8i70fawma

October 9, 2024

[1]:

```
def pagerank(G, alpha=0.85, personalization=None, max_iter=100, tol=1.0e-6,␣
 ↪nstart=None, weight='weight', dangling=None):
    if len(G) == 0:
        return {}

    if not G.is_directed():
        D = G.to_directed()
    else:
        D = G

    # Create a copy in (right) stochastic form
    W = nx.stochastic_graph(D, weight=weight)
    N = W.number_of_nodes()

    # Choose fixed starting vector if not given
    if nstart is None:
        x = dict.fromkeys(W, 1.0 / N)
    else:
        # Normalized nstart vector
        s = float(sum(nstart.values()))
        x = dict((k, v / s) for k, v in nstart.items())

    if personalization is None:
        # Assign uniform personalization vector if not given
        p = dict.fromkeys(W, 1.0 / N)
    else:
        missing = set(G) - set(personalization)
        if missing:
            raise NetworkXError('Personalization dictionary must have a value␣
 ↪for every node. Missing nodes %s' % missing)
        s = float(sum(personalization.values()))
        p = dict((k, v / s) for k, v in personalization.items())

    if dangling is None:
        # Use personalization vector if dangling vector not specified
        dangling_weights = p
    else:
        missing = set(G) - set(dangling)
        if missing:
            raise NetworkXError('Dangling node dictionary must have a value for␣
 ↪every node. Missing nodes %s' % missing)
        s = float(sum(dangling.values()))
        dangling_weights = dict((k, v/s) for k, v in dangling.items())

    dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

     # power iteration: make up to max_iter iterations
    for _ in range(max_iter):
        xlast = x
        x = dict.fromkeys(xlast.keys(), 0)
        danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
        for n in x:
            # this matrix multiply looks odd because it is
            # doing a left multiply x^T=xlast^T*W
            for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
            x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]

        # check convergence, l1 norm
        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N*tol:
            return x
    raise NetworkXError('Pagerank: power iteration failed to converge in %d␣
 ↪iterations.' % max_iter)
```

[2]:

```
import networkx as nx
```

[3]:

```
G = nx.barabasi_albert_graph(60, 41)
pr = nx.pagerank(G, 0.4)
```

[4]:

```
print(pr)
```

```
{0: 0.028121839712461336, 1: 0.013572060204606995, 2: 0.012377192811464105, 3:
0.01276660315849177, 4: 0.012959694372101743, 5: 0.013566505760540899, 6:
0.01315965835549713, 7: 0.012761045622178556, 8: 0.012569962112694677, 9:
0.011966256416940244, 10: 0.01216437797589254, 11: 0.01274439064043514, 12:
0.012355193875562914, 13: 0.013574326955512387, 14: 0.013565586674507725, 15:
0.013161160879256366, 16: 0.012770464290540922, 17: 0.013162913725511647, 18:
0.012549442059234073, 19: 0.013159408219004354, 20: 0.012753536923912618, 21:
0.012962623788426724, 22: 0.01296134609496946, 23: 0.012567245261440485, 24:
0.013151712860473175, 25: 0.012176422979460987, 26: 0.01316880057844345, 27:
0.012966132417362861, 28: 0.012375689396967038, 29: 0.012956100666025625, 30:
0.012556265952474657, 31: 0.013563202995477657, 32: 0.012950837162972404, 33:
0.013563077536531685, 34: 0.012751231097713185, 35: 0.013364585518060983, 36:
```

```
0.012757344165725625, 37: 0.013573406710585708, 38: 0.013362677844185263, 39:
0.012772082743841595, 40: 0.012967147343562274, 41: 0.012959933326479359, 42:
0.027709566315767126, 43: 0.027265874557541092, 44: 0.027057589243851085, 45:
0.026487545775565946, 46: 0.025987193614687674, 47: 0.025668431600195996, 48:
0.025604679445328363, 49: 0.02522963758290587, 50: 0.024569183030765128, 51:
0.024405237934256092, 52: 0.02411942880215776, 53: 0.023561037315504574, 54:
0.023159391433007742, 55: 0.023141335115809156, 56: 0.022526010703206087, 57:
0.02210779500473849, 58: 0.021848949694802174, 59: 0.021341625642380603}
```

[ ]: