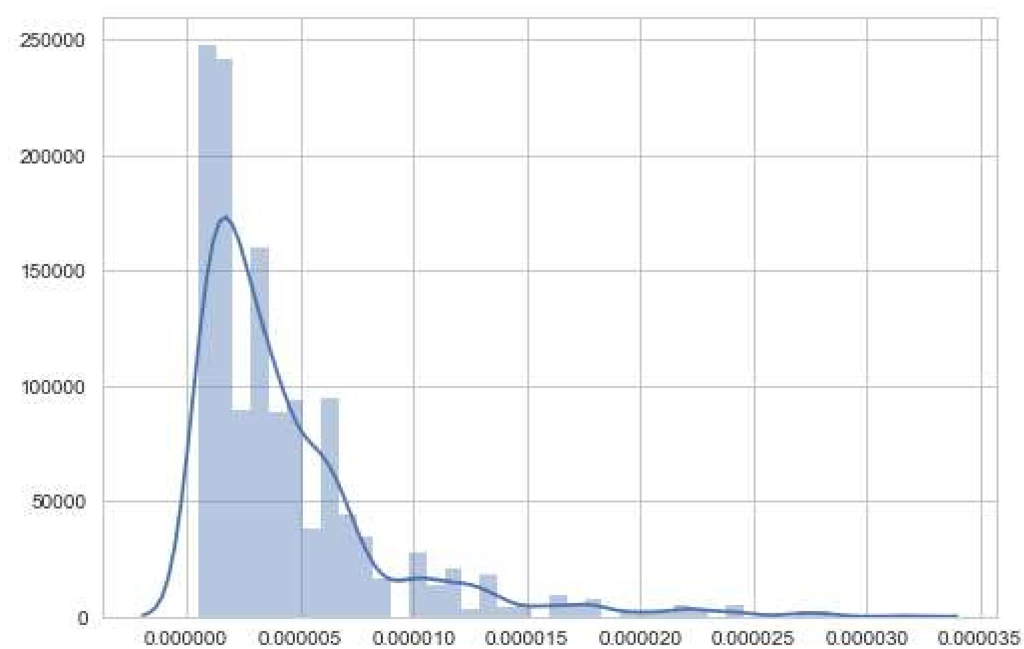
In [21]:
```
all_edge_betweeness_values = edge_betweeness.values()

for percentile in [50, 60, 70, 80, 90, 95]:
    print np.percentile(all_edge_betweeness_values, percentile)

sns.distplot(all_edge_betweeness_values)
```

```
3.00150075038e-06
4.0020010005e-06
5.00250125063e-06
6.50325162581e-06
1.00050025013e-05
1.30065032516e-05
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0xe6b9208>



In [22]:
```
#edges_to_remove = [e for e, val in edge_betweeness.items() if val < np.percenti
#G.remove_edges_from(edges_to_remove)
print G.number_of_nodes(), G.number_of_edges()
```

```
2000 1562
```

In [54]:
```
def rescale_layout(pos,scale=1.):
    maxlim=0
    for i in range(pos.shape[1]):
        pos[:,i]-= pos[:,i].min()
        maxlim=max(maxlim,pos[:,i].max())
    if maxlim>0:
        for i in range(pos.shape[1]):
            pos[:,i]*=scale/maxlim
    return pos

def network_layout(G,iter,repulstype,repulsfactor):
    adjmat=nx.adjacency_matrix(G)
    adjmat=adjmat.todense()
    a,b=adjmat.shape
    adjmat=np.asarray((adjmat),dtype='float64')
    iterations=iter
    pos_array=np.random.random((a,2))
    pos=pos_array
    pos=pos.astype(adjmat.dtype)
    k=np.sqrt(1.0/a)
    t = max(max(pos.T[0]) - min(pos.T[0]), max(pos.T[1]) - min(pos.T[1]))*0.1
    dt=t/float(iterations+1)
    delta = np.zeros((pos.shape[0],pos.shape[0],pos.shape[1]),dtype=adjmat.dtype
    degree=nx.degree_centrality(G)
    degree=np.array(degree.values(),dtype='float64')
    degreemat=[[i*j for i in degree]for j in degree]
    for iteration in range(iterations):
        for i in range(pos.shape[1]):
            delta[:,:,i]=pos[:,i,None]-pos[:,i]
        delta=np.array(delta,dtype=np.float64)
        #if magnitude of delta<R(k), global force=0
        distance=np.sqrt((delta**2).sum(axis=-1))
        distance=np.where(distance<0.01,0.01,distance)


        if repulstype =='edgespring1':
            edgeforce=degree/distance**2
            # old version used in jupyter
            #edgeforce=edgeforce/edgefactor
            force1=k*k/distance**2+edgeforce
            force1=force1/repulsfactor
        if repulstype == 'edgespring2':
            force_edge=degreemat/(distance**2)
            # old version used in jupyter
            #force1=force_edge/edgefactor+(k*k/distance**2)/repulsfactor
            force1=force_edge+(k*k/distance**2)/repulsfactor
        if repulstype == 'trial':
            force_edge=degreemat/(distance**2)
            # old version used by jupyter
            #force1=force_edge/edgefactor
            force1=force_edge
        if repulstype == 'linlog':
            force_edge=degreemat*np.log(distance)
            force1=-1*force_edge
        if repulstype == 'noedge':
            force1=k*k/distance**2
```

```
            force1*=1/repulsfactor
        if repulstype == 'spring':
            force1=k*k/distance
            force1*=1/repulsfactor

        #attraction function
        force2=adjmat*distance/k
        forces=force1-force2
        displacement=np.transpose(np.transpose(delta)*(forces)).sum(axis=1)
        length=np.sqrt((displacement**2).sum(axis=1))
        #in order to avoid dividing by zero or a v small number
        length=np.where(length<0.01,0.01,length)
        #does displacement
        delta_pos=np.transpose(np.transpose(displacement)*t/length)
        pos+=delta_pos
        t-=dt
        #for no fixed nodes only
        pos=rescale_layout(pos)
        #if delta_pos<tolerance
        #break loop, system has converged

    return pos, G
```

In [55]:

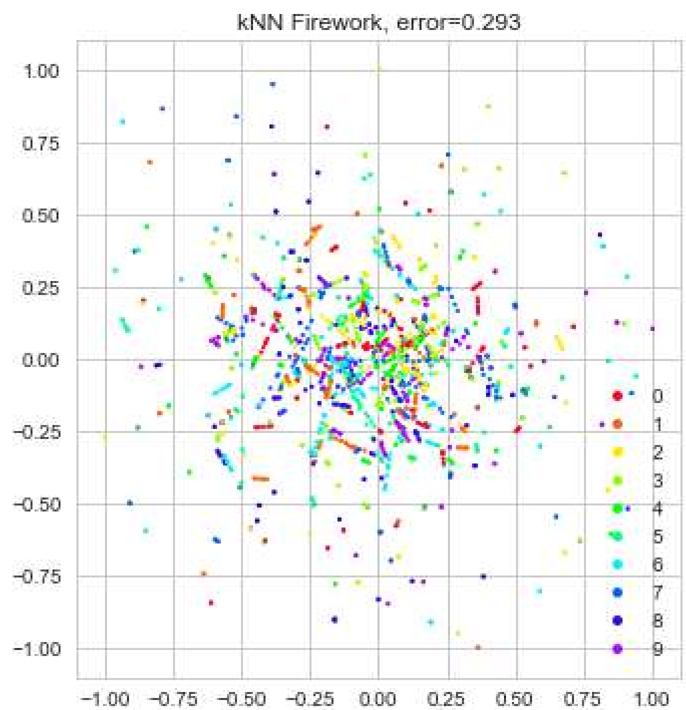
```
X_knn_fw, G_cy = network_layout(G,50,'linlog',0)
#print X_knn_fw.shape
#print(NN_generalization_error(network_layout(G,50,10,1000)[0],labels))
```

In [56]:

```
# Display the current layout from Cytoscape
#display.Image(G_cy.get_png())
```

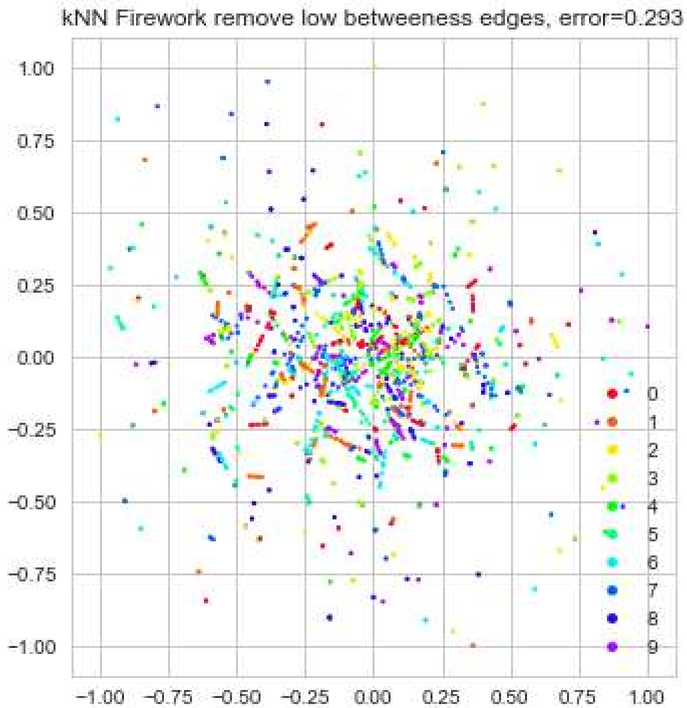
In [57]:
```
error = NN_generalization_error(X_knn_fw, labels)
ax = plot_embed(X_knn_fw, labels)
ax.set_title('kNN Firework, error=%.3f' % error)
#check repuls force
error = [np.average([NN_generalization_error(network_layout(G,50,'noedge',i)[0],
#error2 = [np.average([NN_generalization_error(network_layout(G,50,'trial2',i,j)

print(error)
#print(error2)

#check betweeness cutoff
```

```
[0.78896382090919293]
```



In [58]:
```
error = NN_generalization_error(X_knn_fw, labels)
ax = plot_embed(X_knn_fw, labels)
ax.set_title('kNN Firework remove low betweeness edges, error=%.3f' % error)
```

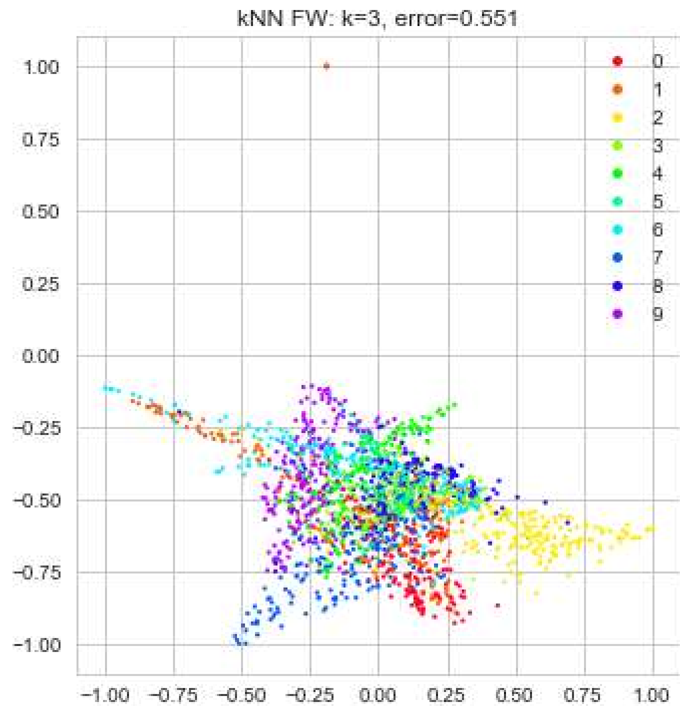
Out[58]: <matplotlib.text.Text at 0x236c70f0>

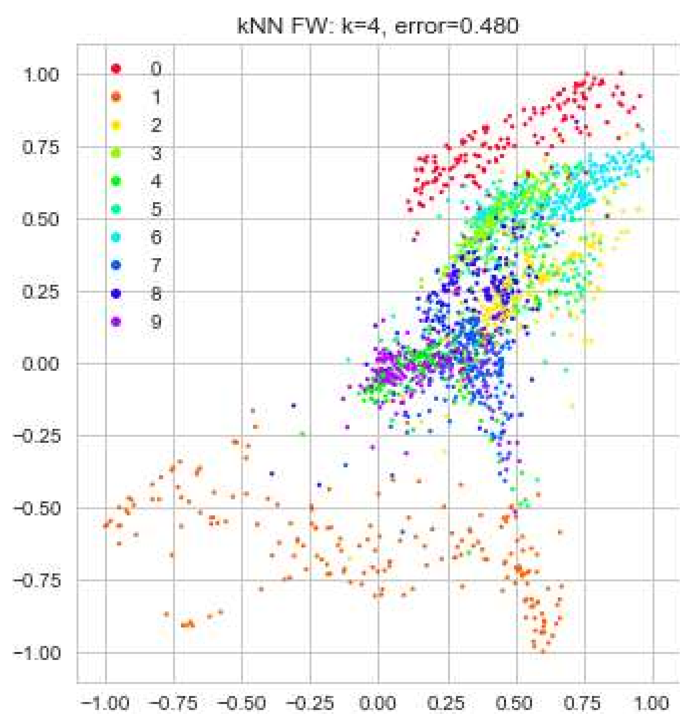


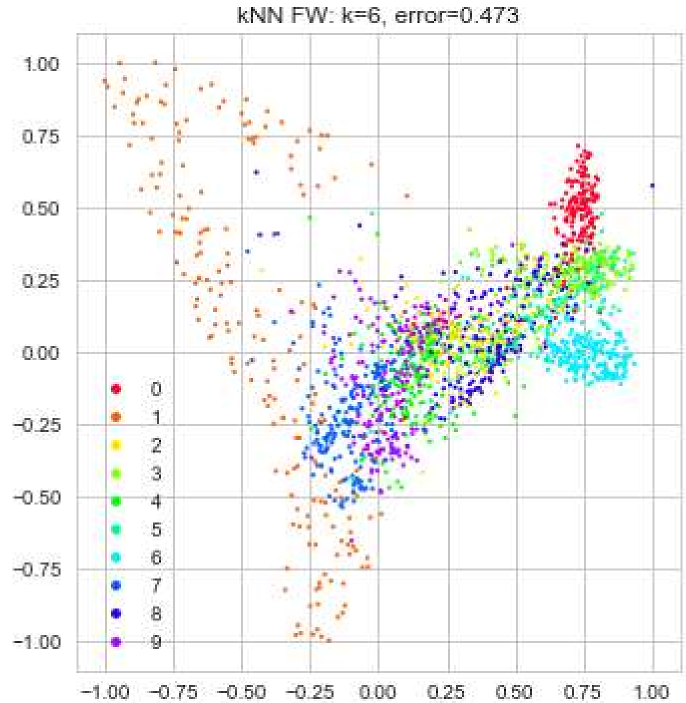
In [59]:
```
#for k in [1, 5, 10, 20, 30, 40, 50, 100]:
#    G = create_knn_graph(X_pca[:, :50], k=k)
#    X_knn_fw, G_cy = network_layout(G,50)
#    error = NN_generalization_error(X_knn_fw, labels)
#   ax = plot_embed(X_knn_fw, labels)
#    ax.set_title('kNN FW: k=%d, error=%.3f' % (k, error))
```

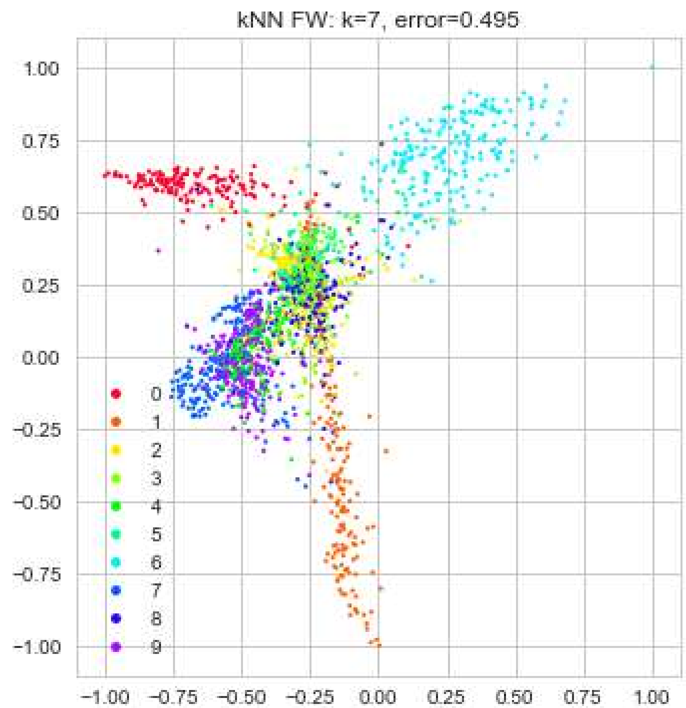
In [60]:

```
for k in [3, 4, 6, 7]:
    G = create_knn_graph(X_pca[:, :50], k=k)
    X_knn_fw, G_cy = network_layout(G,50,'linlog',0)
    error = NN_generalization_error(X_knn_fw, labels)
    ax = plot_embed(X_knn_fw, labels)
    ax.set_title('kNN FW: k=%d, error=%.3f' % (k, error))
```









Perform thresholding to form graphs for the Firework layout

In [61]:
```
# thresholding-Firework
def create_graph_by_threshold(adj_mat, percentile):
    triu_idx = np.tril_indices(adj_mat.shape[0], 1)
    thresold = np.percentile(adj_mat[triu_idx], percentile)
    adj_mat_ = adj_mat.copy()
    adj_mat_[adj_mat<thresold] = 0
    G = nx.from_numpy_matrix(adj_mat_)
    return G

def create_graph_by_threshold_knn(adj_mat, percentile, k=1, X=None):
    '''combine the graph from `create_graph_by_threshold` with a kNN graph.
    '''
    G_thres = create_graph_by_threshold(adj_mat, percentile)
    G_knn = create_knn_graph(X, k=k)
    return nx.compose(G_thres, G_knn)
```

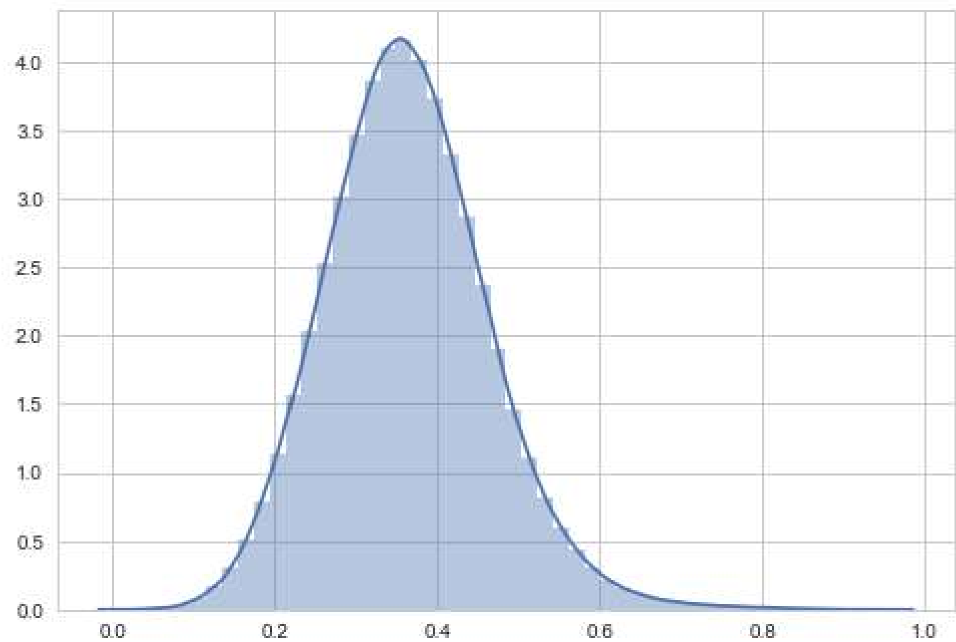
In [62]:
```
adj_mat = compute_adjcency_mat(X_pca[:, :50], metric='euclidean')
print adj_mat.shape
adj_mat[:5, :5]
```

```
(2000L, 2000L)
```

Out[62]:
```
array([[ 0.        ,  0.43427314,  0.36176446,  0.25112794,  0.37515545],
       [ 0.43427314,  0.        ,  0.35054958,  0.15599296,  0.31175289],
       [ 0.36176446,  0.35054958,  0.        ,  0.3351092 ,  0.33283703],
       [ 0.25112794,  0.15599296,  0.3351092 ,  0.        ,  0.20630268],
       [ 0.37515545,  0.31175289,  0.33283703,  0.20630268,  0.        ]])
```

In [63]:
```
# Histogram for values in adj_mat
triu_idx = np.triu_indices(adj_mat.shape[0], 1)
sns.distplot(adj_mat[triu_idx])
```

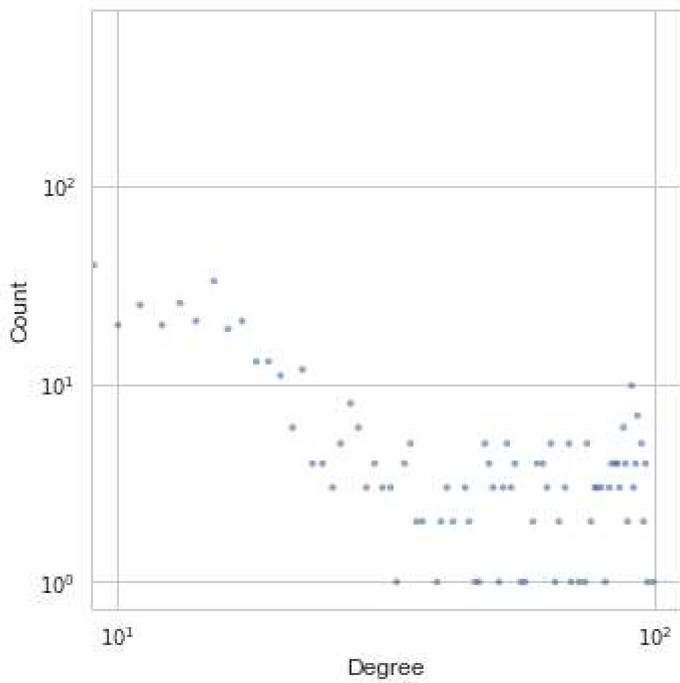
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x230c24e0>



In [64]:
```
percentile = 99.5
# percentile = 90
G = create_graph_by_threshold(adj_mat, percentile)
print G.number_of_nodes(), G.number_of_edges()
plot_degree_distribution(G)
```

2000 10007

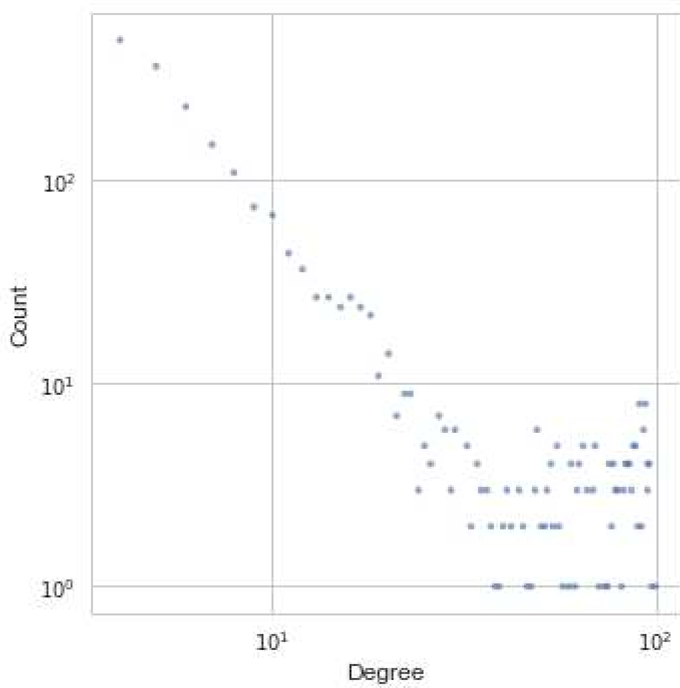
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x282351d0>



In [65]:
```
k = 4
G_thres_knn = create_graph_by_threshold_knn(adj_mat, percentile, k=k, X=X_pca[:,
print G_thres_knn.number_of_edges(), G_thres_knn.number_of_nodes()
plot_degree_distribution(G_thres_knn)
```

13017 2000

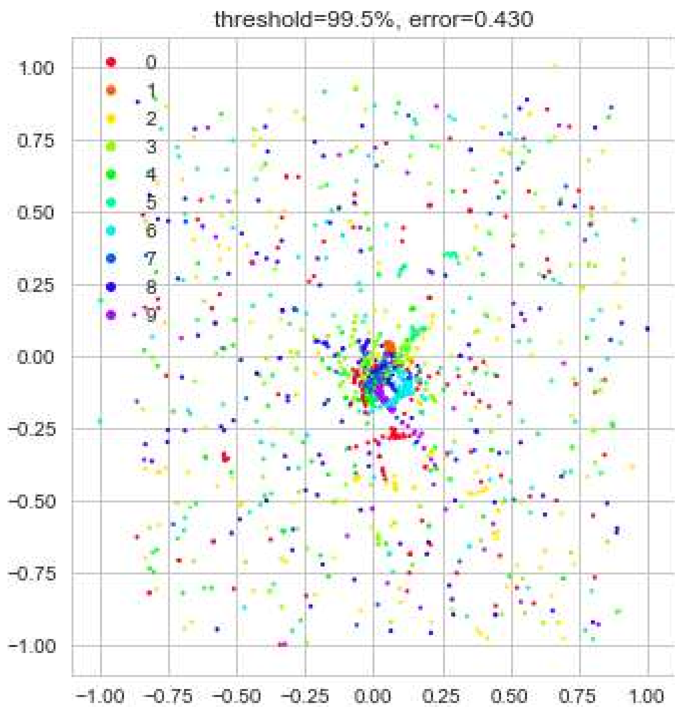
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1d72d940>

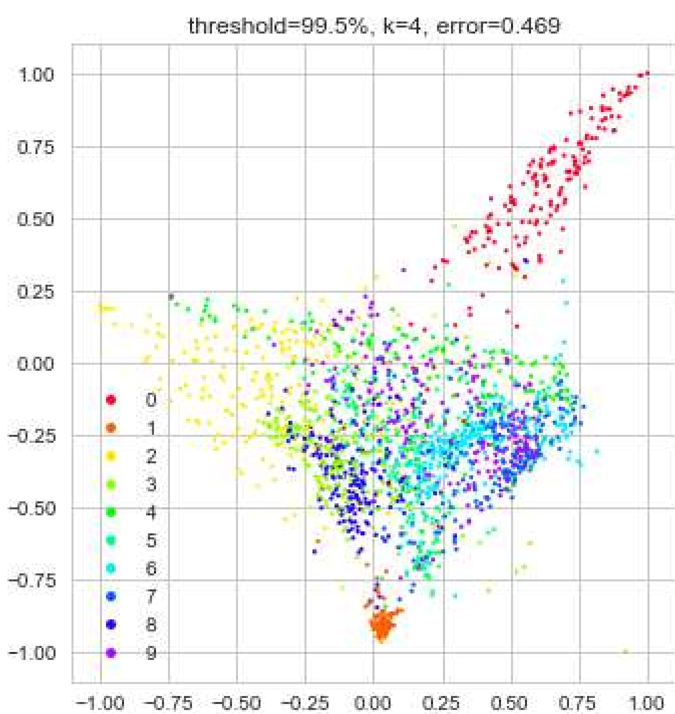


In [66]:
```
X_thre_fw, G_cy = network_layout(G,50,'linlog',0)
X_threknn_fw, G_cy = network_layout(G_thres_knn,50,'linlog',0)
```

In [67]:
```
ax = plot_embed(X_thre_fw, labels)
error = NN_generalization_error(X_thre_fw, labels)
ax.set_title('threshold=%.1f%%, error=%.3f' %(percentile, error))

ax = plot_embed(X_threknn_fw, labels)
error = NN_generalization_error(X_threknn_fw, labels)
ax.set_title('threshold=%.1f%%, k=%d, error=%.3f' %(percentile, k, error))
```

Out[67]: <matplotlib.text.Text at 0x1d716208>





In [68]:
```
display.Image(G_cy.get_png())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-68-72fcd72c72e7> in <module>()
----> 1 display.Image(G_cy.get_png())

AttributeError: 'Graph' object has no attribute 'get_png'
```

Problem: there are some small connected components and even isolated nodes, how to deal with them?

Current ~~solution~~: exclude

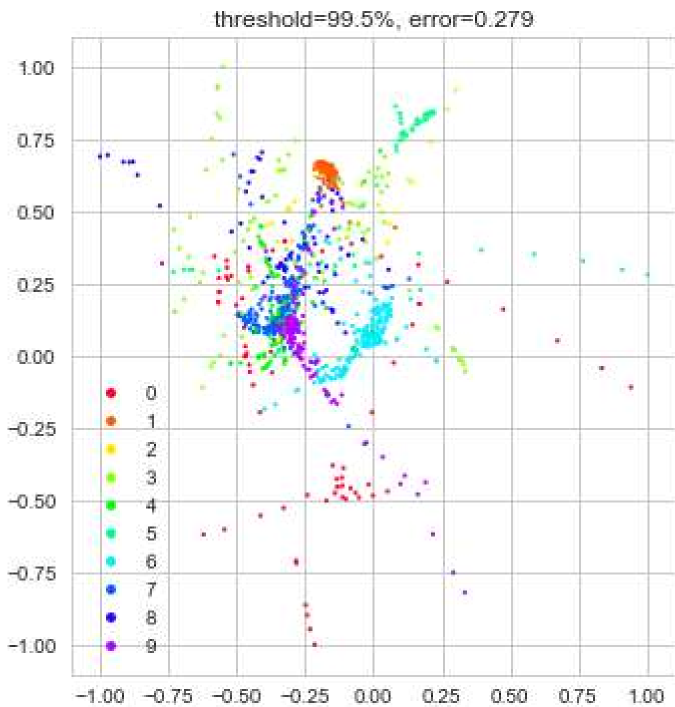
In [70]:
```
G_new = nx.Graph()
G_new_tknn = nx.Graph()
for cc in nx.connected_component_subgraphs(G):
    if cc.number_of_nodes() > 10:
        G_new = nx.compose(G_new, cc)
print G_new.number_of_nodes(), G_new.number_of_edges()

for cc in nx.connected_component_subgraphs(G_thres_knn):
    if cc.number_of_nodes() > 10:
        G_new_tknn = nx.compose(G_new_tknn, cc)
print G_new_tknn.number_of_nodes(), G_new_tknn.number_of_edges()
```

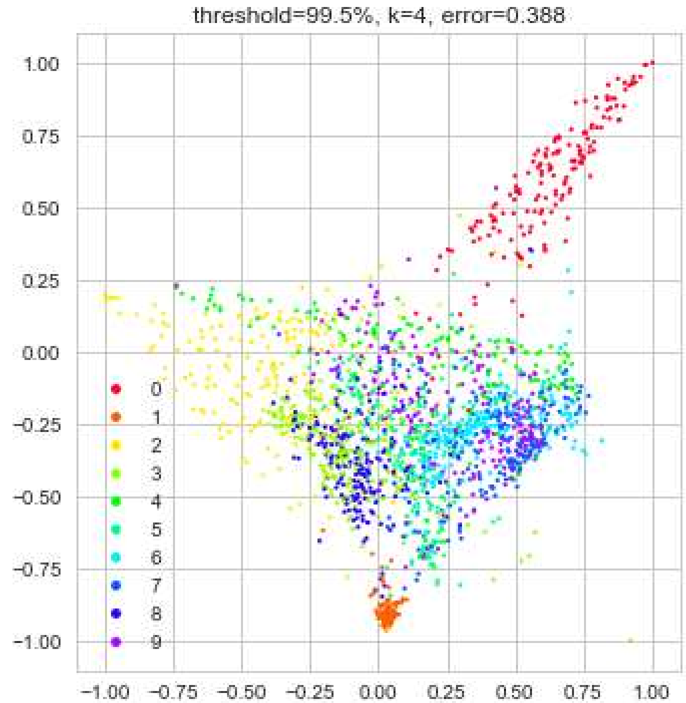
```
1167 9812
2000 13017
```

To make a fair comparison between thresholding-Firework and t-SNE, re-compute the error by ignoring nodes in the small connected components.

In [71]:
```
# Re-compute the error by ignoring the small CCs
node_idx_in_graph = G_new.nodes()
# node_idx_in_graph = G_new_Gknn.nodes()
sample_mask = np.in1d(np.arange(N), node_idx_in_graph)

ax = plot_embed(X_thre_fw[sample_mask], labels[sample_mask])
error = NN_generalization_error(X_thre_fw[sample_mask], labels[sample_mask])
ax.set_title('threshold=%.1f%%, error=%.3f' %(percentile, error))

ax = plot_embed(X_threknn_fw, labels)
error = NN_generalization_error(X_threknn_fw[sample_mask], labels[sample_mask])
ax.set_title('threshold=%.1f%%, k=%d, error=%.3f' %(percentile, k, error))
```

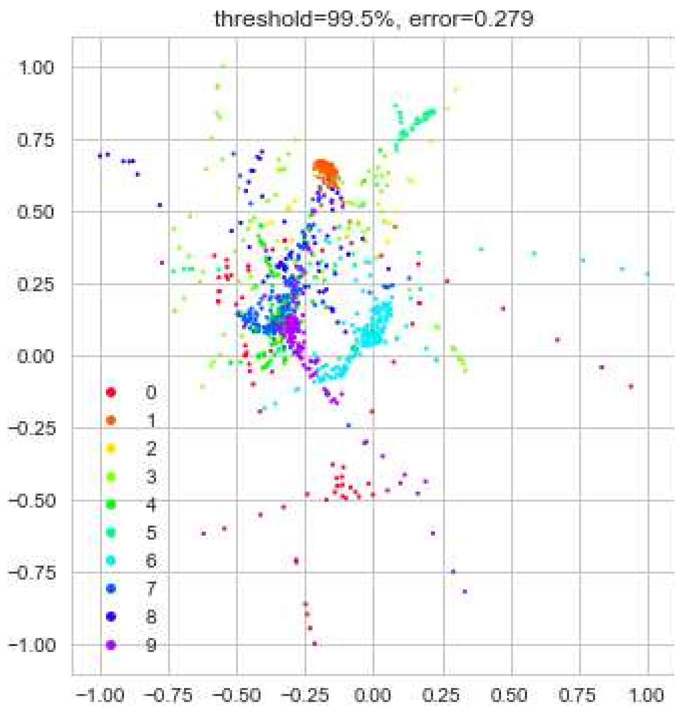
Out[71]: <matplotlib.text.Text at 0xe55dd30>

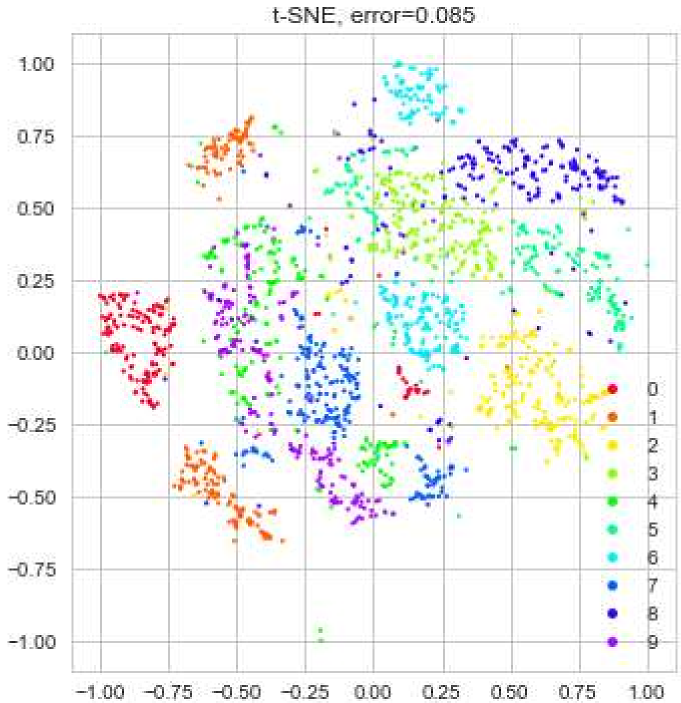



threshold=99.5%, k=4, error=0.388
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
7
8
9
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

In [72]:
```
# Re-compute the error by ignoring the small CCs
node_idx_in_graph = G_new.nodes()
sample_mask = np.in1d(np.arange(N), node_idx_in_graph)

ax = plot_embed(X_thre_fw[sample_mask], labels[sample_mask])
error = NN_generalization_error(X_thre_fw[sample_mask], labels[sample_mask])
ax.set_title('threshold=%.1f%%, error=%.3f' %(percentile, error))

ax = plot_embed(X_tsne, labels)
error = NN_generalization_error(X_tsne[sample_mask], labels[sample_mask])
ax.set_title('t-SNE, error=%.3f' %error)
```

Out[72]: <matplotlib.text.Text at 0x1d233978>





References

- Making sense of principal component analysis, eigenvectors & eigenvalues (https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues)
- MNIST For ML Beginners (https://www.tensorflow.org/get_started/mnist/beginners)
- Visualizing MNIST: An Exploration of Dimensionality Reduction (http://colah.github.io/posts/2014-10-Visualizing-MNIST/)
- Tensorflow Embedding Projector (http://projector.tensorflow.org/)
- van der Maaten's t-SNE page (http://lvdmaaten.github.io/tsne/)
- van der Maaten et al.: Dimensionality Reduction: A Comparative Review (https://www.tilburguniversity.edu/upload/59afb3b8-21a5-4c78-8eb3-6510597382db_TR2009005.pdf)
- van der Maaten: Learning a Parametric Embedding by Preserving Local Structure (http://lvdmaaten.github.io/publications/papers/AISTATS_2009.pdf)
- Kokiopoulou and Saad: Enhanced graph-based dimensionality reduction with repulsion Laplaceans (http://www.sciencedirect.com/science/article/pii/S0031320309001460)