Date
24/01/22

Neeraj Appan 7073 7073

SHETH L.U.J. COLLEGE OF ARTS &
SIR M.V. COLLEGE OF SCIENCE & COMMERCE
Department of Computer Science

Information Retrieval Practical 6

Aim: Implement Page Rank Algorithm (Python / Java)

Write-ups:

Page Rank (PR) is an algorithm used by Google Search to rank websites in their search engine results. Page Rank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.

$$PR(A) = (1-d) + d(PR(T_i)/C(T_i) + \ldots + PR(T_n)/C(T_n))$$

Information Retreival Pratical-6.py - E:/fffiles/college pracs and projects/IR/Information Retreival Pratical-6.py (3.9.6)

File Edit Format Run Options Window Help

```
def pagerank(G, alpha=0.85, personalization=None,
             max_iter=100, tol=1.0e-6, nstart=None, weight='weight',
             dangling=None):
    """Return the PageRank of the nodes in the graph.

    PageRank computes a ranking of the nodes in the graph G based on
    the structure of the incoming links. It was originally designed as
    an algorithm to rank web pages.

    Parameters
    ----------
    G : graph
      A NetworkX graph.  Undirected graphs will be converted to a directed
      graph with two directed edges for each undirected edge.

    alpha : float, optional
      Damping parameter for PageRank, default=0.85.

    personalization: dict, optional
      The "personalization vector" consisting of a dictionary with a
      key for every graph node and nonzero personalization value for each node.
      By default, a uniform distribution is used.

    max_iter : integer, optional
      Maximum number of iterations in power method eigenvalue solver.

    tol : float, optional
      Error tolerance used to check convergence in power method solver.

    nstart : dictionary, optional
      Starting value of PageRank iteration for each node.
```

Ln: 130 Col: 0

Information Retreival Pratical-6.py - E:/fffiles/college pracs and projects/IR/Information Retreival Pratical-6.py (3.9.6)

File Edit Format Run Options Window Help

```
    """
    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()))
```

Ln: 130 Col: 0

Information Retreival Pratical-6.py - E:/fffiles/college pracs and projects/IR/Information Retreival Pratical-6.py (3.9.6)

File Edit Format Run Options Window Help

```
        # 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)
```

Ln: 130 Col: 0

IDLE Shell 3.9.6

File Edit Shell Debug Options Window Help

```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: E:/fffiiles/college pracs and projects/IR/Information Retreival Pratical-6.py
>>> import networkx as nx
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import networkx as nx
ModuleNotFoundError: No module named 'networkx'
>>> import networkx as nx
>>> G=nx.barabasi_albert_graph(60,41)
>>> pr=nx.pagerank(G,0.4)
>>> pr
{0: 0.028610501946263196, 1: 0.012574206003328516, 2: 0.013370792636413435, 3: 0.012963666308028966, 4: 0.012359946240072382, 5: 0.012559376632008425, 6: 0.012160840687233505, 7: 0.012351489727922084, 8: 0.012562114554829269, 9: 0.013168406815135538, 10: 0.013167273569798521, 11: 0.013168356756333561, 12: 0.013365425394756042, 13: 0.013364058271914385, 14: 0.013376110987936727, 15: 0.012787612532334 85, 16: 0.012765290681910214, 17: 0.012355389581602, 18: 0.012968432724922813, 19: 0.013169384506337119, 20: 0.012358480664166867, 21: 0.013150217028142088, 22: 0.013774664484054233, 23: 0.013774664484054233, 24: 0.013375318313965308, 25: 0.013358833781303613, 26: 0.013357178585773833, 27: 0.013384147640023828, 28: 0.013563784104625578, 29: 0.013367202751561183, 30: 0.012754978445360433, 31: 0.013172973882856386, 32: 0.012558771600628585, 33: 0.013168762853915538, 34: 0.012349493695928056, 35: 0.010807672320235006, 36: 0.012368926743061892, 37: 0.013166094705048818, 38: 0.012971039125468842, 39: 0.012760876957117853, 40: 0.012966995213541234, 41: 0.013565795338935 36, 42: 0.02753382649151867, 43: 0.027217801613498238, 44: 0.02667900981666993, 45: 0.026291128888866561, 46: 0.02608724874352708, 47: 0.026068871902038493, 48: 0.025349470558176754, 49: 0.025789542033407123, 50: 0.024632780188187 7, 51: 0.024326830296622224, 52: 0.023761255398667643, 53: 0.023467105828842753, 54: 0.022942686284306388, 55: 0.022578554583372937, 56: 0.0231962108568846, 57: 0.021693047216502538, 58: 0.021730489375088518, 59: 0.021408590645172476}
>>>
```

Ln: 14 Col: 339