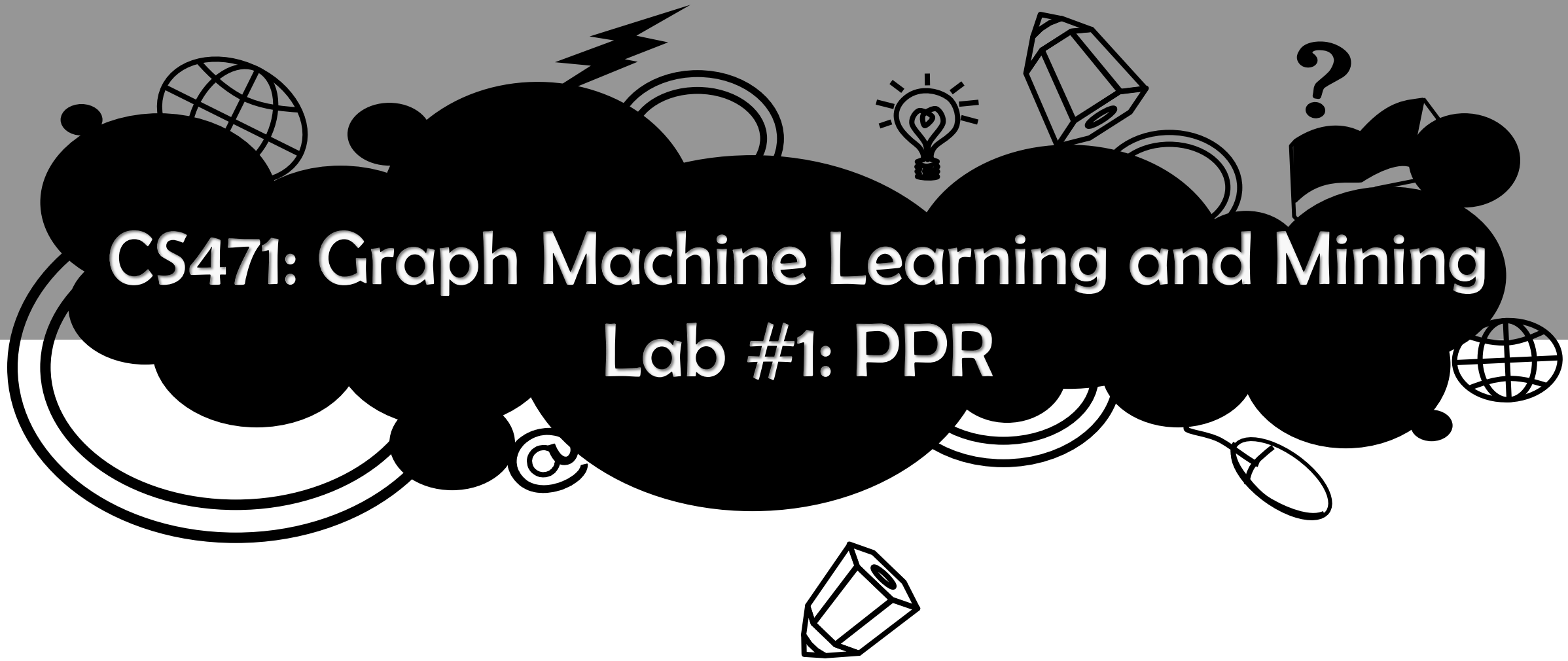

CS471: Graph Machine Learning and Mining
Lab #1: PPR

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

```
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) - 1430989 bytes, last modified: 2024. 3. 21. - 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
  - edge_list: a list of the edges that are in the form of (node, node)
    - [(person:molly_moore, city:washington_d_c), …]
  - node_list: a list of the node names
    - [country:scandinavia, university:emory, …]
  - node_to_id: a mapping for each node to a unique integer value

```
node_list = f['nodes.txt'].decode('utf-8').strip().split('₩n')
node_to_id = dict(zip(node_list, range(len(node_list))))

edge_lines = f['edges.txt'].decode('utf-8').strip().split('₩n')
edge_list = [(edge.split()[0], edge.split()[1]) for edge in edge_lines]

print(f"# Nodes: {len(node_list)}")
print(f"# Edges: {len(edge_list)}")
```

```
# Nodes: 7363
# Edges: 35146
```

Computing PPR – Approach 1

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

```python
from typing import List

class Node:

    def __init__(self, id: str) -> None:

        self.id: str = id
        self.in_neighbors: List[Node] = []
        self.out_neighbors: List[Node] = []

        self.pagerank: float = 0
        self.pagerank_next: float = 0
        self.personalized: float = 0

    @property
    def out_degree(self) -> int:
        return len(self.out_neighbors)
```

```python
from typing import Tuple, Dict

class Graph:

    def __init__(self, node_list: List[str], edge_list: List[Tuple[str, str]]):

        self.nodes: Dict[str, Node] = {}
        self.edges = edge_list

        self._build_graph(node_list)
        self.num_nodes = len(self.nodes)

    def _build_graph(self, node_list: List[str]):

        for id in node_list:
            node = Node(id)
            self.nodes[id] = node

        for edge in self.edges:
            id_head = edge[0]
            id_tail = edge[1]

            self.nodes[id_head].out_neighbors.append(self.nodes[id_tail])
            self.nodes[id_tail].in_neighbors.append(self.nodes[id_head])
```

Computing PPR – Approach 1

- Input parameters for computing PPR
  - max_iters : the predefined maximum number of iterations
  - alpha : the probability to follow out-links
  - tolerance : a small value to check convergence
  - personalize_list : the predefined set Q (entire nodes in the case of Global PageRank)

- A single iteration of computing PPR

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

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

Computing PPR – Approach 1

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

```
from typing import Optional

def compute_pagerank(node_list: List[str], edge_list: List[Tuple[str, str]],
                     max_iters: int, alpha: float, tolerance: float,
                     personalize_list: Optional[List[str]] = None) -> Dict[str, float]:

    graph = Graph(node_list, edge_list)
```

TODO

Computing PPR – Approach 1

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

```
# TODO: Compute Personalized PageRank of a single node
def aggregate_pagerank(self, alpha: float) -> None:
```

TODO

Computing PPR – Approach 1

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

```
for iter in range(max_iters):

    # TODO: Implement one iteration
```

TODO

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, …}

```
pageranks = [node.pagerank for node in graph.nodes.values()]
pageranks = dict(zip(graph.nodes, 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: Dict[str, float], name='PageRank') -> None:

    pageranks_sorted = sorted(pageranks.items(), reverse=True, key=lambda x: x[1])[:10]

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

```
Total iterations :  84
+---------------------------+----------+
|          Node ID          | PageRank |
+---------------------------+----------+
| stateorprovince:california |  0.0313  |
|        plant:trees        |  0.0148  |
|        city:florida       |  0.0147  |
| personmexico:ryan_whitney |  0.0131  |
|   stateorprovince:texas   |  0.0116  |
|        country:usa        |  0.0091  |
|      vegetable:pepper     |  0.0075  |
|  profession:professionals |  0.0073  |
| sportsteam:ncaa_youth_kids |  0.0072  |
|     country:countries     |  0.007   |
+---------------------------+----------+
```

Example

Computing PPR – Approach 1

- Compute the Global PageRank score
  - Max iterations : 10000
  - Alpha : 0.85
  - Tolerance : 1e-8

```
pageranks = compute_pagerank(node_list, edge_list, 10000, 0.85, 1e-8)
print_pagerank_top10(pageranks)
```

- The output is as follows

```
Total iterations :  84
+---------------------------+----------+
|          Node ID          | PageRank |
+---------------------------+----------+
| stateorprovince:california |  0.0313  |
|        plant:trees        |  0.0148  |
|        city:florida       |  0.0147  |
| personmexico:ryan_whitney |  0.0131  |
|   stateorprovince:texas   |  0.0116  |
|        country:usa        |  0.0091  |
|      vegetable:pepper     |  0.0075  |
|  profession:professionals |  0.0073  |
| sportsteam:ncaa_youth_kids |  0.0072  |
|     country:countries     |  0.007   |
+---------------------------+----------+
```

Computing PPR – Approach 1

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

```
# Set the predefined set in the form of a list
personalize_list = ['politicianus:joe_biden', 'politicianus:biden', 'politicianus:senator_biden']
personalized_pageranks = compute_pagerank(node_list, edge_list, 10000, 0.85, 1e-8,
                                          personalize_list=personalize_list)
print_pagerank_top10(personalized_pageranks, name='PPR')
```

- The output is as follows

```
Total iterations :  85
+---------------------------+--------+
|          Node ID          |  PPR   |
+---------------------------+--------+
|    politicianus:biden     | 0.057  |
|    politician:clinton     | 0.0531 |
|  politicianus:joe_biden   | 0.0519 |
| politicianus:senator_biden | 0.0506 |
|    politicianus:palin     | 0.0302 |
| stateorprovince:california | 0.0269 |
|  politicaloffice:office   | 0.0173 |
|     politician:obama      | 0.0162 |
|   politicianus:mccain     | 0.0161 |
| 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
  - Compute $\boldsymbol{P}^{T}$ where $\boldsymbol{P} \equiv \boldsymbol{D}^{-1}\boldsymbol{A}$

```python
import numpy as np
from scipy.sparse import diags, coo_matrix

def compute_pagerank_with_sparse_matrix(node_list: List[str], edge_list: List[Tuple[str, str]], node_to_id: Dict[str, int],
                                        max_iters: int, alpha: float, tolerance: float,
                                        personalize_list: Optional[List[str]] = None) -> Dict[str, float]:

    # TODO: Compute the Personalized PageRank with the predefined list using sparse matrix-vector multiplication
```

TODO

Computing PPR – Approach 2

- TODO: Initialize Personalized PageRank vector

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

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

TODO

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

```
for iter in range(max_iters):

    # TODO: Implement power iteration
```

TODO

Comparing the Results

- Compare the results of the Approach 1 & Approach 2
- Compare the results of the Global & Personalized PageRank
- The results are as follows

Total iterations : 84

Node ID	PageRank
stateorprovince:california	0.0313
plant:trees	0.0148
city:florida	0.0147
personmexico:ryan_whitney	0.0131
stateorprovince:texas	0.0116
country:usa	0.0091
vegetable:pepper	0.0075
profession:professionals	0.0073
sportsteam:ncaa_youth_kids	0.0072
country:countries	0.007

PageRank with Approach 1

Total iterations : 84

Node ID	PageRank
stateorprovince:california	0.0313
plant:trees	0.0148
city:florida	0.0147
personmexico:ryan_whitney	0.0131
stateorprovince:texas	0.0116
country:usa	0.0091
vegetable:pepper	0.0075
profession:professionals	0.0073
sportsteam:ncaa_youth_kids	0.0072
country:countries	0.007

PageRank with Approach 2

Total iterations : 85

Node ID	PPR
politicianus:biden	0.057
politician:clinton	0.0531
politicianus:joe_biden	0.0519
politicianus:senator_biden	0.0506
politicianus:palin	0.0302
stateorprovince:california	0.0269
politicaloffice:office	0.0173
politician:obama	0.0162
politicianus:mccain	0.0161
politicianus:barack_obama	0.0141

Personalized PageRank with Approach 1

Total iterations : 85

Node ID	PPR
politicianus:biden	0.057
politician:clinton	0.0531
politicianus:joe_biden	0.0519
politicianus:senator_biden	0.0506
politicianus:palin	0.0302
stateorprovince:california	0.0269
politicaloffice:office	0.0173
politician:obama	0.0162
politicianus:mccain	0.0161
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 26th by 10:00 AM
  - We do not accept late submissions