Page Rank:

import pandas as pd

import networkx as nx

import numpy as np

from collections import Counter, defaultdict

import matplotlib.pylab as pylab

import matplotlib.pyplot as plt

#setup the plotting

pylab.rcParams['figure.figsize'] = 16, 16

# read the main data source

emails = pd.read\_csv("../input/Emails.csv")

#cleanup the names in the From and To fields

with open("../input/Aliases.csv") as f:

file = f.read().split("\r\n")[1:] #skip the header line

aliases = {}

for line in file:

line = line.split(",")

aliases[line[1]] = line[2]

with open("../input/Persons.csv") as f:

file = f.read().split("\r\n")[1:] #skip header line

persons = {}

for line in file:

line = line.split(",")

persons[line[0]] = line[1]

def resolve\_person(name):

name = str(name).lower().replace(",","").split("@")[0]

#print(name)

#correct for some of the common people who are resolved to several different

# names by the given Aliases.csv file: Cheryl Mills, Huma Abedin, Jake Sullivan

# and Lauren Jiloty

# Also convert "h" and variations to Hillary Clinton

if ("mills" in name) or ("cheryl" in name) or ("nill" in name) or ("miliscd" in name) or ("cdm" in name) or ("aliil" in name) or ("miliscd" in name):

return "Cheryl Mills"

elif ("a bed" in name) or ("abed" in name) or ("hume abed" in name) or ("huma" in name) or ("eabed" in name):

return "Huma Abedin"

#elif (name == "abedin huma") or (name=="huma abedin") or (name=="abedinh"):

# return "Huma Abedin"

elif ("sullivan" in name) or ("sulliv" in name) or ("sulliy" in name) or ("su ii" in name) or ("suili" in name):

return "Jake Sullivan"

elif ("iloty" in name) or ("illoty" in name) or ("jilot" in name):

return "Lauren Jiloty"

elif "reines" in name: return "Phillip Reines"

elif (name == "h") or (name == "h2") or ("secretary" in name) or ("hillary" in name) or ("hrod" in name):

return "Hillary Clinton"

#fall back to the aliases file

elif str(name) == "nan": return "Redacted"

elif name in aliases.keys():

return persons[aliases[name]]

else: return name

emails.MetadataFrom = emails.MetadataFrom.apply(resolve\_person)

emails.MetadataTo = emails.MetadataTo.apply(resolve\_person)

#Extract the to: from: and Raw body text from each record

From\_To\_RawText = []

temp = zip(emails.MetadataFrom,emails.MetadataTo,emails.RawText)

for row in temp:

From\_To\_RawText.append(((row[0],row[1]),row[2]))

#Create a dictionary of all edges, i.e. (sender, recipient) relationships

# and store the individual email text in a list for each key

From\_To\_allText = defaultdict(list)

for people, text in From\_To\_RawText:

From\_To\_allText[people].append(text)

len(From\_To\_allText.keys()), len(From\_To\_RawText)

#Set the weights of each directed edge equal to the number of emails

# (number of raw text documents) associated with that edge

edges\_weights = [[key[0], key[1], len(val)] for key, val in From\_To\_allText.items()]

edge\_text = [val for key, val in From\_To\_allText.items()]

#initialize the graph

graph = nx.DiGraph()

#transform the dict with keys (from,to) and vals weight back to a

# tuple(from, to, weight)

graph.add\_weighted\_edges\_from(edges\_weights)

nx.set\_edge\_attributes(graph, 'text', edge\_text)

#Calculate the pagerank of each person (node) and store it with the node.

pagerank = nx.pagerank(graph)

pagerank\_list = {node: rank for node, rank in pagerank.items()}

nx.set\_node\_attributes(graph, 'pagerank', pagerank\_list)

#draw the graph

positions=nx.spring\_layout(graph)

#size of the graphed node proportional to its pagerank

nodesize = [x['pagerank']\*30000 for v,x in graph.nodes(data=True)]

edgesize = [np.sqrt(e[2]['weight']) for e in graph.edges(data=True)]

nx.draw\_networkx\_nodes(graph, positions, node\_size=nodesize, alpha=0.4)

nx.draw\_networkx\_edges(graph, positions, edge\_size=edgesize, alpha=0.2)

nx.draw\_networkx\_labels(graph, positions, font\_size=10)

plt.savefig("email\_graph.png")

plt.title("Graph of all send/receive relationships in the Clinton email database", fontsize=20)

plt.clf()

#That graph is pretty big. Let's make a smaller one with just the most

# important people.

#This will plot only the nodes with pagerank greater than

# pagerank\_cutoff

pagerank\_cutoff = 0.0045

small\_graph = graph.copy()

for n, p\_rank in small\_graph.nodes(data=True):

if p\_rank['pagerank'] < pagerank\_cutoff: small\_graph.remove\_node(n)

spositions=nx.spring\_layout(small\_graph, weight=None)

snodesize = [x['pagerank']\*30000 for v,x in small\_graph.nodes(data=True)]

sedgesize = [np.log(e[2]['weight']) for e in small\_graph.edges(data=True)]

scolors = np.random.rand(len(small\_graph.nodes()))

nx.draw\_networkx\_nodes(small\_graph, spositions, node\_size=snodesize, node\_color=scolors, alpha=0.3)

nx.draw\_networkx\_edges(small\_graph, spositions, alpha=0.3, arrows=False) #, width=sedgesize)

nx.draw\_networkx\_labels(small\_graph, spositions, font\_size=14)

plt.title("Graph of only those people with a pagerank of greater than %s" % pagerank\_cutoff, fontsize=20)

plt.savefig("small\_graph.png")



