Реализация рекомендательной системы на основе GNN (Graph Neural Network)

Цель: научиться создавать рекомендательную систему на основе графовых нейронных сетей (GNN), таких как легковесная графовая нейронная сеть LightGCN, IRGNN (Item Relationship Graph Neural Network), применяя методологию анализа графа взаимодействий пользователей и товаров/услуг, научиться применять её для прогнозирования предпочтений пользователей, оценить качество полученной модели на реальных данных.

1 часть – общий пример (1 балл)

Ход работы

1. Установите следующие библиотеки для применения GNN

pip install torch_geometric numpy pandas matplotlib sklearn scikit-network networkx

2. Выполните выбор набора данных

Используем открытый набор данных MovieLens. Скачайте данные здесь: https://grouplens.org/datasets/movielens/

3. Создание графа взаимодействия

Построение графа, вершины которого представляют собой пользователей и фильмы, а рёбра — оценки фильмов пользователями.

```
import pandas as pd
from torch_geometric.data import Data
import torch

# Загрузка данных
ratings = pd.read_csv('ml-latest-small/ratings.csv')
movies = pd.read_csv('ml-latest-small/movies.csv')

# Создаем список узлов
users = ratings['userId'].unique()
items = movies['movieId'].unique()

# Присваиваем каждому узлу уникальный индекс
node_id_map = {uid: i for i, uid in enumerate(users)}
item_id_map = {iid: len(users)+i for i, iid in enumerate(items)}
# Создание списка рёбер
```

```
edges = []
for _, row in ratings.iterrows():
    user_idx = node_id_map[row['userId']]
    item_idx = item_id_map[row['movieId']]
    edges.append((user_idx, item_idx))

edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()
data = Data(edge_index=edge_index)
```

4. Подготовка данных для модели

Нормализация оценок. Нормализуйте значения оценок, используя методы нормализации (например, min-max).

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
ratings['rating'] = scaler.fit_transform(ratings[['rating']])
```

Разделение на тренировочный и тестовый наборы. Разделите данные на две части: одна для тренировки, вторая для тестирования.

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(data.edge_index.t(), test_size=0.2, random_state=42)
```

5. Реализация модели GNN

Используя библиотеку PyTorch Geometric, создайте простую архитектуру GNN-модели для предсказания рейтингов.

```
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class RecommenderModel(torch.nn.Module):
    def __init__(self, num_features, hidden_channels):
        super().__init__()
        self.conv1 = GCNConv(num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, 1)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
```

```
        return x.squeeze()
```

Обучаем модель на созданных данных:

```
model = RecommenderModel(num_features=len(users), hidden_channels=16)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.MSELoss()

def train():
    model.train()
    optimizer.zero_grad()
    out = model(train_data.x, train_data.edge_index)
    loss = criterion(out, train_data.y)
    loss.backward()
    optimizer.step()
    return float(loss)

for epoch in range(1, 101):
    loss = train()
    print(f'Epoch: {epoch}, Loss: {loss:.4f}')
```

6. Оценка рекомендательной системы

Проверьте качество модели, сравнив её прогнозы с фактическими значениями рейтинга.

```
def evaluate(model, data):
    model.eval()
    with torch.no_grad():
        pred = model(data.x, data.edge_index)
        mse_loss = criterion(pred, data.y)
    return float(mse_loss)

test_mse = evaluate(model, test_data)
print(f'Test MSE: {test_mse:.4f}')
```

7. Самостоятельное задание

Проведите гиперпараметризацию модели, выбрав оптимальное количество слоёв и размерность скрытых признаков.

Добавьте дополнительные признаки пользователей и объектов (пол, возраст, жанр фильма и др.) и посмотрите влияние на точность рекомендаций.

2 часть – Применение легковесной графовой нейронной сети LightGCN (1 балл)

Ход работы

1. **Выбор данных**
   Выберите публичный набор данных MovieLens (ml-1m) или аналогичный набор данных, содержащий рейтинги пользователей относительно различных элементов (фильмов, книг, музыки и т.п.). Далее пример показан на MovieLens.

```
wget http://files.grouplens.org/datasets/movielens/ml-1m.zip
unzip ml-1m.zip
```

2. **Подготовка данных:**

Преобразуйте сырые данные в удобный формат, пригодный для ввода в LightGCN.

```
import pandas as pd
import os

dataset_path = 'ml-1m'
ratings_df = pd.read_csv(os.path.join(dataset_path, 'ratings.dat'), sep='::', engine='python',
                         names=['UserID', 'ItemID', 'Rating', 'Timestamp'])

# Приведение датасета к виду, необходимому для загрузки в LightGCN
lightgcn_dataset = {
   'user': ratings_df.UserID.values,
   'item': ratings_df.ItemID.values,
   'rating': ratings_df.Rating.values.astype(float),
}
```

3. **Обучение модели:**

Создайте экземпляр класса LightGCN и установите основные гиперпараметры.

```
from lightgcn.models.lightgcn import LightGCN
from lightgcn.utils.dataset import DatasetLightGCN
from lightgcn.utils.evaluation import Evaluator
from pytorch_lightning.callbacks import EarlyStopping
from pytorch_lightning.loggers import TensorBoardLogger
import pytorch_lightning as pl

# Настройка моделей и базовых параметров
hyper_params = dict(
   embedding_dim=64, # Размерность векторных представлений
   layers_num=3,     # Количество слоев GNN
   dropout_rate=0.1, # Коэффициент дропаут
   batch_size=1024,  # Размер батча
   learning_rate=0.001, # Скорость обучения
```

```
    epochs=50         # Число эпох
)

dataset = DatasetLightGCN(lightgcn_dataset)
evaluator = Evaluator(k_list=[10], metrics=['NDCG'], mode='test')
early_stop_callback = EarlyStopping(patience=5, monitor="val_ndcg@10")
logger = TensorBoardLogger("logs", name="lightgcn_logs")

model = LightGCN(hyper_params)
trainer = pl.Trainer(max_epochs=hyper_params["epochs"],
callbacks=[early_stop_callback],
                     logger=logger, gpus=-1 if torch.cuda.is_available() else None)

trainer.fit(model, dataset)
```

Отследите процесс обучения модели через метрики NDCG и Precision, записанные в лог-файлы TensorBoard.

4. **Тестирование и оценка результатов:**

Оцените эффективность вашей модели на заранее отложенном тестовом наборе данных.

```
results = trainer.test(model, datamodule=dataset)
print(results)
```

Проанализируйте полученные результаты по показателям NDCG@k и другим метрикам.

Самостоятельное задание

Проделайте шаги 1-4 на предложенном наборе данных fashion.zip

Исследуйте возможности улучшения точности рекомендации путём изменения следующих парамеров:

количества слоёв LightGCN;

размера скрытого представления;

метода агрегации данных.

3 часть – Применение графовой нейронной сети IRGNN для реализации рекомендательной системы (2 балла)

Ход работы

1. Выбор и подготовка данных Мы будем использовать стандартный набор данных MovieLens, доступный на сайте GroupLens Research. Скачайте набор данных и подготовьте его для дальнейшего использования.
wget http://files.grouplens.org/datasets/movielens/ml-100k/u.data.gz
gunzip u.data.gz
Загрузите данные и разделите их на два массива: один для создания графа взаимосвязей, второй для проверки рекомендаций.

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('u.data', delimiter='\t', header=None, names=["user_id", "item_id", "rating", "timestamp"])
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

2. Формирование графа взаимосвязей

Здесь мы создадим ориентированный граф взаимоотношений между предметами (фильмами), основываясь на совместных просмотрах пользователями.

```
import networkx as nx
import numpy as np

graph = nx.DiGraph()

# Формируем связи между фильмами на основе общих пользователей
for index, group in df.groupby(['item_id']):
    graph.add_node(index)
    common_users = set(group['user_id'])
    for other_item_id, other_group in df.groupby(['item_id']):
        if other_item_id != index and not graph.has_edge(index, other_item_id):
            intersection = common_users.intersection(set(other_group['user_id']))
            weight = len(intersection)
            if weight > 0:
                graph.add_edge(index, other_item_id, weight=weight)
```

Далее преобразуем полученный граф в формат, совместимый с используемой вами библиотекой (например, DGL или PyTorch Geometric).

```
import dgl
```

```
dgl_graph = dgl.from_networkx(graph, edge_attrs=['weight'])
```

3. Реализация модели IRGNN

Теперь приступим к созданию модели IRGNN с использованием выбранной библиотеки для работы с графами.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from dgl.nn.pytorch import GraphConv

class IRGNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(IRGNN, self).__init__()
        self.conv1 = GraphConv(input_dim, hidden_dim)
        self.conv2 = GraphConv(hidden_dim, output_dim)

    def forward(self, g, features):
        h = F.relu(self.conv1(g, features))
        h = F.relu(self.conv2(g, h))
        return h
```

Затем инициализируем модель и запустим обучение.

```
input_dim = 16  # размерность входных признаков
hidden_dim = 32
output_dim = 8

model = IRGNN(input_dim, hidden_dim, output_dim)

features = torch.randn(dgl_graph.number_of_nodes(), input_dim)
labels = torch.zeros(dgl_graph.number_of_nodes())

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(100):
    logits = model(dgl_graph, features)
    loss = F.cross_entropy(logits, labels)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print(f"Epoch {epoch}: Loss {loss.item():.4f}")
```

4. Генерация рекомендаций

Получив обученную модель, мы можем формировать персональные рекомендации для каждого пользователя. Для этого нужно рассчитать предпочтения пользователя на основе представленного графа и ранжировать предметы (фильмы) согласно рекомендациям.

```
def recommend_items(user_id, top_k=10):
    items_rated_by_user = df[df.user_id == user_id]['item_id']
    unrated_items = list(set(df.item_id.unique()) - set(items_rated_by_user))

    predictions = {}
    for item_id in unrated_items:
        predictions[item_id] = predict_rating(user_id, item_id)

    sorted_predictions = sorted(predictions.items(), key=lambda x: x[1], reverse=True)
    return [item[0] for item in sorted_predictions[:top_k]]

recommendations = recommend_items(test_df.iloc[0].user_id)
print(recommendations)
```

Самостоятельное задание

Примените предложенную реализацию на другом открытом наборе данных lastfm.zip.

Попробуйте реализовать расширенные версии модели (например, добавив веса ребрам графа или учитывая временные факторы при формировании связей).