TGCI: Temporal Graph Causal Inference

import torch

import torch.nn as nn

import torch.optim as optim

import numpy as np

import pandas as pd

from torch\_geometric\_temporal.signal import DynamicGraphTemporalSignal

from torch\_geometric\_temporal.nn.recurrent import TGNMemory, TGNMessagePassing

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import NearestNeighbors

from econml.dml import LinearDML

from sklearn.metrics import mean\_absolute\_error, r2\_score

# ==========================================

# 1. Temporal Graph Encoder (TGE)

# ==========================================

class TemporalGraphEncoder(nn.Module):

def \_\_init\_\_(self, node\_features, memory\_dim=64, time\_dim=16, embedding\_dim=128):

super().\_\_init\_\_()

self.memory\_dim = memory\_dim

self.time\_dim = time\_dim

self.embedding\_dim = embedding\_dim

self.memory = TGNMemory(

num\_nodes=node\_features,

raw\_message\_dim=embedding\_dim,

memory\_dimension=memory\_dim,

time\_dimension=time\_dim

)

self.gnn = TGNMessagePassing(

in\_channels=memory\_dim,

out\_channels=embedding\_dim

)

def forward(self, x, edge\_index, edge\_attr, t):

memory = self.memory(x, t)

out = self.gnn(memory, edge\_index, edge\_attr)

return out

# ==========================================

# 2. Causal Identification Layer (CIL)

# ==========================================

def causal\_matching(z, treatment, covariates):

"""

Estimate propensity scores and perform nearest-neighbor matching.

"""

logit = LogisticRegression(max\_iter=1000)

logit.fit(covariates, treatment)

propensity = logit.predict\_proba(covariates)[:, 1]

# Nearest neighbor matching

treated\_idx = np.where(treatment == 1)[0]

control\_idx = np.where(treatment == 0)[0]

nbrs = NearestNeighbors(n\_neighbors=1).fit(propensity[control\_idx].reshape(-1, 1))

\_, indices = nbrs.kneighbors(propensity[treated\_idx].reshape(-1, 1))

matched\_pairs = list(zip(treated\_idx, control\_idx[indices[:, 0]]))

return matched\_pairs, propensity

# ==========================================

# 3. Counterfactual Simulation Engine (CSE)

# ==========================================

def simulate\_counterfactual(model, data, intervention\_mask):

"""

Predict outcomes under no-intervention condition.

"""

y\_true, y\_pred\_cf = [], []

for t in range(len(data.features)):

x\_t = data.features[t]

edge\_index\_t = data.edge\_indices[t]

edge\_attr\_t = data.edge\_weights[t]

y = data.targets[t]

pred = model(x\_t, edge\_index\_t, edge\_attr\_t, t)

y\_true.append(y.mean().item())

y\_pred\_cf.append(pred.mean().item() \* (1 - intervention\_mask[t])) # simulate no intervention

return np.array(y\_true), np.array(y\_pred\_cf)

# ==========================================

# 4. TGCI Training + Causal Estimation

# ==========================================

def train\_tgci(data):

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model = TemporalGraphEncoder(node\_features=data.features[0].shape[1]).to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-3)

print("Training TGCI model...")

for epoch in range(5): # Simplified training loop

total\_loss = 0

for t in range(len(data.features)):

x\_t = data.features[t].to(device)

edge\_index\_t = data.edge\_indices[t].to(device)

edge\_attr\_t = data.edge\_weights[t].to(device)

y\_t = data.targets[t].to(device)

optimizer.zero\_grad()

y\_pred = model(x\_t, edge\_index\_t, edge\_attr\_t, t)

loss = nn.MSELoss()(y\_pred.mean(), y\_t.mean())

loss.backward()

optimizer.step()

total\_loss += loss.item()

print(f"Epoch {epoch+1}: Loss={total\_loss:.4f}")

return model

# ==========================================

# 5. Example: Running TGCI on Dummy Data

# ==========================================

def generate\_synthetic\_temporal\_graph(num\_nodes=100, timesteps=10):

edge\_indices, edge\_weights, features, targets = [], [], [], []

for t in range(timesteps):

edges = torch.randint(0, num\_nodes, (2, num\_nodes))

weights = torch.rand(num\_nodes)

x = torch.rand(num\_nodes, 8)

y = torch.rand(num\_nodes, 1)

edge\_indices.append(edges)

edge\_weights.append(weights)

features.append(x)

targets.append(y)

return DynamicGraphTemporalSignal(edge\_indices, edge\_weights, features, targets)

if \_\_name\_\_ == "\_\_main\_\_":

data = generate\_synthetic\_temporal\_graph()

tgci\_model = train\_tgci(data)

intervention\_mask = np.random.randint(0, 2, size=len(data.features))

y\_true, y\_cf = simulate\_counterfactual(tgci\_model, data, intervention\_mask)

# Causal Effect Estimation (ATE)

att = np.mean(y\_true - y\_cf)

print(f"\nEstimated ATT (Causal Effect): {att:.4f}")