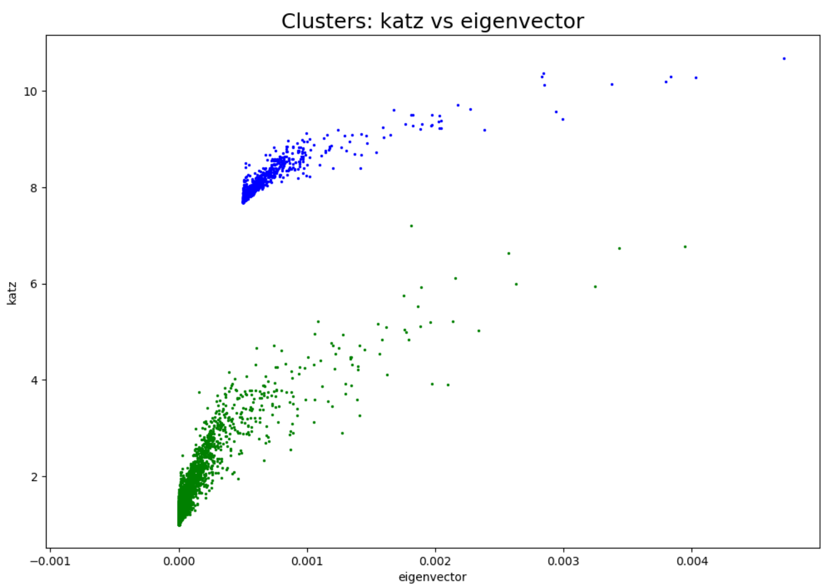
Assortativity and Katz vs Eigenvector Centrality

January 20, 2019

Purpose We are testing the idea that all assortatively mixed graphs with differing densities show distinct clusters in the Katz centrality vs Eigenvector centrality plot, as shown in the figure below. More specifically, it is supposed that in strongly assortatively mixed graphs, the denser of the two node classes benefits more from the free centrality in the Katz calculation than the sparser.



Methods We will first randomly generate two graphs of different densities. These two graphs will serve as the two node classes. A selection of nodes from both classes will be connected to create the overall assortatively mixed graph. Once the assortatively mixed network is created the Katz vs eigenvector centrality plot will be generated to determine if the effect is present.

```
In [2]: import zen
        import matplotlib.pyplot as plt
        import numpy as np
```

```
In [283]: # first two graphs
          Size1 = 1000
          Size2 = 250
          Size = Size1 + Size2
```

```
G1 = zen.generating.barabasi_albert(Size1,2)
G2 = zen.generating.barabasi_albert(Size2,30)
#G1 = zen.generating.erdos_renyi(Size1,0.01)
#G2 = zen.generating.erdos_renyi(Size2,0.2)

gcc1 = zen.algorithms.clustering.gcc(G1)
gcc2 = zen.algorithms.clustering.gcc(G2)

print 'Graph 1:'
print 'Average Degree:    %.2f'%(2.0*G1.num_edges /G1.num_nodes)
print 'Global Clustering: %.3f'%gcc1
print ''
print 'Graph 2:'
print 'Average Degree:    %.2f'%(2.0*G2.num_edges /G2.num_nodes)
print 'Global Clustering: %.3f'%gcc2
```

```
Graph 1:
Average Degree:    3.99
Global Clustering: 0.008

Graph 2:
Average Degree:    52.80
Global Clustering: 0.298
```

In [284]:
```
# Form Overall Graph
G = zen.Graph()
for i in range(Size):
    G.add_node(i)

for edge in G1.edges_iter():
    u = edge[0]
    v = edge[1]
    G.add_edge(u,v)

for edge in G2.edges_iter():
    u = edge[0]+Size1
    v = edge[1]+Size1
    G.add_edge(u,v)

# Select random pairs of nodes to connect the subgraphs
EDGE_NUM=400
join_nodes = np.empty((EDGE_NUM,2),dtype=np.int64)
nodes1 = np.random.randint(0,Size1,size=EDGE_NUM)
nodes2 = np.random.randint(Size1,Size,size=EDGE_NUM)
join_nodes[:,0] = nodes1
join_nodes[:,1] = nodes2
```

```
        for edge in join_nodes:
            if not G.has_edge(edge[0],edge[1]):
                G.add_edge(edge[0],edge[1])
```

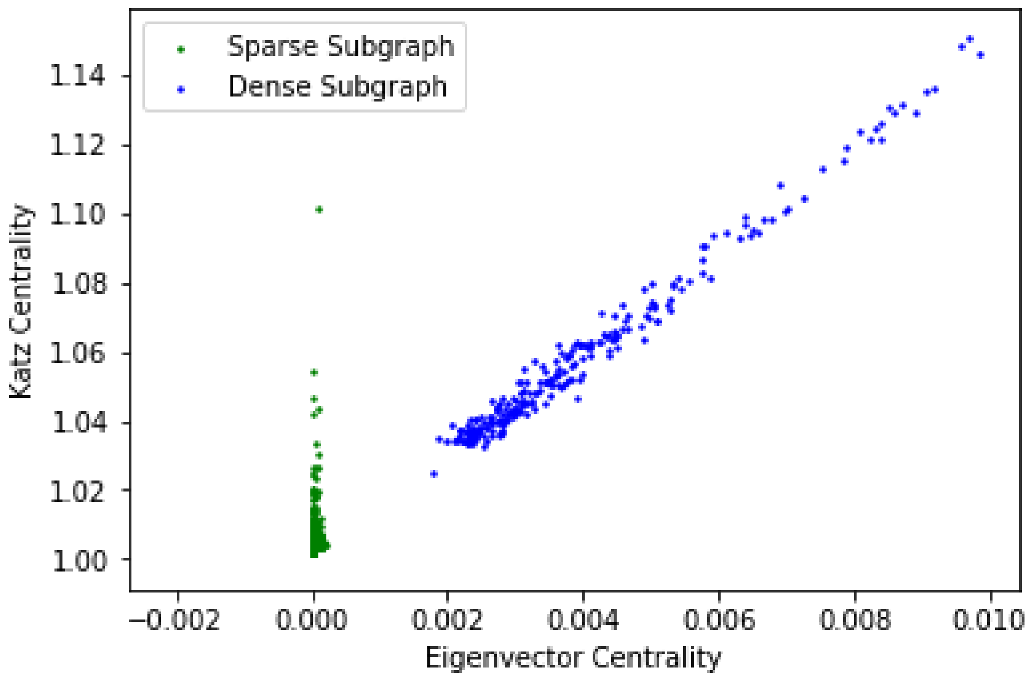
```
In [285]: # Modularity Calculation
          classes = {0:np.arange(0,Size1),1:np.arange(Size1,Size)}
          Q = zen.algorithms.modularity(G,classes)

          # Maximum Modularity
          count=0.0
          for e in G.edges():
              if (e[0]<Size1 and e[1]<Size1) or (e[0]>=Size1 and e[1]>=Size1):
                  count += 1
          same = count / G.num_edges
          rand = same - Q
          qmax = 1 - rand

          print 'Modularity:      %.3f'%Q
          print 'Max. Modularity: %.3f'%qmax
          print 'Normalized Mod:  %.3f'%(Q/qmax)
```

```
Modularity:      0.325
Max. Modularity: 0.369
Normalized Mod:  0.880
```

At this point we have generated two random graphs using preferential attachment, and joined them with 20 random edges. The resulting combined graph is strongly assortatively mixed as indicated by the normalized modularity value. To see if this graph indicates the same phemonenon of boosted Katz centrality, we will display the two groups in a Katz vs eigenvector centrality plot.

```
In [286]: def katz(G,tol=0.01,max_iter=1000,alpha=0.001,beta=1):
              iteration = 0
              centrality = np.zeros(G.num_nodes)
              while iteration < max_iter:
                  iteration += 1          # increment iteration count
                  centrality_old = centrality.copy()

                  for node in G.nodes_():
                      Ax = 0
                      for neighbor in G.neighbors_(node):
                          #weight = G.weight_(G.edge_idx_(neighbor,node))
                          #Ax += np.multiply(centrality[neighbor],weight)

                          Ax += centrality[neighbor]      #exclude weight due to overflow in mul

                      centrality[node] = np.multiply(alpha,Ax)+beta

                  if np.sum(np.abs(np.subtract(centrality,centrality_old))) < tol:
```

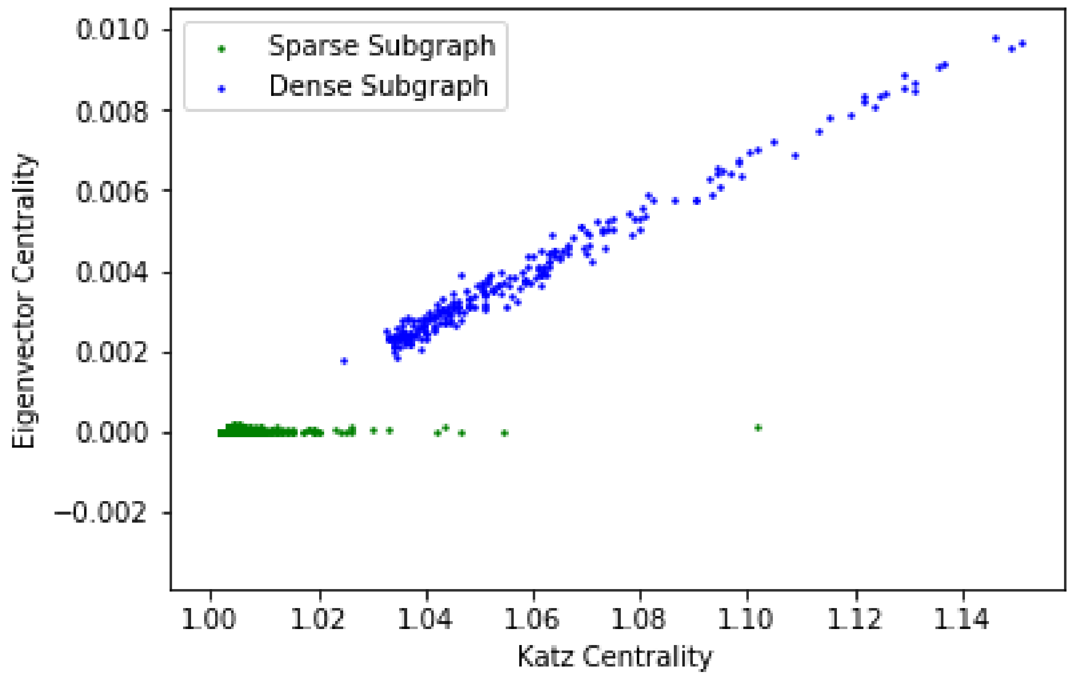
```
            return centrality

    evc = zen.algorithms.eigenvector_centrality_(G)
    kc = katz(G)
    plt.scatter(evc[:Size1],kc[:Size1],c='g',s=2,label='Sparse Subgraph')
    plt.scatter(evc[Size1:],kc[Size1:],c='b',s=2,label='Dense Subgraph')
    plt.xlabel('Eigenvector Centrality')
    plt.ylabel('Katz Centrality')
    plt.legend()
    plt.show()
```



This resulting plot clearly shows distinct behavior in the two classes of nodes. A linear regression of each class's plot will yeild different slopes. But the separation of the classes in this resulting plot is different from the initial plot. The initial plot showed a scaled and translation relationship between the classes. This resulting plot shows a scaled and rotation relationship.

Interestingly, however, the plot of eigenvector vs katz centrality more closely resembles the initial plot.

```
In [287]: plt.scatter(kc[:Size1],evc[:Size1],c='g',s=2,label='Sparse Subgraph')
          plt.scatter(kc[Size1:],evc[Size1:],c='b',s=2,label='Dense Subgraph')
          plt.ylabel('Eigenvector Centrality')
          plt.xlabel('Katz Centrality')
          plt.legend()
          plt.show()
```

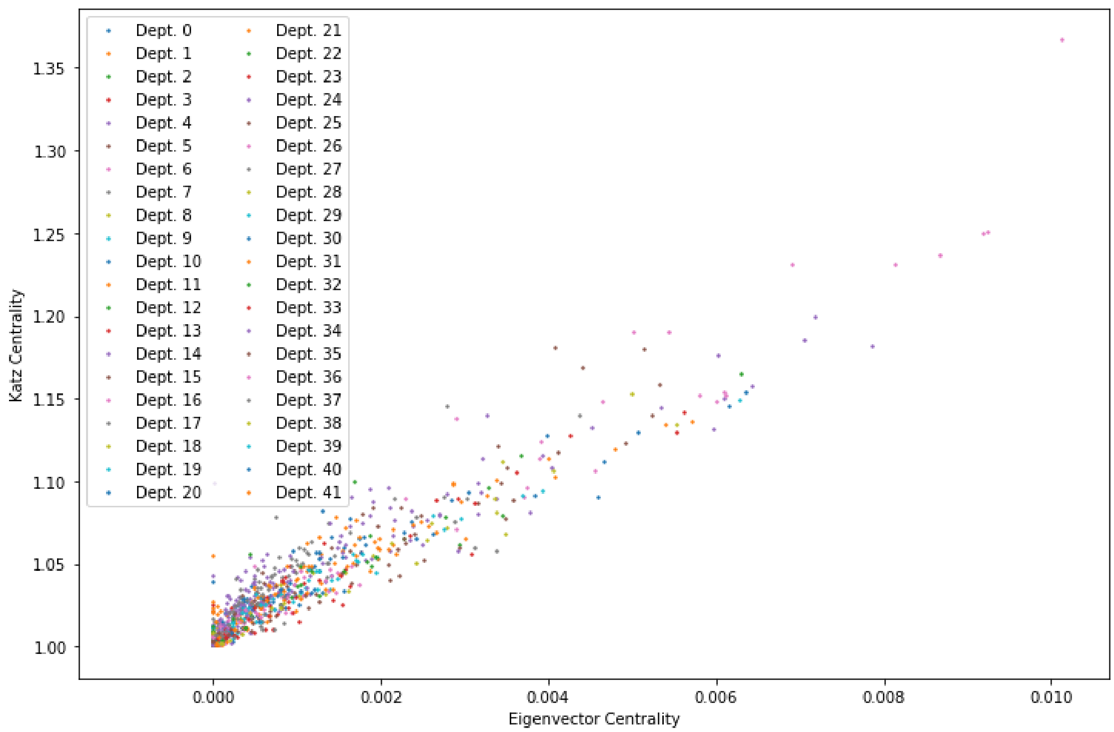


It is unlikely that a mistake was made in the axis labeling in the initial plot, since the scale of the axis match with what is expected. Eigenvector centralities are typically much less than 1, where as Katz centralities are bounded below by β (set to 1 in this case).

Now looking at a real world network with pre-labeled communities. The Eu-core network is an email network that "contains 'ground-truth' community memberships of the nodes. Each individual belongs to exactly one of 42 departments at the research institute". Plotting katz centrality against eigenvector centrality for each department may show the clustering phenomenon seen above.

```
In [145]: import pandas as pd
          Department = pd.read_csv('email-Eu-core-department-labels.txt',delimiter=' ',\
                                   header=None,names=['Node','Department'])

          departmentGroupings = Department.groupby('Department').apply(lambda x: x['Node'])

          Communities = {}
          for dept in departmentGroupings.index.get_level_values(0).unique():
              Communities[dept] = departmentGroupings.loc[dept].values.astype(str)
          Department.set_index('Node',inplace=True)

In [130]: G_eu_email = zen.io.edgelist.read('email-Eu-core.txt',directed=True) # as undirected s

          evc = zen.centrality.eigenvector_centrality_(G_eu_email)
          kc = katz(G_eu_email)
```

```
In [137]: depts = Department['Department'].unique()
          depts.sort()
          fig = plt.figure(figsize=(12,8))
          for dept in depts:
              nodes = Department[Department['Department']==dept].index.values.tolist()
              plt.scatter(evc[nodes],kc[nodes],s=2,label='Dept. %d'%dept)
          plt.legend(ncol=2)
          #plt.gca().set_xscale('log')
          #plt.gca().set_yscale('log')
          plt.xlabel('Eigenvector Centrality')
          plt.ylabel('Katz Centrality')
          plt.show()
```



Here there is not a lot of distinguishability between the different departments. All nodes tend to fall in the same overall trend. This may be because the network is not strongly assortatively mixed

```
In [155]: G_eu_skel = G_eu_email.skeleton()

          Q = zen.algorithms.modularity(G_eu_skel,Communities)

          # Maximum Modularity
          count=0.0
          for e in G_eu_skel.edges():
              if Department.loc[int(e[0]),'Department'] == Department.loc[int(e[1]),'Department'
```

```
            count += 1
        same = count / G_eu_skel.num_edges
        rand = same - Q
        qmax = 1 - rand

        print 'Modularity:      %.3f'%Q
        print 'Max. Modularity: %.3f'%qmax
        print 'Normalized Mod:  %.3f'%(Q/qmax)
```

```
Modularity:      0.314
Max. Modularity: 0.953
Normalized Mod:  0.329
```

The preivous networks that demonstrated the distinct clusters in the Katz vs eigenvector centrality had normalized nodularity values closer to 1 than is seen in the Eu-core network.

Let's take another larger network from SNAP with pre-determined communities, looking only at the subgraph of nodes belonging to the two largest communities. "The DBLP computer science bibliography provides a comprehensive list of research papers in computer science. We construct a co-authorship network where two authors are connected if they publish at least one paper together."

```
In [162]: dblp_df = pd.DataFrame(columns=['Node','Community'])

          idx=0
          comm=0
          for line in log_progress(open('com-dblp.top5000.cmty.txt'),every=1,size=5000):
              nodes = line.split('\t')
              for node in nodes:
                  dblp_df.loc[idx] = [int(node), comm]
                  idx+=1
              comm+=1

          dblp_df.head()
```

```
VBox(children=(HTML(value=u''), IntProgress(value=0, max=5000)))
```

Out[162]:

	Node	Community
0	105653	0
1	105654	0
2	210737	0
3	210738	0
4	210739	0

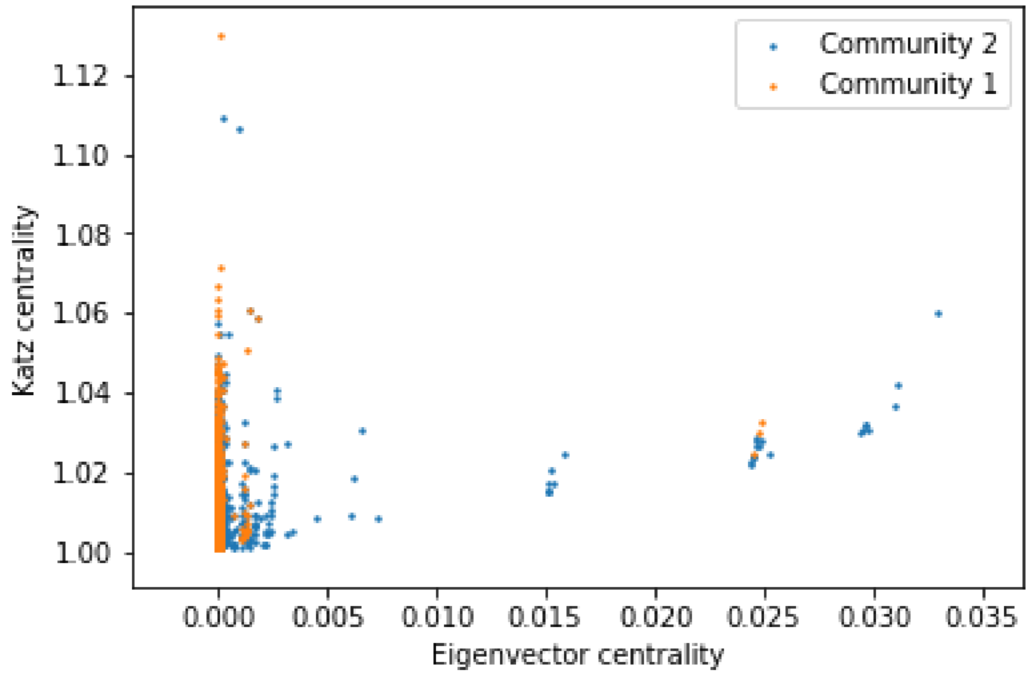
```
In [180]: TwoLargestCom = dblp_df.groupby('Community').count().sort_values('Node',ascending=Fals

          sampled_nodes=dblp_df[(dblp_df['Community']==TwoLargestCom[0])|\
                                (dblp_df['Community']==TwoLargestCom[1])].set_index('Node')

In [224]: G = zen.Graph()
          for line in log_progress(open('com-dblp.ungraph.txt'),every=1,size=1049870):
              info = line.split('\t')
              n1 = int(info[0])
              n2 = int(info[1])

              node1_in = True
              try:
                  _ = sampled_nodes.loc[n1,:]
              except KeyError:
                  # not in comm
                  node1_in = False

              node2_in = True
              try:
                  _ = sampled_nodes.loc[n2,:]
              except KeyError:
                  # not in comm
                  node2_in = False

              if node1_in and node2_in:
                  if not G.has_edge(n1,n2):
                      G.add_edge(n1,n2)

VBox(children=(HTML(value=u''), IntProgress(value=0, max=1049870)))


In [216]: communities = sampled_nodes.groupby('Community').apply(lambda x: x.iloc[:,0])
          comm_names = communities.index.get_level_values(0).unique()

          C = {}
          for name in comm_names:
              C[name] = communities.loc[name].index.values

In [243]: Q = zen.algorithms.modularity(G,C)

          # Maximum Modularity
          count=0.0
          for e in log_progress(G.edges(),every=1):
              c1 = sampled_nodes.loc[e[0],'Community']
              c2 = sampled_nodes.loc[e[1],'Community']
              if type(c1) == pd.core.series.Series or type(c2) == pd.core.series.Series:
                  count += 1
              elif sampled_nodes.loc[e[0],'Community'] == sampled_nodes.loc[e[1],'Community']:
```

```
                count += 1
        same = count / G.num_edges
        rand = same - Q
        qmax = 1 - rand

        print 'Modularity:      %.3f'%Q
        print 'Max. Modularity: %.3f'%qmax
        print 'Normalized Mod:  %.3f'%(Q/qmax)
```

```
VBox(children=(HTML(value=u''), IntProgress(value=0, max=34281)))


Modularity:      0.438
Max. Modularity: 0.468
Normalized Mod:  0.936
```

```
In [255]: evc = zen.algorithms.eigenvector_centrality_(G)
          kc = katz(G)
          cc = zen.algorithms.clustering.lcc_(G)
```

```
In [267]: comms = communities.index.get_level_values(0).unique()
          # comm 1
          c1nodes = communities.loc[comms[0],:].index.get_level_values(1).values
          c1nodeIdx = np.zeros(len(c1nodes),dtype=np.int64)
          for i,x in enumerate(c1nodes):
              idx = G.node_idx(x)
              c1nodeIdx[i] = idx

          # comm 2
          c2nodes = communities.loc[comms[1],:].index.get_level_values(1).values
          c2nodeIdx = np.zeros(len(c2nodes),dtype=np.int64)
          for i,x in enumerate(c2nodes):
              idx = G.node_idx(x)
              c2nodeIdx[i] = idx

          plt.scatter(evc[c2nodeIdx],kc[c2nodeIdx],s=2,label='Community 2')
          plt.scatter(evc[c1nodeIdx],kc[c1nodeIdx],s=2,label='Community 1')
          plt.legend()
          plt.xlabel('Eigenvector centrality')
          plt.ylabel('Katz centrality')
          plt.show()
```



The graph more closely results from the first synthetic graph, with a distinction. Ine the first results, the denser of the two communities corresponded to the cluster with the smaller linear slope. However, Community 1 has a slightly larger average local clustering coefficient than Community 2, yet has a steeper linear slope.

```
In [260]: avgCC1 = np.mean(cc[c1nodeIdx])
          avgCC2 = np.mean(cc[c2nodeIdx])

          print 'Community 1 average lcc: %.3f'%avgCC1
          print 'Community 2 average lcc: %.3f'%avgCC2

Community 1 average lcc: 0.638
Community 2 average lcc: 0.576
```

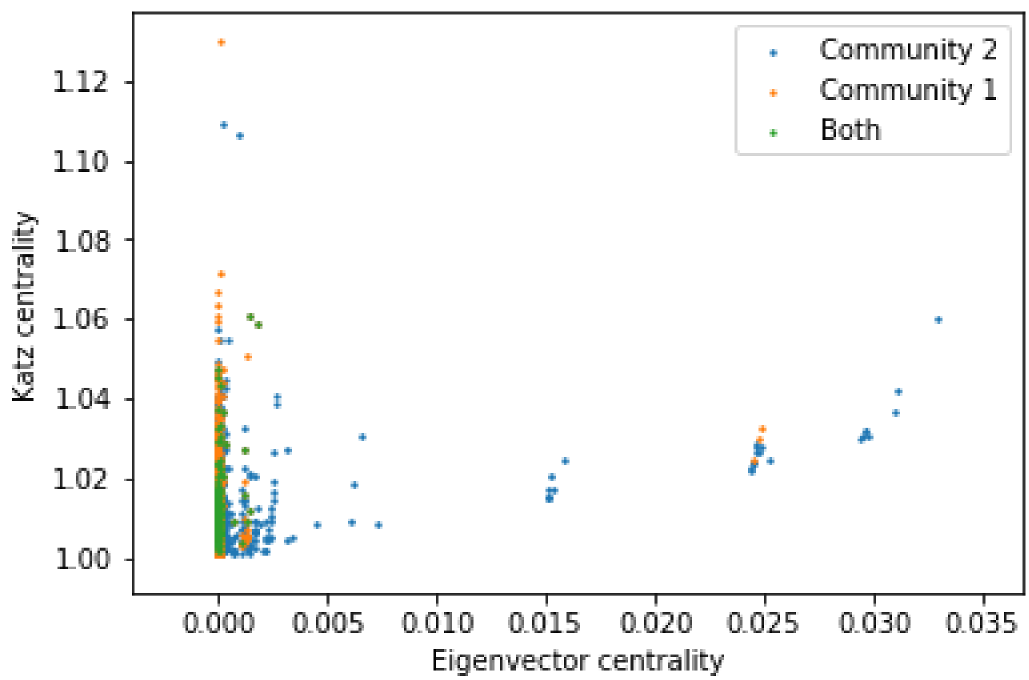
There is much more of a distinction between the two communities in the centralities plot than seen in the EU-core network. The graph more closely results from the first synthetic graph. That the DBLP graph is strongly assortively mixed support the claim that distinct clusters in the Katz vs Eigenvector centrality plot only appear with strong assortative mixing. However, there are several nodes that appear in regions dominated by the opposite community. This might be explained by one or more of the following details about the network:

1. There are several nodes that are members of both communities. "Publication venue, e.g, journal or conference, defines an individual ground-truth community; authors who published to a certain journal or conference form a community." Authors frequently publish in numerous journals and conferences, causing community membership not to be disjoint.

2. Venue is not the only basis for ground-truth communities in the network. "We regard each connected component in a group as a separate ground-truth community." One of the communities sampled here may not be a publication venue community, but a connected component, hence the overlap in membership.

3. The different communities suggested by the centralities plot are different from the communities defined by publication venue/connected component.

In [279]:
```
countedCom = sampled_nodes.groupby('Node').count()
doubled_nodes = countedCom.where(countedCom > 1).dropna().index.values

doubled_node_idx = [G.node_idx(node) for node in doubled_nodes]

plt.scatter(evc[c2nodeIdx],kc[c2nodeIdx],s=2,label='Community 2')
plt.scatter(evc[c1nodeIdx],kc[c1nodeIdx],s=2,label='Community 1')
plt.scatter(evc[doubled_node_idx],kc[doubled_node_idx],s=2,label='Both')
plt.legend()
plt.xlabel('Eigenvector centrality')
plt.ylabel('Katz centrality')
plt.show()
```



The "out of place" nodes are **not** the nodes in both communities. Thus, it seems that the third explaination is more likely.