Deshpande-Charudatta-PS2

January 19, 2018

In [121]:
```
# Student Name - Charudatta Deshpande
#
# Collaborators - Ram Ganesan, Charles Hemstreet, Mehdi Muntazir
#
#############################################
# Import required libraries.                #
#############################################
#
import pandas as pd
import numpy as np
import scipy as sp
import networkx as nx
from sklearn.preprocessing import normalize
from numpy import *
import os as os
import time
os.chdir('C:\\Users\deshc\Desktop\INFX 574 Data Science 2\Problem Set 2')
```

In [122]:
```
#
#############################################
# Step 1 - Input data. Create the matrix    #
# as specified in the assignment.           #
#############################################
#
A=np.matrix([[1,0,2,0,4,3],[3,0,1,1,0,0],[2,0,4,0,1,0],[0,0,1,0,0,1],[8,0,3,0,5,2],[0
#
#############################################
# Step 2 - Zero out diagonal values         #
# as specified in the assignment.           #
#############################################
#
np.fill_diagonal(A, 0)
#
#############################################
# Step 3 - Normalize the matrix (divide by  #
# sum of the column values)                 #
# as specified in the assignment.           #
```

```
############################################
#
H = normalize(A, norm='l1', axis=0)
```

In [123]:
```
#
############################################
# Step 4 - Create a dangling node matrix d  #
# as specified in the assignment.           #
############################################
#
column_sum = H.sum(axis=0) == 0
d = 1*column_sum
```

In [124]:
```
#
############################################
# Step 5 - Create article vector. For now   #
# hardcode this with values specified.      #
############################################
#
Art_Vector=np.array([3/14, 2/14, 5/14, 1/14, 2/14, 1/14])
```

In [125]:
```
#
############################################
# Step 6 - Create initial start vector.     #
# This can be controlled by providing value #
# of 'n'.                                   #
############################################
#
n=H.shape[0]
start_vector = np.repeat((1/n), n)
```

In [126]:
```
#
############################################
# Step 7 - Iterate given equation to calcu- #
# te influence matrix. Set max_iter to appr.#
# value.                                    #
############################################
#
max_iter = 50
alpha = 0.85
epsilon = 0.00001

def eigen_function(H,d,start_vector,max_iter):
    for i in range(max_iter):
        prev_start_vector = start_vector
        start_vector = (alpha * np.dot(H,start_vector))+ np.dot(((alpha * np.dot(d,s
        start_vector_max = np.abs(start_vector).max()
        if start_vector_max == 0.0:
            start_vector_max = 1.0
```

```
        err = np.abs(start_vector - prev_start_vector).max() / start_vector_max
        if err < epsilon:
            return start_vector
    return start_vector
test_matrix_result = eigen_function(H,d,start_vector,max_iter)
test_matrix_result
```

Out[126]:
```
array([ 0.30402333,  0.1636025 ,  0.1897965 ,  0.04661903,  0.27531131,
        0.02064733])
```

In [127]:
```
#
#############################################
# Step 8 - Calculate Eigenfactor (EFi) from #
# given formula                             #
#############################################
#
B = np.dot(H,test_matrix_result)
C = np.sum(B, axis=0)
EF = 100*B/C
EF
```

Out[127]:
```
array([ 34.05082911,  17.2037876 ,  12.17544066,   3.65316819,
        32.91677445,   0.        ])
```

In [128]:
```
#
#############################################
# Step 9 - Read links file                  #
# Convert into adjacency matrix             #
# Convert NA to zero                        #
# Convert to squared pivot table by using   #
# reindex function.                         #
# The convert to array format.              #
#############################################
#
links = pd.read_csv('links.txt', delimiter=',', header=None).pivot(0,1,2)
where_are_NaNs = isnan(links)
links[where_are_NaNs] = 0
new_links = links
index = new_links.index.union(test.columns)
new_links = new_links.reindex(index=index, columns=index, fill_value=0)
new_links_matrix = new_links.as_matrix()
```

In [129]:
```
#
#############################################
# Step 10 - Zero out diagonal values         #
# as specified in the assignment.           #
#############################################
#
np.fill_diagonal(new_links_matrix, 0)
```

```
#
############################################
# Step 11 - Normalize the matrix (divide by  #
# sum of the column values)                 #
# as specified in the assignment.           #
############################################
#
H = normalize(new_links_matrix, norm='l1', axis=0)
```

In [130]:
```
#
############################################
# Step 12 - Create a dangling node matrix d  #
# as specified in the assignment.           #
############################################
#
column_sum = H.sum(axis=0) == 0
d = 1*column_sum
```

In [131]:
```
#
############################################
# Step 13 - Create article vector. For this #
# assume that all journals publish 1 article#
############################################
#
n=H.shape[0]
Art_Vector=np.repeat((1/n), n)
```

In [132]:
```
#
############################################
# Step 14 - Create initial start vector.     #
# This can be controlled by providing value #
# of 'n'.                                   #
############################################
#
start_vector = np.repeat((1/n), n)
```

In [133]:
```
#
############################################
# Step 15 - Iterate given equation to calcu-#
# te influence matrix. Set max_iter to appr.#
# value.                                    #
############################################
#
max_iter = 50
alpha = 0.85
epsilon = 0.00001

start_time = time.time()
def eigen_function1(H,d,start_vector,max_iter):
```

```
        counter = 0
        for i in range(max_iter):
            prev_start_vector = start_vector
            counter = counter + 1
            start_vector = (alpha * np.dot(H,start_vector))+ np.dot(((alpha * np.dot(d,s
            start_vector_max = np.abs(start_vector).max()
            if start_vector_max == 0.0:
                start_vector_max = 1.0
            err = np.abs(start_vector - prev_start_vector).max() / start_vector_max
            if err < epsilon:
                print("Number of Iterations -  ", counter)
                return start_vector
        return start_vector
    links_matrix_result = eigen_function1(H,d,start_vector,max_iter)
    elapsed_time = time.time() - start_time
    links_matrix_result
    print("Time taken for completion -  ", elapsed_time)
    links_matrix_result
```

```
Number of Iterations -   33
Time taken for completion -   1.843142032623291
```

```
Out[133]: array([  8.49864549e-05,   2.05657561e-05,   4.63588916e-04, ...,
                   5.01720413e-05,   3.85319565e-05,   8.22743469e-05])
```

```
In [134]: #
          ###############################################
          # Step 16 - Calculate Eigenfactor (EFi) from#
          # given formula                             #
          ###############################################
          #
          B = np.dot(H,links_matrix_result)
          C = np.sum(B, axis=0)
          EF = 100*B/C
          EF[1:20]
```

```
Out[134]: array([ 0.0007564 ,  0.05300762,  0.00736773,  0.00668631,  0.00526172,
                  0.00396954,  0.00162661,  0.00362065,  0.00174103,  0.01511717,
                  0.03534588,  0.00764791,  0.00604364,  0.00133588,  0.00657294,
                  0.04367131,  0.002332  ,  0.00331866,  0.00180398])
```

```
In [135]: EF[::-1].sort()
          EF[1:20]
```

```
Out[135]: array([ 0.24738969,  0.24381262,  0.23516722,  0.226117  ,  0.22524979,
                  0.21670162,  0.20648099,  0.20143041,  0.18503212,  0.18273964,
                  0.18076642,  0.17508003,  0.17043871,  0.17019741,  0.16799231,
                  0.16356693,  0.15073475,  0.1493683 ,  0.14899786])
```