CS471: Graph Machine Learning and Mining

Lab #1: PPR (Solution)

JOYCE JIYOUNG WHANG, School of Computing, KAIST

Load Dataset

- We provide the NELL995 dataset and the skeleton code
- Load "nodes.txt" and "edges.txt"
  1. Run this cell

```python
from google.colab import files
f = files.upload()
```

  2. Click the "Choose Files" button
  3. Find and choose both files
  4. The output is as follows

```
파일 선택 파일 2개
• edges.txt(text/plain) - 1514330 bytes, last modified: 2023. 3. 16. - 100% done
• nodes.txt(text/plain) - 155406 bytes, last modified: 2023. 3. 16. - 100% done
Saving edges.txt to edges.txt
Saving nodes.txt to nodes.txt
```

Load Dataset

- Preprocess the uploaded files
- edgeList: a list of the edges that are in the form of (node, node)
  - [(person:molly_moore, city:washington_d_c), …]
- nodeList: a list of the node names
  - [country:scandinavia, university:emory, …]
- node2id: assign each node to a unique value

```
edges = f['edges.txt'].decode('utf-8').strip().split('\n')
edgeList = [(edge.split()[0], edge.split()[1]) for edge in edges]

nodeList = f['nodes.txt'].decode('utf-8').strip().split('\n')
node2id = dict(zip(nodeList, range(len(nodeList))))
```

Computing PPR – Approach 1

- For the implementation of computing PPR, we need to define the node and graph classes

```python
class Node:
    def __init__(self, id):
        self.id = id
        self.inNode = []
        self.outNode = []
        self.outDegree = 0

        self.pagerank = 0
        self.pagerankNext = 0
        self.personalized = 0
```

```python
class Graph:
    def __init__(self, nodeList, edgeList):
        self.nodeList = nodeList
        self.edgeList = edgeList
        self.numNodes = len(self.nodeList)
        self._build_graph()

    def _build_graph(self):
        self.nodes = {}
        for id in self.nodeList:
            node = Node(id)
            self.nodes[id] = node

        for edge in self.edgeList:
            headID = edge[0]
            tailID = edge[1]

            self.nodes[headID].outNode.append(self.nodes[tailID])
            self.nodes[tailID].inNode.append(self.nodes[headID])

            self.nodes[headID].outDegree += 1
```

Computing PPR – Approach 1

- Input parameters for computing PPR
  - alpha : the probability to follow out-links
  - maxIters : the predefined maximum number of iterations
  - tolerance : a small value to check convergence
  - personalize : the predefined set $\mathcal{Q}$ (entire nodes in the case of Global PageRank)
- A single iteration of computing PPR

$$x_v^{(k+1)} = \alpha \Sigma_{w \in \mathcal{S}_v} \frac{x_w^{(k)}}{|\mathcal{T}_w|} + \frac{1-\alpha}{n_q}, \qquad v \in \mathcal{Q}$$

$$x_v^{(k+1)} = \alpha \Sigma_{w \in \mathcal{S}_v} \frac{x_w^{(k)}}{|\mathcal{T}_w|}, \qquad v \notin \mathcal{Q}$$

Computing PPR – Approach 1

- TODO: Complete the 'compute_PageRank' function
  - Generate a graph object
  - Initialize the PageRank score of each node

```python
def compute_PageRank(nodeList, edgeList, maxIters, alpha, tolerance, personalize = None):
    graph = Graph(nodeList, edgeList)

    if personalize is None:
        personalize = graph.nodeList
    norm = len(personalize)

    for node in graph.nodes.values():
        if node.id in personalize:
            node.pagerank = (1-alpha) / norm
            node.personalized = (1-alpha) / norm
```

Computing PPR – Approach 1

- TODO: Compute the PPR score of a single node
  - Update the 'Node' class for computing the PPR score

```python
class Node:
    def __init__(self, id):
        self.id = id
        self.inNode = []
        self.outNode = []
        self.outDegree = 0

        self.pagerank = 0
        self.pagerankNext = 0
        self.personalized = 0

    def aggregate_pagerank(self, alpha):
        self.pagerankNext = alpha * sum((node.pagerank/node.outDegree) for node in self.inNode) + self.personalized

    def update_pagerank(self):
        self.pagerank = self.pagerankNext
```

Computing PPR – Approach 1

- TODO: Compute the PageRank score of the next iteration iteratively
  - Calculate the L_∞ norm to check convergence

```python
for iter in range(maxIters):

    prevPageRanks = [node.pagerank for node in graph.nodes.values()]

    for node in graph.nodes.values():
        node.aggregate_pagerank(alpha)

    for node in graph.nodes.values():
        node.update_pagerank()

    currPageRanks = [node.pagerank for node in graph.nodes.values()]

    error = max(abs(prevPageRank-currPageRank) for prevPageRank, currPageRank in zip(prevPageRanks, currPageRanks))

    if error < tolerance:
        print("Total iterations : ", iter)
        break
else:
    print("It reaches the maximum iterations. Please increase the maxIters.")
```

Computing PPR – Approach 1

- Return it in the form of a dictionary
  - Key : a node name
  - Value : the PageRank score of the corresponding node
  - Ex) {'city:baker': 0.01, 'city:kenner': 0.0001, … }

```python
pageranks = [node.pagerank for node in graph.nodes.values()]
pageranks = dict(zip(graph.nodeList, pageranks))

return pageranks
```

Printing PageRank Values

- Print the top 10 values of PageRank
- Print PageRank scores with the 'prettytable' library
  - More information: https://pypi.org/project/prettytable/
- You can use any other library for well-visualizing
  - Just using the built-in 'print' function is okay

```python
from prettytable import PrettyTable

def print_PageRank_top10(pageranks, name='PageRank'):
    pagerankSorted = sorted(pageranks.items(), reverse=True, key=lambda x: x[1])[:10]

    table = PrettyTable(field_names = ['Node ID', name])
    for id, score in pagerankSorted[:10]:
        table.add_row([id, round(score, 4)])
    print(table)
```

Total iterations : 84

Node ID	PageRank
stateorprovince:california	0.032
city:florida	0.0154
plant:trees	0.0146
personmexico:ryan_whitney	0.0131
stateorprovince:texas	0.0116
country:usa	0.0098
sportsteam:ncaa_youth_kids	0.0077
vegetable:pepper	0.0074
profession:professionals	0.0073
country:countries	0.0072

Example

Computing PPR – Approach 1

- Compute the Global PageRank score
  - Max iterations : 10000
  - Alpha : 0.85
  - Tolerance : 1e-8
- The output is as follows

```python
pageranks = compute_PageRank(nodeList, edgeList, 10000, 0.85, 1e-8)

print_PageRank_top10(pageranks)
```

```
Total iterations :  84
+----------------------------+----------+
|          Node ID           | PageRank |
+----------------------------+----------+
| stateorprovince:california |  0.032   |
|        city:florida        |  0.0154  |
|        plant:trees         |  0.0146  |
| personmexico:ryan_whitney  |  0.0131  |
|   stateorprovince:texas    |  0.0116  |
|        country:usa         |  0.0098  |
| sportsteam:ncaa_youth_kids |  0.0077  |
|      vegetable:pepper      |  0.0074  |
|  profession:professionals  |  0.0073  |
|     country:countries      |  0.0072  |
+----------------------------+----------+
```

Computing PPR – Approach 1

- Compute the Personalized PageRank with the same parameters
  - Predefined set :
    ['politicianus:joe_biden', 'politicianus:biden', 'politicianus:senator_biden']

```python
personalizeList = ['politicianus:joe_biden', 'politicianus:biden', 'politicianus:senator_biden']

personalizedPageranks = compute_PageRank(nodeList, edgeList, 10000, 0.85, 1e-8, personalize = personalizeList)

print_PageRank_top10(personalizedPageranks, name='PPR')
```

- The output is as follows

Total iterations : 85

Node ID	PPR
politicianus:biden	0.0569
politician:clinton	0.0529
politicianus:joe_biden	0.0519
politicianus:senator_biden	0.0506
politicianus:palin	0.0301
stateorprovince:california	0.0274
politicaloffice:office	0.0172
politician:obama	0.0161
politicianus:mccain	0.016
politicianus:barack_obama	0.0141

Computing PPR – Approach 2

- We can implement the Power method with matrix-vector multiplication
- We use NumPy and SciPy libraries
  - NumPy: https://numpy.org/doc/stable/index.html
  - SciPy: https://docs.scipy.org/doc/scipy/index.html

Computing PPR – Approach 2

- TODO: Complete the 'compute_PageRank_with_sparse_matrix' function
  - Generate an adjacency matrix from edge list with 'scipy.sparse.coo_matrix'
  - Since the given dataset is large and sparse, you should use a sparse matrix format

```
import numpy as np
from scipy.sparse import coo_matrix

def compute_PageRank_with_sparse_matrix(nodeList, edgeList, node2id, maxIters, alpha, tolerance, personalize = None):
    numNodes = len(nodeList)

    edgeList = [(node2id[edge[0]], node2id[edge[1]]) for edge in edgeList]

    rows, cols, data = zip(*[(edge[0], edge[1], 1) for edge in edgeList])
    adjSparse = coo_matrix((data, (rows, cols)), shape = (numNodes, numNodes), dtype=float)

    D = np.array(adjSparse.sum(axis=1), dtype=float)
    D[D!=0] = 1.0 / D[D!=0]
    P = adjSparse.multiply(D)
    PT = P.transpose()
```

Computing PPR – Approach 2

- TODO: Initialize Personalized PageRank vector

$$\boldsymbol{x} = \frac{(1-\alpha)}{n_q}\boldsymbol{e}_q$$

Global PageRank vector: $n_q \equiv n, \boldsymbol{e}_q \equiv \boldsymbol{e}$

```python
if personalize is None:
    personalizeVector = np.ones((numNodes, 1), dtype=float)
else:
    personalizeVector = np.array([1 if node in personalize else 0 for node in nodeList], dtype=float).reshape((numNodes, 1))
personalizeVector /= np.sum(personalizeVector)
personalizeVector *= (1-alpha)
pageranks = np.copy(personalizeVector)
```

Computing PPR – Approach 2

- TODO: Compute the PageRank score of the next iteration iteratively
  - Implement the power method
  - You should use the sparse matrix-vector multiplication
  - Also, calculate the L_∞ norm to check convergence

$$\boldsymbol{x} = \alpha \boldsymbol{P}^T \boldsymbol{x} + \frac{(1-\alpha)}{n_q} \boldsymbol{e}_q$$

```python
for iter in range(maxIters):
    prevPageranks = pageranks

    pageranks = alpha * PT @ pageranks + personalizeVector

    error = max(np.absolute(prevPageranks - pageranks))

    if error < tolerance:
        print("Total iterations : ", iter)
        break
else:
    print("It reaches the maximum iterations. Please increase the maxIters.")
```

Computing PPR – Approach 2

- Return it in the form of dictionary

```python
pageranks = np.asarray(pageranks)
pageranks = dict(zip(nodeList, pageranks.squeeze().tolist()))
return pageranks
```

Comparing the Results

- Compare the results of the Approach 1 & Approach 2
- Compare the results of the Global & Personalized PageRank

```
pageranks = compute_PageRank(nodeList, edgeList, 10000, 0.85, 1e-8)
print_PageRank_top10(pageranks)

pageranks = compute_PageRank_with_sparse_matrix(nodeList, edgeList, node2id, 10000, 0.85, 1e-8)
print_PageRank_top10(pageranks)


personalizeList = ['politicianus:joe_biden', 'politicianus:biden', 'politicianus:senator_biden']

personalizedPageranks = compute_PageRank(nodeList, edgeList, 10000, 0.85, 1e-8, personalize = personalizeList)
print_PageRank_top10(personalizedPageranks, name='PPR')

personalizedPageranks = compute_PageRank_with_sparse_matrix(nodeList, edgeList, node2id, 10000, 0.85, 1e-8, personalize = personalizeList)
print_PageRank_top10(personalizedPageranks, name='PPR')
```

Comparing the Results

- The results are as follows

Total iterations : 84

Node ID	PageRank
stateorprovince:california	0.032
city:florida	0.0154
plant:trees	0.0146
personmexico:ryan_whitney	0.0131
stateorprovince:texas	0.0116
country:usa	0.0098
sportsteam:ncaa_youth_kids	0.0077
vegetable:pepper	0.0074
profession:professionals	0.0073
country:countries	0.0072

PageRank with Approach 1

Total iterations : 84

Node ID	PageRank
stateorprovince:california	0.032
city:florida	0.0154
plant:trees	0.0146
personmexico:ryan_whitney	0.0131
stateorprovince:texas	0.0116
country:usa	0.0098
sportsteam:ncaa_youth_kids	0.0077
vegetable:pepper	0.0074
profession:professionals	0.0073
country:countries	0.0072

PageRank with Approach 2

Total iterations : 85

Node ID	PPR
politicianus:biden	0.0569
politician:clinton	0.0529
politicianus:joe_biden	0.0519
politicianus:senator_biden	0.0506
politicianus:palin	0.0301
stateorprovince:california	0.0274
politicaloffice:office	0.0172
politician:obama	0.0161
politicianus:mccain	0.016
politicianus:barack_obama	0.0141

Personalized PageRank with Approach 1

Total iterations : 85

Node ID	PPR
politicianus:biden	0.0569
politician:clinton	0.0529
politicianus:joe_biden	0.0519
politicianus:senator_biden	0.0506
politicianus:palin	0.0301
stateorprovince:california	0.0274
politicaloffice:office	0.0172
politician:obama	0.0161
politicianus:mccain	0.016
politicianus:barack_obama	0.0141

Personalized PageRank Approach 2

Submission Guide

- After completion of implementation, you should run all the cells
- Submit your ipython notebook in 'ipynb' format
  - Do not remove your output results from every cell
- File name format: lab1_studentID_name.ipynb
  - Ex) lab1_20233809_MinsungHwang.ipynb
- Submission due: March 27th by 10:00 AM
  - We do not accept late submissions