Comparison of Character Rankings Between Graphs & Centrality Methods

Table 1: Rankings of characters in terms of degree centrality

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* - Weighted	
	Character	**Degree Centrality**	**Character**	**Degree Centrality**	**Character**	**Degree Centrality**
1	Gandalf	153	Gandalf	762	Gandalf	901
2	Frodo	144	Frodo	661	Frodo	661
3	Aragorn	140	Aragorn	632	Aragorn	632
4	Pippin	137	Pippin	606	Pippin	606
5	Elrond	125	Elrond	484	Bilbo	602

Interesting things which can be found with degree centrality, as seen in **Table 1**, is that in the graphs only considering *The Lord of the Rings*, the rankings do not change from the unweighted to weighted graphs. This effectively tells us that for at least the ranking of the top 5 most central characters, the proportion of which they share chapters with *different* characters is similar to the proportion of which they share chapters with other characters in general. Another important detail to be seen is that the degrees of Frodo, Aragorn, and Pippin remain the same whether or not *The Hobbit* is considered, meaning that these characters do not appear in *The Hobbit*, or they are the sole character of any chapters that they appear in in *The Hobbit.*

Table 2: Ranking of characters in terms of eigenvector centrality.

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* - Weighted	
	Character	**Eigenvector Centrality**	**Character**	**Eigenvector Centrality**	**Character**	**Eigenvector Centrality**
1	Gandalf	0.1682	Gandalf	0.3412	Gandalf	0.3570

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* - Weighted	
	Character	**Eigenvector Centrality**	**Character**	**Eigenvector Centrality**	**Character**	**Eigenvector Centrality**
2	Aragorn	0.1641	Frodo	0.2920	Frodo	0.2827
3	Frodo	0.1618	Aragorn	0.2908	Aragorn	0.2783
4	Elrond	0.1541	Pippin	0.2839	Pippin	0.2723
5	Pippin	0.1533	Sauron	0.2300	Bilbo	0.2287

The ranking of characters in terms of their eigenvector centrality is shown in **Table 2**. Unlike degree centrality, there is a large change in the character rankings when weights are considered. This is reflective of the fact that very important characters, such as Gandalf, who interact with a very large number of different characters, also spend more time interacting with other important characters than they do with less important characters. The rankings do not change much when considering *The Hobbit* in the graph. This can be due to the same reason three of the top five characters remained in the top 5 when considering *The Hobbit* despite not appearing in it, which is that *The Lord of The Rings* trilogy has over three times as many chapters as *The Hobbit*, and the importance of the characters from *The Lord of The Rings* is a larger factor when looking at the overall picture.

Table 3: Ranking of characters in terms of Katz centrality.

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* - Weighted	
	Character	**Katz Centrality** $(\alpha = 0.0157)$	**Character**	**Katz Centrality** $(\alpha = 0.0032)$	**Character**	**Katz Centrality** $(\alpha = 0.0031)$
1	Gandalf	292.9	Gandalf	81.5	Gandalf	86.0
2	Aragorn	285.6	Frodo	69.9	Frodo	68.1
3	Frodo	281.7	Aragorn	69.5	Aragorn	67.0
4	Elrond	268.2	Pippin	67.9	Pippin	65.5

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* - Weighted	
	Character	**Katz Centrality** $(\alpha = 0.0157)$	**Character**	**Katz Centrality** $(\alpha = 0.0032)$	**Character**	**Katz Centrality** $(\alpha = 0.0031)$
5	Pippin	266.9	Sauron	55.1	Bilbo	55.6

The ranking of characters in terms of their Katz centrality is shown in **Table 3**, where α is set to a value very close to λ_1^{-1}, where λ_1 is the largest eigenvalue of A. The results between Katz centrality and eigenvalue centrality are very similar, which is to be expected as all of the graphs are undirected. Thus, all of the trends and observations made for eigenvector centrality apply to Katz centrality as the results yielded are very similar. The value of α changes from graph to graph to satisfy the expression and give the most accurate result. During the tests it is seen with varying values of α, the rankings of the characters do not change from what has been seen, but the centrality values approach 1 as α approaches 0, and the difference between the centrality values for each character decreases.

Table 4: Ranking of characters in terms of PageRank when $\alpha = 0.95$.

Rank	*Lord of The Rings* - Unweighted		*Lord of The Rings* - Weighted		*Lord of The Rings* & *The Hobbit* – Weighted	
	Character	**PageRank**	**Character**	**PageRank**	**Character**	**PageRank**
1	Gandalf	68.09	Gandalf	145.77	Gandalf	160.20
2	Frodo	64.18	Frodo	127.53	Frodo	118.77
3	Aragorn	61.39	Aragorn	120.04	Aragorn	111.77
4	Pippin	60.88	Pippin	116.34	Pippin	108.42
5	Bilbo	54.95	Elrond	92.52	Bilbo	107.69

The ranking of characters in terms of their PageRank when $\alpha = 0.95$ is shown in **Table 4**. The fact that Elrond replaces Bilbo as the last character on the list when weights are considered, but regains that position once *The Hobbit* is considered can be attributed to a few things with the measurement of PageRank. PageRank is used to reduce the effect of high-degree nodes on their low-degree neighbors

when calculating centrality. Elrond being able to take the last spot on the list from bilbo when weights are considered can mean that Bilbo does not have as many interactions as Elrond, which is confirmed by the degree centrality, but those interactions that Bilbo does have are with other characters that are highly central. Bilbo commonly increases on the ranks when *The Hobbit* is introduced, which is likely a product of the fact that Bilbo is a more central character in *The Hobbit* than in *The Lord of The Rings* trilogy.

Table 5: Ranking of characters in terms of Betweenness Centrality.

Rank	***Lord of The Rings* - Unweighted**		***Lord of The Rings* - Weighted**		***Lord of The Rings* & *The Hobbit* – Weighted**	
	Character	**Centrality**	**Character**	**Centrality**	**Character**	**Centrality**
1	Gandalf	0.0797	Gandalf	0.0437	Gandalf	0.0498
2	Frodo	0.0675	Pippin	0.0366	Pippin	0.0370
3	Pippin	0.0613	Aragorn	0.0345	Aragorn	0.0339
4	Aragorn	0.0486	Frodo	0.0289	Frodo	0.0296
5	Bilbo	0.0393	Bilbo	0.0270	Bilbo	0.0292

The ranking of characters in terms of their betweenness centrality is shown in **Table 5**. We see a drastic change in the order when weights are considered, which is to be expected. The edges between highly central characters are likely to have a very large weight as main characters likely interact very frequently throughout the series. Betweenness centrality is calculated using shortest paths, and as weights, which are likely very large, are considered, it is very likely that these shortest paths will have significant changes, thus changing the nodes which are passed through frequently by these shortest paths. There is virtually no change in the betweenness centrality when *The Hobbit* is considered, which is likely due to the fact that *The Hobbit* has many less chapters than *The Lord of The Rings* trilogy, and thus contributes much less to the centrality of the network.

Lord of The Rings Unweighted Network Print

The following is the output of the lab3_tjk190000.ipynb file when analyzing the unweighted *Lord of The Rings* graph.

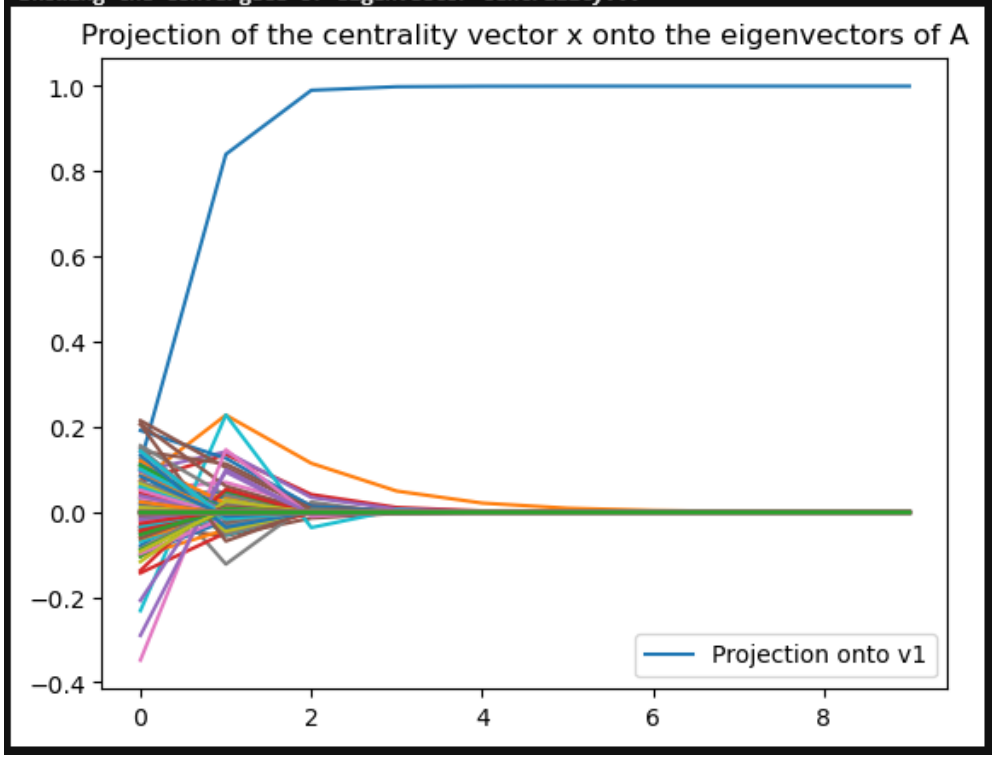
```
top 5 degree centrality characters
  1. gandalf (153.0000)
  2. frodo (144.0000)
  3. aragorn (140.0000)
  4. pippin (137.0000)
  5. elrond (125.0000)
```

```
Eigenvector Centrality (by NetworkX):
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

```
Eigenvector Centrality (by linear algebra):
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

```
Confirming that eigenvector centrality is a steady-state of sorts for node aragorn:
Eigenvector centrality for node aragorn: 0.1641
Sum of the centralities of neighbors of aragorn normalized by the larges eigenvalue: 0.1641
```

```
Showing the convergece of eigenvector centrality...
```




```
Katz Centrality:
Top 5 central characters using Katz centrality (alpha = 0.0158):
  1. gandalf (1549115332298075.5000)
  2. aragorn (1510834676614963.0000)
  3. frodo (1489883525045676.5000)
  4. elrond (1419026387096076.5000)
  5. pippin (1411232561834083.5000)
Top 5 central characters using Katz centrality (alpha = 0.0157):
  1. gandalf (292.9143)
  2. aragorn (285.5536)
  3. frodo (281.7088)
  4. elrond (268.1689)
  5. pippin (266.9178)
Top 5 central characters using Katz centrality (alpha = 0.0079):
  1. gandalf (3.1165)
  2. frodo (3.0126)
  3. aragorn (2.9915)
  4. pippin (2.9150)
  5. elrond (2.8220)
Top 5 central characters using Katz centrality (alpha = 0.0053):
  1. gandalf (2.1064)
  2. frodo (2.0484)
  3. aragorn (2.0306)
  4. pippin (1.9979)
  5. elrond (1.9351)
Top 5 central characters using Katz centrality (alpha = 0.0040):
  1. gandalf (1.7543)
  2. frodo (1.7135)
  3. aragorn (1.6993)
  4. pippin (1.6791)
  5. elrond (1.6319)
```

PageRank	Betweenness Centrality
1. gandalf (68.0983)	1. gandalf (0.0797)
2. frodo (64.1828)	2. frodo (0.0675)
3. aragorn (61.3929)	3. pippin (0.0613)
4. pippin (60.8781)	4. aragorn (0.0486)
5. bilbo (54.9509)	5. bilbo (0.0393)

Lord of The Rings Weighted Network Print

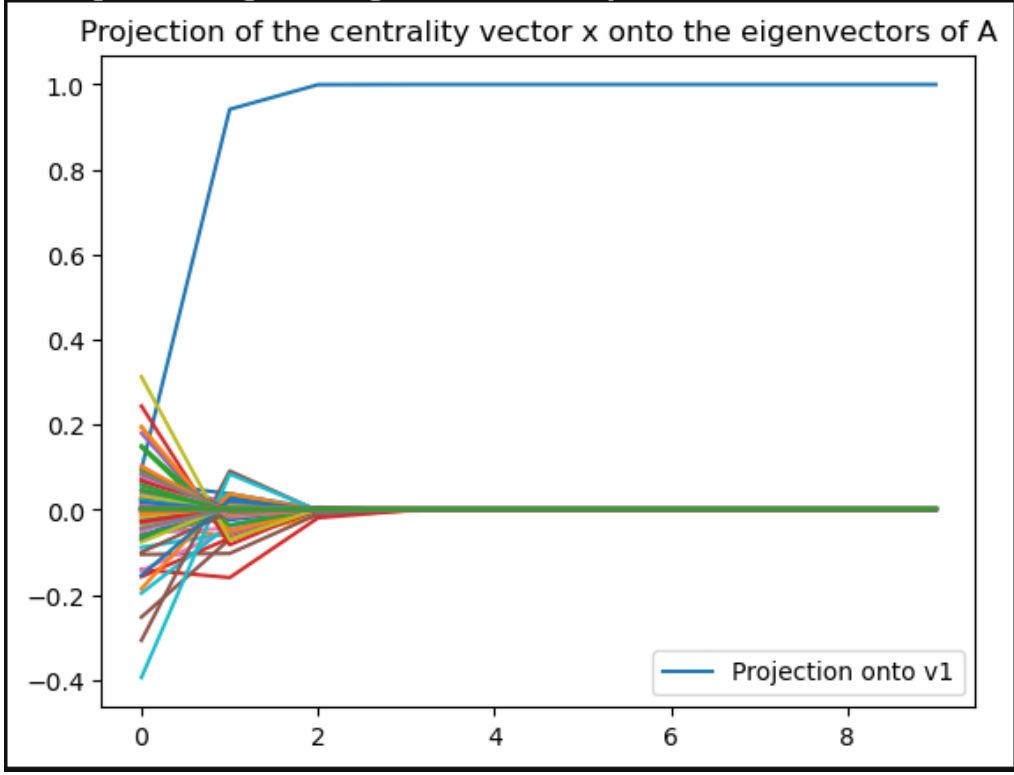
The following is the output of the lab3_tjk190000.ipynb file when analyzing the weighted *Lord of The Rings* graph.

```
top 5 degree centrality characters
  1. gandalf (762.0000)
  2. frodo (661.0000)
  3. aragorn (632.0000)
  4. pippin (606.0000)
  5. elrond (484.0000)
```

```
Eigenvector Centrality (by NetworkX):
  1. gandalf (0.3412)
  2. frodo (0.2920)
  3. aragorn (0.2908)
  4. pippin (0.2839)
  5. sauron (0.2300)
```

```
Eigenvector Centrality (by linear algebra):
  1. gandalf (0.3412)
  2. frodo (0.2920)
  3. aragorn (0.2908)
  4. pippin (0.2839)
  5. sauron (0.2300)
```

```
Confirming that eigenvector centrality is a steady-state of sorts for node aragorn:
Eigenvector centrality for node aragorn: 0.2908
Sum of the centralities of neighbors of aragorn normalized by the larges eigenvalue: 0.2908
Showing the convergece of eigenvector centrality...
```




```
Katz Centrality:
Top 5 central characters using Katz centrality (alpha = 0.0033):
  1. ghan (-27288611503717.1094)
  2. orome (-27288611503717.1094)
  3. twofoot (-34698631076483.9375)
  4. proudfoot (-34698631076483.9414)
  5. adelard (-34698631076483.9414)
Top 5 central characters using Katz centrality (alpha = 0.0032):
  1. gandalf (81.4709)
  2. frodo (69.9094)
  3. aragorn (69.5233)
  4. pippin (67.8687)
  5. sauron (55.1369)
Top 5 central characters using Katz centrality (alpha = 0.0017):
  1. gandalf (3.5108)
  2. frodo (3.1648)
  3. aragorn (3.1121)
  4. pippin (3.0444)
  5. sauron (2.6388)
Top 5 central characters using Katz centrality (alpha = 0.0011):
  1. gandalf (2.2607)
  2. frodo (2.0893)
  3. aragorn (2.0557)
  4. pippin (2.0187)
  5. sauron (1.8135)
Top 5 central characters using Katz centrality (alpha = 0.0008):
  1. gandalf (1.8423)
  2. frodo (1.7285)
  3. aragorn (1.7037)
  4. pippin (1.6780)
  5. sauron (1.5403)
```

PageRank	Betweenness Centrality
1. gandalf (145.7734)	1. gandalf (0.0437)
2. frodo (127.5262)	2. pippin (0.0366)
3. aragorn (120.0421)	3. aragorn (0.0345)
4. pippin (116.3430)	4. frodo (0.0289)
5. elrond (92.5220)	5. bilbo (0.0270)

Lord of The Rings & *The Hobbit* Weighted Network Print-out

The following is the output of the lab3_tjk190000.ipynb file when analyzing the weighted *Lord of The Rings* & *The Hobbit* graph.

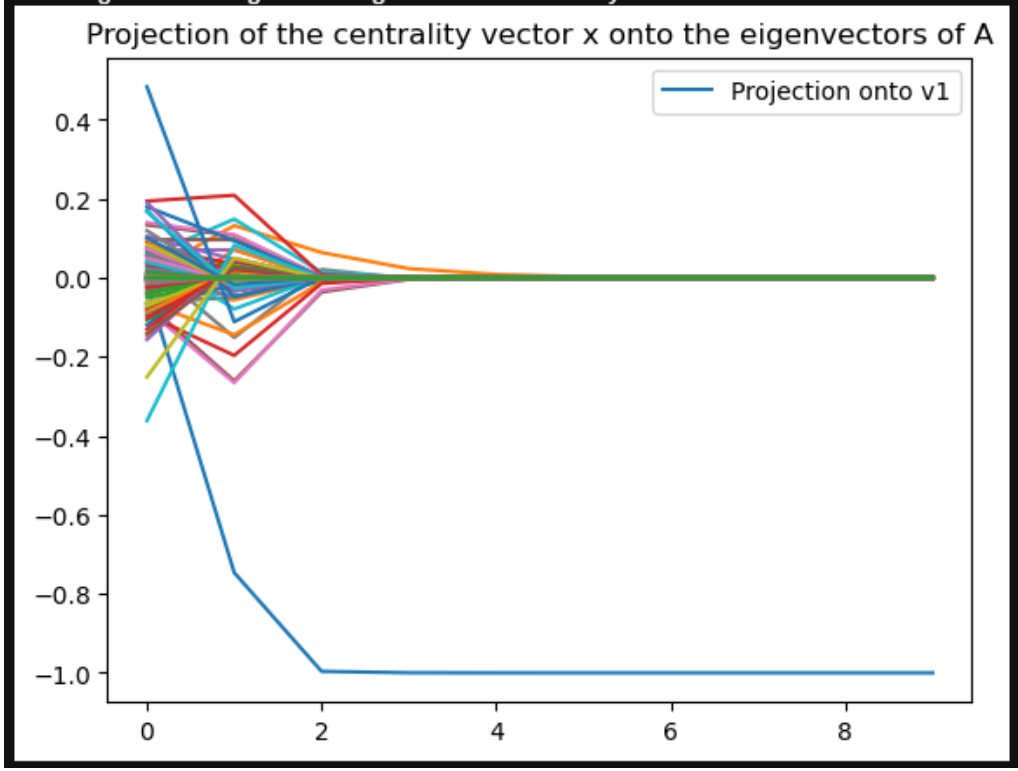
```
top 5 degree centrality characters
  1. gandalf (901.0000)
  2. frodo (661.0000)
  3. aragorn (632.0000)
  4. pippin (606.0000)
  5. bilbo (602.0000)
```

```
Eigenvector Centrality (by NetworkX):
  1. gandalf (0.3570)
  2. frodo (0.2827)
  3. aragorn (0.2783)
  4. pippin (0.2723)
  5. bilbo (0.2287)
```

```
Eigenvector Centrality (by linear algebra):
  1. gandalf (0.3570)
  2. frodo (0.2827)
  3. aragorn (0.2783)
  4. pippin (0.2723)
  5. bilbo (0.2287)
```

```
Confirming that eigenvector centrality is a steady-state of sorts for node aragorn:
Eigenvector centrality for node aragorn: 0.2783
Sum of the centralities of neighbors of aragorn normalized by the larges eigenvalue: 0.2783
```

```
Showing the convergece of eigenvector centrality...
```




```
Katz Centrality:
Top 5 central characters using Katz centrality (alpha = 0.0032):
  1. gandalf (3506233965768915.0000)
  2. frodo (2776134896343451.5000)
  3. aragorn (2733443552956267.5000)
  4. pippin (2674606740861643.0000)
  5. bilbo (2246654230718593.0000)
Top 5 central characters using Katz centrality (alpha = 0.0031):
  1. gandalf (85.9849)
  2. frodo (68.0973)
  3. aragorn (66.9911)
  4. pippin (65.5381)
  5. bilbo (55.6171)
Top 5 central characters using Katz centrality (alpha = 0.0016):
  1. gandalf (3.8101)
  2. frodo (3.1354)
  3. aragorn (3.0695)
  4. pippin (3.0064)
  5. bilbo (2.8592)
Top 5 central characters using Katz centrality (alpha = 0.0011):
  1. gandalf (2.4204)
  2. frodo (2.0663)
  3. aragorn (2.0286)
  4. pippin (1.9938)
  5. bilbo (1.9446)
Top 5 central characters using Katz centrality (alpha = 0.0008):
  1. gandalf (1.9521)
  2. frodo (1.7105)
  3. aragorn (1.6839)
  4. pippin (1.6595)
  5. bilbo (1.6343)
```

PageRank	Betweenness Centrality
1. gandalf (160.2037)	1. gandalf (0.0498)
2. frodo (118.7663)	2. pippin (0.0370)
3. aragorn (111.7720)	3. aragorn (0.0339)
4. pippin (108.4245)	4. frodo (0.0296)
5. bilbo (107.6881)	5. bilbo (0.0292)

Lab 3

Tanner Kogel tjk190000

MECH 6317.001: Dynaics of Complex Networks & Systems

imoport all important libraries

In [302…
```
import networkx as nx
import numpy
import numpy.linalg as la
import matplotlib.pyplot as plt
```

Stub code functions

This function prints the top five (or `num`) nodes according to the centrality vector `v`, where `v` takes the form: `v[nidx]` is the centrality of the node that is the `nidx`-th element of `G.nodes()`

In [303…
```
def print_top_5(G,v, num=5):
    thenodes = list(G.nodes())
    idx_list = [(i,v[i]) for i in range(len(v))]
    idx_list = sorted(idx_list, key = lambda x: x[1], reverse=True)
    for i in range(min(num,len(idx_list))):
        nidx, score = idx_list[i]
        print('  %i. %s (%1.4f)' % (i+1,thenodes[nidx],score))
        #print '  %i. %s' % (i+1,G.node_object(idx))
```

This function returns the index of the maximum of the array. If two or more indices have the same max value, the first index is returned.

In [304…
```
def index_of_max(v):
    return numpy.where(v == max(v))[0]
```

This function accepts a dictionary of nodes with centrality values and returns a centrality vector

In [305…
```
def centrality_vector(G,d):
    thenodes = list(G.nodes())
    v = numpy.zeros((G.number_of_nodes(),))
    for i,u in enumerate(thenodes):
        v[i] = d[u]
    return v
```

This function provides the index of a node based on its order in `G.nodes()`

In [306…
```
def node_index(G,n):
    thenodes = list(G.nodes())
    return thenodes.index(n)
```

Now we read in the edgelist file that contains the coappearance network we will analyze. We will look at two different networks, corresponding to only the *Lord of the Rings* series and the *Lord of the Rings* series plus the prequel, The *Hobbit*. The `unweighted` boolean, if set to `True` will set all the edge weights to one. Recall that setting all weights to 1 is different (in NetworkX) from having no weights assigned, which could be accomplished instead by: `G = nx.read_edgelist('LoTR_characters.edgelist',data=False)`.

In [307…
```
unweighted = False
#G = nx.read_weighted_edgelist('LotR_characters.edgelist') # just Lord of the Rings
G = nx.read_weighted_edgelist('hobbit_LotR_characters.edgelist') # with the Hobbit
if unweighted:
    for u,v in G.edges():
        G[u][v]['weight'] = 1
A = nx.to_numpy_array(G)
N = G.number_of_nodes()
```

Section 7.1: Degree Centrality

use the defined function from the stub code to print out top 5 characters with highest degree centrality

In [308…
```
d = dict.fromkeys(G.nodes(),0)              # predefine d as a dictionary for all nodes in G with degree 0
for i in G.nodes():                         # loop over all nodes
    idx = node_index(G,i)                   # get index for node i
    d[i] = sum(A[idx])                      # sum all weights in row of adjacency matrix
v = centrality_vector(G,d)                  # use function to get centrality vector
```

```
print('top 5 degree centrality characters') # label output
print_top_5(G,v)                            # use function to print top five characters in terms of degree centrality (
```

```
top 5 degree centrality characters
  1. gandalf (901.0000)
  2. frodo (661.0000)
  3. aragorn (632.0000)
  4. pippin (606.0000)
  5. bilbo (602.0000)
```

Section 7.2: Eigenvector Centrality

print out the eigenvector centrality by using the built in function eigenvector_centrality

In [309…

```
print('Eigenvector Centrality (by NetworkX):')   # display the method via networkx
d = nx.eigenvector_centrality(G,weight='weight') # dictionary output of networkx function
v = centrality_vector(G,d)                       # get eigenvector centrality vector
print_top_5(G,v)                                 # use function to output top 5 characters
```

```
Eigenvector Centrality (by NetworkX):
  1. gandalf (0.3570)
  2. frodo (0.2827)
  3. aragorn (0.2783)
  4. pippin (0.2723)
  5. bilbo (0.2287)
```

print out the eigenvector centrality by using linear algebra

In [310…

```
print('Eigenvector Centrality (by linear algebra):') # display the method via linear algebra
k, v = la.eig(A)                                     # find eigenvalues k and eigenvectors v
k1_idx = index_of_max(k)                             # find the index of the largest eigenvalue
v_R = numpy.abs(v[:,k1_idx])                         # centrality vector of function (assumes no complex values)
print_top_5(G,v_R)                                   # use function to output top 5 characters
```

```
Eigenvector Centrality (by linear algebra):
  1. gandalf (0.3570)
  2. frodo (0.2827)
  3. aragorn (0.2783)
  4. pippin (0.2723)
  5. bilbo (0.2287)
```

print out both the centrality and the weighted sum of the centralities(normalized by k_1) of the neighbors of any character

In [311…

```
noi = 'aragorn'             # choose any character
noi_idx = node_index(G,noi) # use function to find index value for given character
# label outupt
print('Confirming that eigenvector centrality is a steady-state of sorts for node %s:' % noi)

centrality = numpy.abs(v[noi_idx,k1_idx])                                    # find eigenvector centrality from centra
print('Eigenvector centrality for node %s: %1.4f' % (noi,centrality))        # output eigenvector centrality
centrality_sum = 0                                                           # initialize a value at 0
noi_neighb_list = list(G.neighbors(noi))                                     # find all neighbors
for neighb in noi_neighb_list:                                               # add each neighbor
    neighb_idx = node_index(G,neighb)                                        # index of neighbors
    neighb_centrality = numpy.abs(v[neighb_idx,k1_idx])                      # centrality value of neighbors
    centrality_sum = centrality_sum + A[noi_idx,neighb_idx]*neighb_centrality # add weighted centrality of neighbors
centrality_sum = centrality_sum / numpy.abs(max(k))                          # normalize by largest eigenvector
# output normalized centrality sum
print('Sum of the centralities of neighbors of %s normalized by the larges eigenvalue: %1.4f' % (noi,centrality_sum))
```

```
Confirming that eigenvector centrality is a steady-state of sorts for node aragorn:
Eigenvector centrality for node aragorn: 0.2783
Sum of the centralities of neighbors of aragorn normalized by the larges eigenvalue: 0.2783
```

The following section should run and produce a plot that you need to interpret.

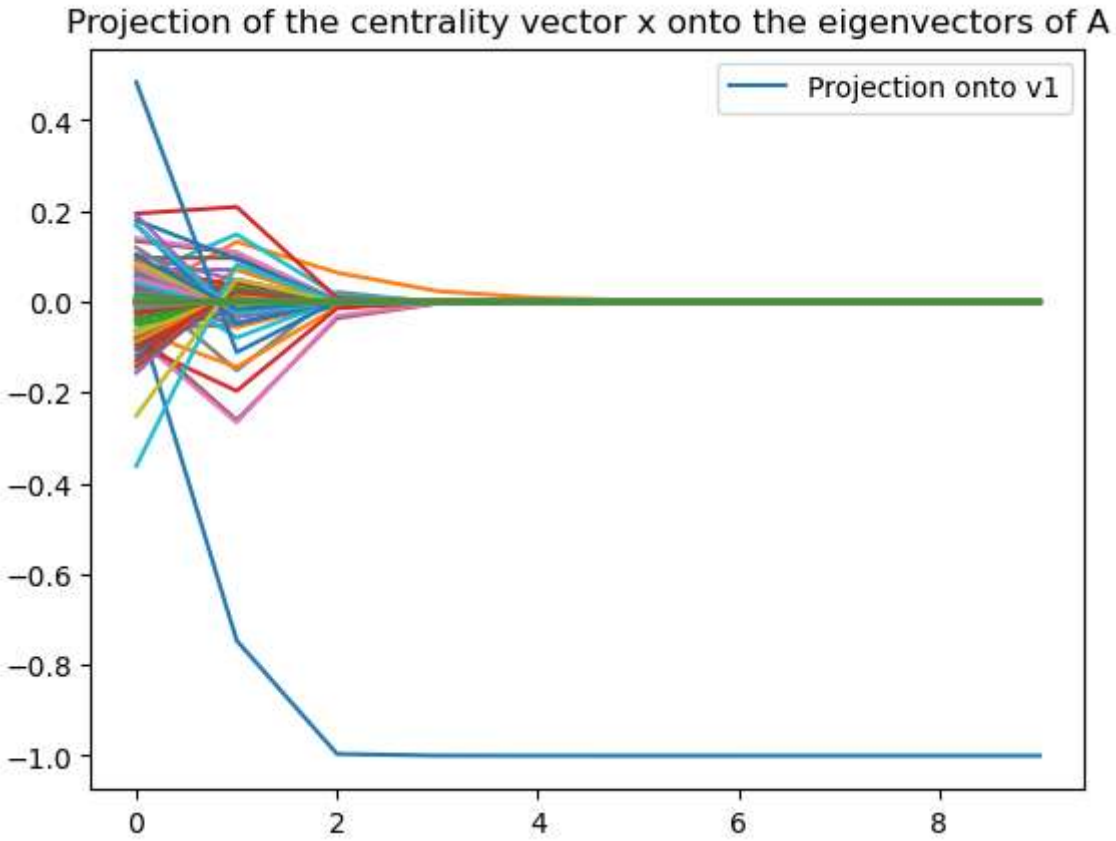
In [312…

```
print('Showing the convergece of eigenvector centrality...')
num_steps = 10
x = numpy.zeros((N,1)) # initial centrality vector
x[76] = 1
cs = numpy.zeros((N,num_steps))
for i in range(num_steps):
    x = x/la.norm(x) # at each step we need to normalize the centrality vector
    for j in range(G.number_of_nodes()):
        cs[j,i] = numpy.real(numpy.dot( x.T , v[:,j] ))[0] # project x onto each of the eigenvectors
    x = numpy.dot(A,x) # "pass" the centrality one step forward

plt.figure() # this creates a figure to plot in
for i in range(G.number_of_nodes()): # for each eigenvector plot the projection of x onto it over the steps
    if i == k1_idx:
        plt.plot(range(num_steps),cs[i,:],label='Projection onto v1') # only label the eigenvector v1
    else:
        plt.plot(range(num_steps),cs[i,:])
```

```
#plt.ylim([-0.2,1.1]) # this sets the limits for the y axis
plt.legend(loc='best') # this attaches a legend
plt.title('Projection of the centrality vector x onto the eigenvectors of A') # this adds a title
plt.show() # this makes the figure appear
```

```
Showing the convergece of eigenvector centrality...
```



In this graph, we are seeing the evolution of the eigenvector centrality from an arbitrary starting point. The blue line that quickly approaches 1 is the eigenvector that relates to the largest eigenvalue. This eigenvector is equivalent to the centraliy vector while the other eigenvectors are ignored. The graph proves that this method is valid as the ignored eigenvectors converge to zero in steady state, and the eigenvector centrality converges to 1 in steady state.

Section 7.3: Katz Centrality

print the top 5 central characters in terms of Katz centrality for varying values of α using linear algebra

In [313…

```
print('Katz Centrality:')                                                    # label section
inv_max_eig = 1/max(numpy.abs(k))                                            # inverse of largest eigenvalue
alpha_list = [inv_max_eig,inv_max_eig-0.0001,inv_max_eig/2,inv_max_eig/3,inv_max_eig/4] # various alpha values
for alpha in alpha_list:                                                     # repeat for each alpha value
    v = numpy.dot( la.inv(numpy.eye(N) - alpha*A) , numpy.ones((N,1)) )      # linear algebraic equation for
    print('Top 5 central characters using Katz centrality (alpha = %1.4f):' % alpha)    # label output with alpha value
    print_top_5(G,v)                                                         # output top characters
```

```
Katz Centrality:
Top 5 central characters using Katz centrality (alpha = 0.0032):
  1. gandalf (3506233965768915.0000)
  2. frodo (2776134896343451.5000)
  3. aragorn (2733443552956267.5000)
  4. pippin (2674606740861643.0000)
  5. bilbo (2246654230718593.0000)
Top 5 central characters using Katz centrality (alpha = 0.0031):
  1. gandalf (85.9849)
  2. frodo (68.0973)
  3. aragorn (66.9911)
  4. pippin (65.5381)
  5. bilbo (55.6171)
Top 5 central characters using Katz centrality (alpha = 0.0016):
  1. gandalf (3.8101)
  2. frodo (3.1354)
  3. aragorn (3.0695)
  4. pippin (3.0064)
  5. bilbo (2.8592)
Top 5 central characters using Katz centrality (alpha = 0.0011):
  1. gandalf (2.4204)
  2. frodo (2.0663)
  3. aragorn (2.0286)
  4. pippin (1.9938)
  5. bilbo (1.9446)
Top 5 central characters using Katz centrality (alpha = 0.0008):
  1. gandalf (1.9521)
  2. frodo (1.7105)
  3. aragorn (1.6839)
  4. pippin (1.6595)
  5. bilbo (1.6343)
```

Section 7.4: PageRank

print the top 5 central characters in terms of PageRank using linear algebra

In [314…

```
print('PageRank')              # label section
D = numpy.zeros((N,N))         # create fully zero matrix of size NxN
for i in G.nodes():            # loop to find nodes that have zero out-degree
    idx = node_index(G,i)      # find index value of node
    k_out = sum(A[:,idx])      # calculate out-degree of the node
    D[idx,idx] = max(k_out,1) # definition of D
alpha = 0.95                   # guess below 1
v = numpy.dot( la.inv( numpy.eye(N) - alpha*numpy.dot(A,la.inv(D)) ) , numpy.ones((N,1)) ) # calculate PageRank central
print_top_5(G,v) # output top characters
```

```
PageRank
  1. gandalf (160.2037)
  2. frodo (118.7663)
  3. aragorn (111.7720)
  4. pippin (108.4245)
  5. bilbo (107.6881)
```

The proof that $v_1 = (k_1,k_2,...,k_n)$ where k_i is the degree of node i, is an eigenvector of AD^{-1} is shown in the lab3_tjk19000.pdf file submitted with this lab

Section 7.5: Hubs & Authorites

the expression of hub eigenvectors in terms of the authorities eigenvectors is shown in the lab3_tjk190000.odf file submitted with this lab

Section 7.7: Betweenness Centrality

Using the networkx function for betweenness centrality print the top 5 characters with the highest betweenness centrality

In [315…

```
print('Betweenness Centrality')                  # label section
d = nx.betweenness_centrality(G,weight='weight') # use function to recieve dictionary
```

```
v = centrality_vector(G,d)                    # find betweenness centrality vector from dictionary
print_top_5(G,v)                              # output top characters
```

```
Betweenness Centrality
  1. gandalf (0.0498)
  2. pippin (0.0370)
  3. aragorn (0.0339)
  4. frodo (0.0296)
  5. bilbo (0.0292)
```

By definition of a connected, undirected graph, the adjacency matrix is symmetric, and no nodes have an out degree of 0, and out-degree is equivalent to degree. Therefore the following definitions can be defined:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1n} & \cdots & & a_{nn} \end{bmatrix} \qquad D = \begin{bmatrix} k_1 & & & \\ & k_2 & & \\ & & \ddots & \\ & & & k_n \end{bmatrix}$$

$$v_1 = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix} \qquad k_i = \sum_{j=1}^{n} a_{ij}$$

Because D is a diagonal matrix, its inverse can easily be found.

$$D^{-1} = \begin{bmatrix} k_1^{-1} & & & \\ & k_2^{-1} & & \\ & & \ddots & \\ & & & k_n \end{bmatrix}$$

The definition of an eigenvector v, of AD^{-1}, is that $AD^{-1}v = \lambda v$ for some eigenvalue λ that is a scalar. To prove that v_1 is an eigenvector of AD^{-1}, $AD^{-1}v_1$ will be found.

$$AD^{-1}v_1 = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1n} & \cdots & & a_{nn} \end{bmatrix} \begin{bmatrix} k_1^{-1} & & & \\ & k_2^{-1} & & \\ & & \ddots & \\ & & & k_n \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix}$$

$$AD^{-1}v_1 = \begin{bmatrix} a_{11}k_1^{-1} & a_{12}k_2^{-1} & \cdots & a_{1n}k_n^{-1} \\ a_{12}k_1^{-1} & a_{22}k_2^{-1} & & \vdots \\ \vdots & & \ddots & \\ a_{1n}k_1^{-1} & \cdots & & a_{nn}k_n^{-1} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j} \\ \sum_{j=1}^{n} a_{2j} \\ \vdots \\ \sum_{j=1}^{n} a_{nj} \end{bmatrix}$$

$$AD^{-1}v_1 = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix} \qquad \therefore AD^{-1}v_1 = (1)v_1$$

By definition, v_1 is an eigenvalue of AD^{-1} with an eigen value of $\underline{1}$.

α must be chosen as, $\alpha < \lambda_1^{-1}$, where λ_1 is the largest eigenvalue of AD^{-1}, and in this case, that value is $\underline{1}$.

The definition of an eigen vector, and its corresponding eigen value of A is:

$$Av = \lambda v$$

where A is the matrix of interest, v is an eigen vector, & λ is the corresponding eigen value.
By definition of authority & hub centrality:

$$AA^T x = \lambda x \qquad A^T A y = \lambda y$$

where A is the adjacency matrix, x is the authority eigenvector(s) and y is the hub eigen vector(s).

$$AA^T x = \lambda x$$
$$A^T(AA^T x) = A^T(\lambda x)$$

Because λ is a scalar, A^T can commute to create:

$$A^T A(A^T x) = \lambda(A^T x)$$

where it can be seen that $A^T x$ is an eigen value of $A^T A$. From the definitions we know that the eigen vectors of $A^T A$ are the hub eigen vectors. Thus:

$$\boxed{y = A^T x}$$