Working with CoreRec and VishGraphs

Node : Some functionalities are yet to come but already been mentioned in below use cases

`Here are the questions marked as "Unreleased" written as intervals:

Unreleased Features
Q9 to Q12
Q16 to Q20
Q22 to Q25
Q27 to Q30
Q32 to Q35
Q36 to Q40
Q41 to Q44
Q45 to Q49

`

Q1: Generate a Random Graph and Visualize it in 2D

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q2: Print Adjacency Matrix and Find Top Nodes

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
print("The Adj matrix is :", adj_matrix)
top_nodes = vg.find_top_nodes(adj_matrix)
vg.draw_graph(adj_matrix, top_nodes=top_nodes)
```

Q3: Generate a Bipartite Graph and Visualize it with Cosine Similarity

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship_with_cosine(matrix)
```

Q4: Recommend Nodes to Node 7 and Visualize it

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix)
vg.draw_graph(adj_matrix, top_nodes=top_nodes)

num_layers = 3
d_model = 128
num_heads = 4
d_feedforward = 256
input_dim = adj_matrix.shape  # Input dimension should match the number
of nodes in the graph
model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)

node_index = 2
predictions = cs.predict(model, adj_matrix, node_index, top_k=5)
print(f"Recommended nodes for node {node_index}: {predictions}")
print("Popular Nodes are :", top_nodes)

vg.draw_graph_3d(adj_matrix, top_nodes=top_nodes,
recommended_nodes=predictions)
```

Q5: Draw a 3D Graph with Recommended Nodes Highlighted

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix)
vg.draw_graph_3d(adj_matrix, top_nodes=top_nodes)
```

Q6: Visualize a Bipartite Graph with Communities

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship_with_cosine(matrix)
```

Q7: Train a Graph Transformer Model

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

num_layers = 3
d_model = 128
num_heads = 4
d_feedforward = 256
input_dim = adj_matrix.shape  # Input dimension should match the number
of nodes in the graph
model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q8: Predict Node Connections using a Trained Model

```python
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
node_index = 2
predictions = cs.predict(model, adj_matrix, node_index, top_k=5)
print(f"Recommended nodes for node {node_index}: {predictions}")
```

Q9: Visualize a Graph with Node Labels (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q10: Generate a Random Graph with a Specific Seed

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q11: Find Top Nodes in a Graph

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix)
print(f"The top nodes are: {top_nodes}")
```

Q12: Visualize a Graph in 3D with Node Labels (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph_3d(adj_matrix)
```

Q13: Train a Model with a Custom Dataset

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q14: Visualize a Bipartite Graph

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship(matrix)
```

Q15: Recommend Nodes to a Specific Node

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
node_index = 2
predictions = cs.predict(model, adj_matrix, node_index, top_k=5)
print(f"Recommended nodes for node {node_index}: {predictions}")
```

Q16: Draw a Graph with Custom Node Colors (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, node_colors=['red', 'green', 'blue'])
```

Q17: Generate a Random Graph with a Specific Number of Nodes

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q18: Visualize a Graph with Edge Weights (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, edge_weights=True)
```

Q19: Train a Model with a Custom Loss Function

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

 # Initialize Transformer Model
num_layers = 3
d_model = 128
num_heads = 4
d_feedforward = 256
input_dim = adj_matrix.shape[0] # Input dimension should match the
number of nodes in the graph
```

```
model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)

# Custom Loss Function
class CustomLoss(nn.Module):
    # Define your custom loss function here
    return torch.mean((output - target) ** 2)  # Example: Mean Squared
Error


# Define your loss function, optimizer, and other training parameters
criterion = CustomLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150


# training the model
cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q20: Visualize a Graph with Node Sizes (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, node_sizes=[100, 200, 300])
```

Q21: Generate a Random Graph with a Specific Probability of Edge Formation (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332, edge_prob=0.5)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q22: Find Top Nodes in a Graph with a Custom Threshold (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix, threshold=0.5)
print(f"The top nodes are: {top_nodes}")
```

Q23: Visualize a Graph in 3D with Custom Node Colors (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph_3d(adj_matrix, node_colors=['red', 'green', 'blue'])
```

Q24: Train a Model with a Custom Optimizer

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q25: Visualize a Bipartite Graph with Custom Node Colors (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship(matrix, node_colors=['red', 'green',
'blue'])
```

Q26: Recommend Nodes to a Specific Node with a Custom Model

```
import numpy as np
import core_rec as cs
import vish_graphs as vg

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader


# Generate random graph and load adjacency matrix
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix)
# vg.draw_graph(adj_matrix, top_nodes=top_nodes)
```

```
class SimpleNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Convert adjacency matrix to a PyTorch tensor of dtype float32
adj_matrix = torch.tensor(adj_matrix, dtype=torch.float32)

# Initialize Transformer Model
num_layers = 3
d_model = 128
num_heads = 4
d_feedforward = 256
input_dim = adj_matrix.shape[1]  # Input dimension should match the
number of features per node
hidden_dim = 64  # Define hidden layer dimension
output_dim = adj_matrix.shape[1]  # Output dimension should match the
input dimension

# model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
model = SimpleNN(input_dim, hidden_dim, output_dim)

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

# Define your loss function, optimizer, and other training parameters
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

# Train the model
cs.train_model(model, data_loader, criterion, optimizer, num_epochs)

# Use the trained model for node recommendations
node_index = 2
predictions = cs.predict(model, adj_matrix, node_index, top_k=5)
print(f"Recommended nodes for node {node_index}: {predictions}")
print("Popular Nodes are:", top_nodes)

vg.draw_graph_3d(adj_matrix, top_nodes=top_nodes,
recommended_nodes=predictions)
```

Q27: Draw a Graph with Custom Edge Colors (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, edge_colors=['red', 'green', 'blue'])
```

Q28: Generate a Random Graph with a Specific Number of Edges

```
file_path = vg.generate_random_graph(72, seed=332, num_edges=100)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q29: Visualize a Graph with Node Shapes (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, node_shapes=['o', 's', 'D'])
```

Q30: Train a Model with a Custom Batch Size

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=32, shuffle=True)

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q31: Visualize a Graph in 3D with Custom Edge Colors (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph_3d(adj_matrix, edge_colors=['red', 'green', 'blue'])
```

Q32: Find Top Nodes in a Graph with a Custom Metric (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix, metric='degree_centrality')
print(f"The top nodes are: {top_nodes}")
```

Q33: Generate a Random Graph with a Specific Community Structure (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332,
community_structure=True)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q34: Visualize a Bipartite Graph with Custom Edge Weights(Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship(matrix, edge_weights=True)
```

Q35: Recommend Nodes to a Specific Node with a Custom Threshold

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
node_index = 2
predictions = cs.predict(model, adj_matrix, node_index, top_k=5,
threshold=0.5)
print(f"Recommended nodes for node {node_index}: {predictions}")
```

Q36: Draw a Graph with Custom Node Labels (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, node_labels=['A', 'B', 'C'])
```

Q37: Train a Model with a Custom Number of Layers (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

model = cs.GraphTransformer(num_layers=5, d_model=128, num_heads=4,
d_feedforward=256, input_dim=input_dim)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q38: Visualize a Graph in 3D with Custom Node Labels (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph_3d(adj_matrix, node_labels=['A', 'B', 'C'])
```

Q39: Find Top Nodes in a Graph with a Custom Number of Nodes (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix, num_nodes=10)
print(f"The top nodes are: {top_nodes}")
```

Q40: Generate a Random Graph with a Specific Degree Distribution (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332,
degree_distribution='power_law')
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q41: Visualize a Bipartite Graph with Custom Node Shapes

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship(matrix, node_shapes=['o', 's', 'D'])
```

Q42: Recommend Nodes to a Specific Node with a Custom Model and Threshold

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import core_rec as cs
import vish_graphs as vg
from torch.utils.data import Dataset, DataLoader


# Load adjacency matrix
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
num_layers = 2
d_model = 128
num_heads = 8
d_feedforward = 512
input_dim = len(adj_matrix[0])

#     # Convert adjacency matrix to dataset
graph_dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(graph_dataset, batch_size=5, shuffle=True)

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)

# Specify the node index for recommendation
node_index = 2
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
num_epochs = 10
cs.train_model(model, data_loader, criterion, optimizer, num_epochs)


# Predict recommendations using the simple neural network
# predictions = cs.predict(model, adj_matrix, node_index, top_k=5)
predictions = cs.predict(model, adj_matrix, node_index, top_k=5,
threshold=0.1)
print(f"Recommended nodes for node {node_index}: {predictions}")

# #[18, 26, 7, 39] 0.9
# #[37, 26, 3, 39, 18] 0.8
# #[37, 11, 26, 9, 22] 0.4
# #[34, 18, 35, 37, 22] 0.1
```

Q43: Draw a Graph with Custom Edge Labels (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix, edge_labels=['A', 'B', 'C'])
```

Q44: Train a Model with a Custom Learning Rate

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")

dataset = cs.GraphDataset(adj_matrix)
data_loader = DataLoader(dataset, batch_size=16, shuffle=True)

model = cs.GraphTransformer(num_layers, d_model, num_heads,
d_feedforward, input_dim)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
num_epochs = 150

cs.train_model(model, data_loader, criterion, optimizer, num_epochs)
```

Q45: Visualize a Graph in 3D with Custom Edge Labels (Unreleased)

```
file_path = vg.generate_random_graph(40, seed=122)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph_3d(adj_matrix, edge_labels=['A', 'B', 'C'])
```

Q46: Find Top Nodes in a Graph with a Custom Metric and Threshold (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
adj_matrix = np.loadtxt(file_path, delimiter=",")
top_nodes = vg.find_top_nodes(adj_matrix, metric='degree_centrality',
threshold=0.5)
print(f"The top nodes are: {top_nodes}")
```

Q47: Generate a Random Graph with a Specific Clustering Coefficient (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332,
clustering_coefficient=0.5)
adj_matrix = np.loadtxt(file_path, delimiter=",")
vg.draw_graph(adj_matrix)
```

Q48: Visualize a Bipartite Graph with Custom Edge Labels (Unreleased)

```
file_path = vg.generate_random_graph(72, seed=332)
matrix = vg.bipartite_matrix_maker(file_path)
vg.show_bipartite_relationship(matrix, edge_labels=['A', 'B', 'C'])
```

Q49: Recommend Nodes to a Specific Node with a Custom Model, Threshold, and Metric (Unreleased)

```
I am working on it !!!
```