Part 1: Node2Vec and Logistic Regression

Assignment 2

Name: Saikat Moi

Roll Number: 20CS10050

In [ ]:
```
# Importing nessesary Libraries

import networkx as nx # to visulaize and load graph data
import random
import numpy as np
from typing import List
from tqdm import tqdm
from gensim.models.word2vec import Word2Vec # for implimenting Skip-Gram Model

import matplotlib.pyplot as plt

from IPython.display import display
from PIL import Image
```

In [ ]:
```
# graph: Accepts a graph as input.
# probs: Requires an empty dictionary to calculate probabilities for all neighbo
# p: Returns a parameter.
# q: Represents an in-out parameter.

def compute_probabilities(graph, probs, p, q):
    G = graph
    for source_node in G.nodes():
        for current_node in G.neighbors(source_node):
            probs_ = list()
            for destination in G.neighbors(current_node):

                if source_node == destination:
                    prob_ = G[current_node][destination].get('weight',1) * (1/p)
                elif destination in G.neighbors(source_node):
                    prob_ = G[current_node][destination].get('weight',1)
                else:
                    prob_ = G[current_node][destination].get('weight',1) * (1/q)

                probs_.append(prob_)

            probs[source_node]['probabilities'][current_node] = probs_/np.sum(pr

    return probs
```

In [ ]:
```
# graph: Requires a graph as input.
# probs: Indicates computed probabilities.
# max_walks: Specifies the maximum number of walks allowed per node.
# walk_len: Denotes the maximum length of each walk.
```

```python
def generate_random_walks(graph,probs,max_walks, walk_len):


    G = graph
    walks = list()
    for start_node in G.nodes():
        for i in range(max_walks):

            walk = [start_node]
            walk_options = list(G[start_node])
            if len(walk_options)==0:
                break
            first_step = np.random.choice(walk_options)
            walk.append(first_step)

            for k in range(walk_len-2):
                walk_options = list(G[walk[-1]])
                if len(walk_options)==0:
                    break
                probabilities = probs[walk[-2]]['probabilities'][walk[-1]]
                next_step = np.random.choice(walk_options, p=probabilities)
                walk.append(next_step)

            walks.append(walk)
    np.random.shuffle(walks)
    walks = [list(map(str,walk)) for walk in walks]

    return walks
```

In [ ]:
```python
def Node2Vec_impl(generated_walks,window_size,embedding_vector_size):
    model = Word2Vec(sentences=generated_walks, window=window_size, vector_size=
    return model
```

In [ ]:
```python
def load_data():
    # Read .content file to get node features and labels
    with open("../../dataset/cora.content", "r") as content_file:
        content_lines = content_file.readlines()

    # Read .cites files to build the citation graph
    train_cites = np.loadtxt("../../dataset/cora_train.cites", dtype=int)
    test_cites = np.loadtxt("../../dataset/cora_test.cites", dtype=int)

    # Create a directed graph
    citation_graph = nx.DiGraph()
    train_graph = nx.DiGraph()
    test_graph = nx.DiGraph()

    # Add edges to the graph
    for paper1, paper2 in train_cites:
        citation_graph.add_edge(paper2, paper1)  # Adding the edge with correct
        train_graph.add_edge(paper2, paper1)



    for paper1, paper2 in test_cites:
        citation_graph.add_edge(paper2, paper1)  # Adding the edge with correct
        test_graph.add_edge(paper2, paper1)
```

```python
    # Extract node features and labels
    node_features = []
    node_labels = {}
    for line in content_lines:
        data = line.strip().split()
        paper_id = int(data[0])
        class_label = data[-1]
        node_features.append([int(x) for x in data[1:-1]])
        node_labels[paper_id] = class_label

    return node_features, node_labels, citation_graph,train_graph,test_graph
```

In [ ]:
```python
node_features, node_labels, citation_graph,train_graph,test_graph= load_data()
```

In [ ]:
```python
G = citation_graph
```

In [ ]:
```python
from collections import defaultdict

probs = defaultdict(dict)
for node in G.nodes():
    probs[node]['probabilities'] = dict()
```

In [ ]:
```python
cp = compute_probabilities(G,probs,1,1)

walks = generate_random_walks(G,cp,50,10)
```

In [ ]:
```python
# generate embeddings

n2v_emb = Node2Vec_impl(walks,5,32)
```

In [ ]:
```python
X_train = []
y_train=[]
for node_id in train_graph.nodes():
    if str(node_id) in n2v_emb.wv:
        embedding = n2v_emb.wv.get_vector(str(node_id))
        X_train.append(embedding)
        y_train.append(node_labels[node_id])

X_test = []
y_test=[]

for node_id in test_graph.nodes():
    if str(node_id) in n2v_emb.wv:
        embedding = n2v_emb.wv.get_vector(str(node_id))
        X_test.append(embedding)
        y_test.append(node_labels[node_id])
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix,accuracy_sco
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

# Step 5: Evaluate the classifier
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7524091919940696
```

In [ ]:
```
report = classification_report(y_test, y_pred)

# Print the classification report
print(report)

# Write the classification report to a file
with open("lr_metrics.txt", "w") as file:
    file.write(report)
```

```
                         precision    recall  f1-score   support

             Case_Based       0.73      0.61      0.67       166
     Genetic_Algorithms       0.88      0.87      0.88       210
        Neural_Networks       0.70      0.85      0.77       366
  Probabilistic_Methods       0.84      0.79      0.81       207
 Reinforcement_Learning       0.86      0.71      0.78       128
          Rule_Learning       0.69      0.56      0.62        98
                 Theory       0.62      0.63      0.63       174

               accuracy                           0.75      1349
              macro avg       0.76      0.72      0.74      1349
           weighted avg       0.76      0.75      0.75      1349
```