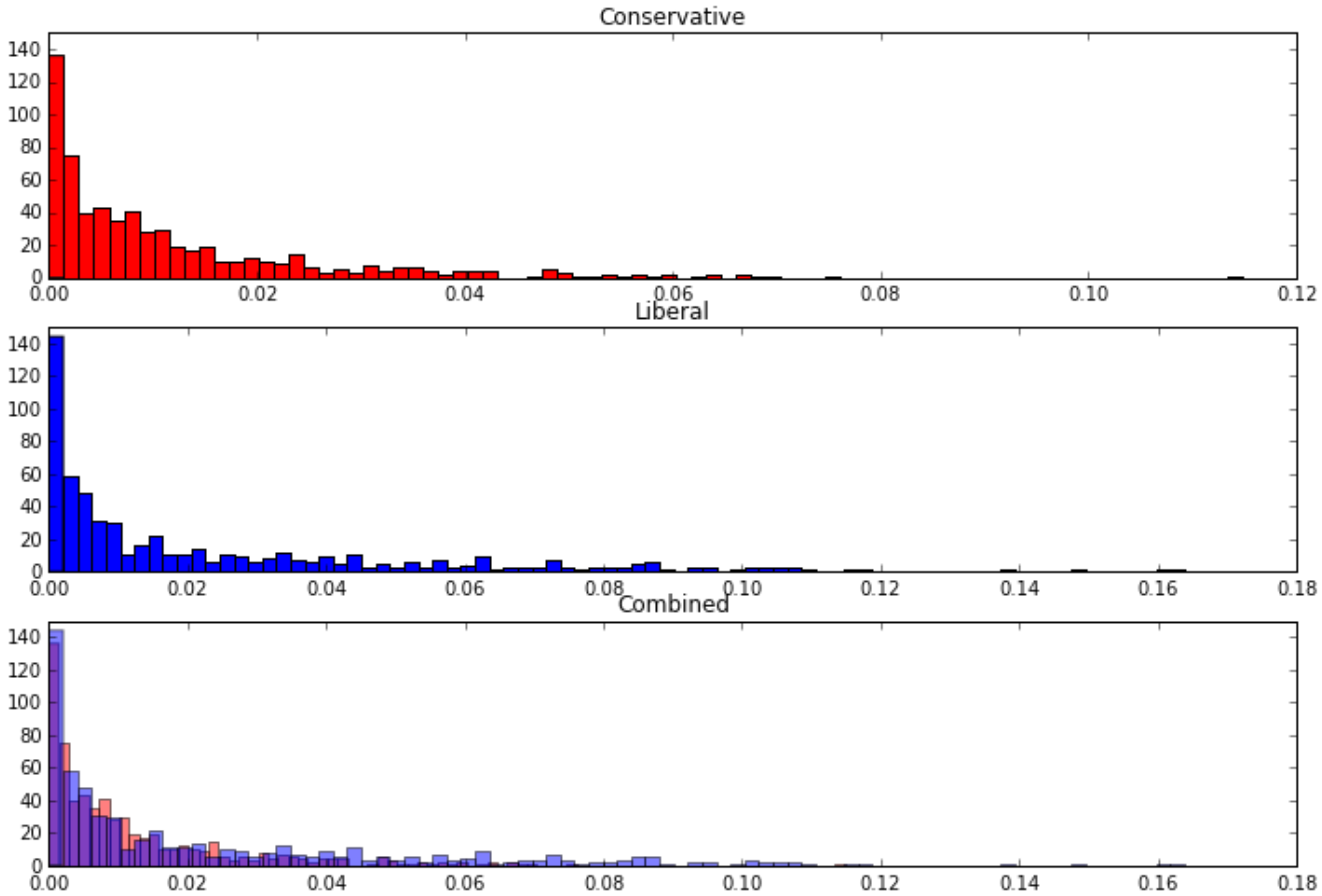
On betweenness, Conservatives are ahead of Liberals.

In [20]:
```
# And finally, sort data by Eigenvector
df.sort('Eigenvector', ascending=False)
```

Out[20]:

	ID	Category	Degree	Closeness	Betweenness	Eigenvector
10	155	0	352	0.518692	0.088355	0.163853
16	55	0	278	0.498367	0.034194	0.160075
0	641	0	275	0.503090	0.047669	0.149094
7	729	0	218	0.490165	0.027744	0.138358
11	363	0	171	0.440476	0.012325	0.117904
15	180	0	147	0.458850	0.005039	0.116637
8	1051	1	307	0.519353	0.049596	0.114883
14	99	0	133	0.430384	0.002473	0.109835
1	642	0	148	0.467816	0.012136	0.107519
12	493	0	144	0.448567	0.007638	0.106894
9	1245	1	224	0.474543	0.023734	0.075798
3	1153	1	212	0.484140	0.027795	0.069879
6	1112	1	196	0.494532	0.024184	0.067420
13	1437	1	199	0.481087	0.042235	0.064435
2	963	1	245	0.485101	0.068247	0.055585
5	855	1	302	0.478448	0.098009	0.053419
4	979	1	75	0.429627	0.028141	0.016857

In [21]:
```
b = 80
maxy = 150
fig, ax = plt.subplots(3, 1, figsize=(12, 8))
ax[0].hist([j for (i, j) in es if g.node[i]['value'] == 1],
           bins=b, color='red')
ax[0].set_ylim([0, maxy])
ax[0].set_title("Conservative")

ax[1].hist([j for (i, j) in es if g.node[i]['value'] == 0],
           bins=b, color='blue')
ax[1].set_ylim([0, maxy])
ax[1].set_title("Liberal")

ax[2].hist([j for (i, j) in es if g.node[i]['value'] == 1],
           bins=b, color='red', alpha=0.5)
ax[2].hist([j for (i, j) in es if g.node[i]['value'] == 0],
           bins=b, color='blue', alpha=0.5)
ax[2].set_ylim([0, maxy])
ax[2].set_title("Combined")

plt.show()
```



Conclusion

While we have used a simple weighing mechanism, Liberals are ahead of Conservatives using this measure.