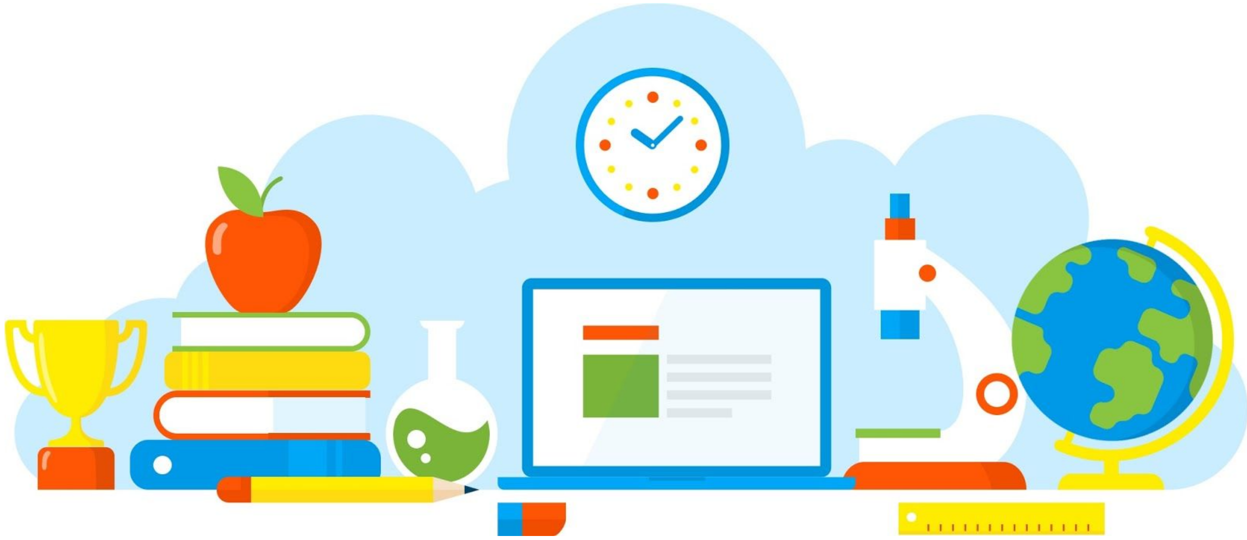

Mining Massive Datasets HW4

Teacher: Dr. Haratizadeh

Mahdi Sadeghi
830598035

Loading Dataset

For loading dataset i've written a function that reads the data file line by line and generates a degree list (which indicates outgoing degree of each node) and a link matrix. There is a utility variable called id_converter that holds the mapping between nodes real names in the file and their index in the generated matrices. After generating link matrix and degree list, we can generate the transition matrix by dividing the link matrix to the degrees. The matrices are of scipy csr sparse matrices so that they don't take as much memory as the dense version.

```
def load_dataset():
   """
   this functions loads dataset and generates link matrix, transition
matrix and a id converter to map matrix indices
   to node names in the file and vice versa
   :return:
   """
   l_matrix_rows = []
   l_matrix_columns = []

   degree_list = [0] * NODES_COUNT

   id_converter = {}
   last_mat_pos = 0

   with open(path.join(path.abspath(path.dirname(__file__)),
'../data/wiki.data'), 'r') as fp:
       line = fp.readline()

       while line:
           node1_id, node2_id = line.split('\t')
           node1_id = int(node1_id)
           node2_id = int(node2_id)

           if node1_id not in id_converter:
               id_converter[node1_id] = last_mat_pos
               id_converter['r' + str(last_mat_pos)] = node1_id
```

```
                last_mat_pos += 1

            if node2_id not in id_converter:
                id_converter[node2_id] = last_mat_pos
                id_converter['r' + str(last_mat_pos)] = node2_id
                last_mat_pos += 1

            node1 = id_converter[node1_id]
            node2 = id_converter[node2_id]

            l_matrix_rows.append(node1)
            l_matrix_columns.append(node2)

            degree_list[node1] += 1

            line = fp.readline()

    matrix_data = [1.0] * len(l_matrix_rows)
    l_matrix = sparse.csr_matrix((matrix_data, (l_matrix_rows,
l_matrix_columns)), shape=(NODES_COUNT, NODES_COUNT))

    for i, node in enumerate(l_matrix_rows):
        matrix_data[i] = 1 / degree_list[node]

    m_matrix = sparse.csr_matrix((matrix_data, (l_matrix_rows,
l_matrix_columns)), shape=(NODES_COUNT, NODES_COUNT))

    return l_matrix, m_matrix.T, id_converter
```

Pagerank

After loading the transition dataset, we pass it to the pagerank function which calculates page rank of all nodes using the following formula:

$$v' = \beta M v + (1 - \beta)e/n$$

We are using teleport matrix and taxation to avoid dead end nodes and spider traps. We repeat applying this formula for 100 times to converge. After that nodes with the highest page ranks are returned to the user. Results are as follows:

Top nodes according to Pagerank:

Node Id	Page Rank
4037	0.0019237982657767765
15	0.0015365855168042307
6634	0.0014977469730989852
2625	0.0013711426241683426
2398	0.0010892770092432117
2470	0.001053840867992798
2237	0.0010425060275712954
4191	0.0009469774364547144
7553	0.0009060053264184172
5254	0.0008978085399789357

```
def pagerank(m_matrix):
    """
    computes pagerank algorithm on input transition matrix
    :param m_matrix: transition matrix
    :return:
    """
    n = m_matrix.shape[0]
```

```
    ranks = np.ones((n, 1)) / n  # initial ranks is same as e/n

    bm_matrix = PAGERANK_BETA * m_matrix  # b*M
    bteleport_matrix = (1 - PAGERANK_BETA) * ranks  # (1-b) * (e/n)
    for i in range(PAGERANK_ITERATIONS):
        ranks = (bm_matrix * ranks) + bteleport_matrix  # v' = bMv +
(1-b)e/n

    return ranks  # a n * 1 vector showing pagerank of each node (ith
element shows ith node pagerank)
```

Hubs and Authorities

For computing node importance from HITS algorithm, we use the link matrix we have already obtained from load_dataset function. We use the following algorithm to compute hubbiness and authority of each node (take h as hubbiness vector and a as authority vector)[1].

1. Compute $a = L^T h$ and scale it so that the largest component is 1.
2. Compute $h = La$ and scale it so that the largest component is 1.
3. Repeat until convergence

In this homework i used 100 iterations for the algorithm. The results are as follows:

Top Nodes according to Hubbiness:

Node Id	Hubbiness Score
2565	1.0
766	0.9538873185707487
2688	0.8110641527855376
457	0.8081199399226742
1166	0.7569515045643498
1549	0.7204532852731531
11	0.619757771298162
1151	0.5757880360582264
1374	0.5626714810945165
1133	0.49353130543630436

[1] In first step h is all 1

Top Nodes according to Authority:

Node Id	Authority Score
2398	1
4037	0.9973233876586834
3352	0.9024349895010808
1549	0.8928682441421767
762	0.8743202230920875
3089	0.8733636234620302
1297	0.8720993344260224
2565	0.8617973900502678
15	0.8532627562217322
2625	0.8518493914363475

```
def hits(l_matrix):
    """
    computes hub and authority ranks using hits algorithm and link
matrix
    :param l_matrix: link matrix
    :return:
    """
    n = l_matrix.shape[0]

    h = np.ones((n, 1))
    a = np.ones((n, 1))

    l_t_matrix = l_matrix.T

    for i in range(HITS_ITERATIONS):
        a = l_t_matrix * h
        f = a.max()
        if f != 0:
            a = a / f
```

```
        h = l_matrix * a
        f = h.max()
        if f != 0:
            h = h / f

    return h, a
```