Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Методы машинного обучения»

Отчет по лабораторной работе №6

«Разработка системы предсказаний поведения на основании графовых моделей»

Выполнил:
студент группы ИУ5-23Б
Белоусов Евгений
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2022 г.

Описание задания

Цель лабораторной работы: обучение работе с предварительной обработкой графовых типов данных и обучением нейронных сетей на графовых данных.

1. Подготовить датасет графовых данных
2. Подобрать модель и гиперпараметры обучения для получения качества AUC > 0.65

```
# Slow method of installing pytorch geometric
# !pip install torch_geometric
# !pip install torch_sparse
# !pip install torch_scatter

# Install pytorch geometric
!pip install torch-sparse -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-cluster -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-spline-conv -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-geometric -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-scatter==2.0.8 -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
```

```
import numpy as np
import pandas as pd
import pickle
import csv
import os

from sklearn.preprocessing import LabelEncoder

import torch

# PyG - PyTorch Geometric
from torch_geometric.data import Data, DataLoader, InMemoryDataset

from tqdm import tqdm


RANDOM_SEED = 42 #@param { type: "integer" }
BASE_DIR = '/content/' #@param { type: "string" }
np.random.seed(RANDOM_SEED)
```

RANDOM_SEED: 42

BASE_DIR: '/content/'

```
[5]  # Check if CUDA is available for colab
     torch.cuda.is_available()
```

```
True
```

```
[6]  # Unpack files from zip-file
     import zipfile
     with zipfile.ZipFile(BASE_DIR + 'yoochoose-data-lite.zip', 'r') as zip_ref:
         zip_ref.extractall(BASE_DIR)
```

Анализ исходных данных

```
# Read dataset of items in store
df = pd.read_csv(BASE_DIR + 'yoochoose-clicks-lite.dat')
# df.columns = ['session_id', 'timestamp', 'item_id', 'category']
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (3) have mixed types.Specify
  exec(code_obj, self.user_global_ns, self.user_ns)
```

	session_id	timestamp	item_id	category
0	9	2014-04-06T11:26:24.127Z	214576500	0
1	9	2014-04-06T11:28:54.654Z	214576500	0
2	9	2014-04-06T11:29:13.479Z	214576500	0
3	19	2014-04-01T20:52:12.357Z	214561790	0
4	19	2014-04-01T20:52:13.758Z	214561790	0

[8]
```
# Read dataset of purchases
buy_df = pd.read_csv(BASE_DIR + 'yoochoose-buys-lite.dat')
# buy_df.columns = ['session_id', 'timestamp', 'item_id', 'price', 'quantity']
buy_df.head()
```

	session_id	timestamp	item_id	price	quantity
0	420374	2014-04-06T18:44:58.314Z	214537888	12462	1
1	420374	2014-04-06T18:44:58.325Z	214537850	10471	1
2	489758	2014-04-06T09:59:52.422Z	214826955	1360	2
3	489758	2014-04-06T09:59:52.476Z	214826715	732	2
4	489758	2014-04-06T09:59:52.578Z	214827026	1046	1

[9]
```
# Filter out item session with length < 2
df['valid_session'] = df.session_id.map(df.groupby('session_id')['item_id'].size() > 2)
df = df.loc[df.valid_session].drop('valid_session',axis=1)
df.nunique()
```

```
session_id    1000000
timestamp     5557758
item_id         37644
category          275
dtype: int64
```

```
# Randomly sample a couple of them
NUM_SESSIONS = 50000 #@param { type: "integer" }
sampled_session_id = np.random.choice(df.session_id.unique(), NUM_SESSIONS, replace=False)
df = df.loc[df.session_id.isin(sampled_session_id)]
df.nunique()
```

NUM_SESSIONS: 50000

```
session_id     50000
timestamp     278442
item_id        18461
category         110
dtype: int64
```

[11]
```
# Average length of session
df.groupby('session_id')['item_id'].size().mean()
```

```
5.56902
```

```
# Encode item and category id in item dataset so that ids will be in range (0,len(df.item.unique()))
item_encoder = LabelEncoder()
category_encoder = LabelEncoder()
df['item_id'] = item_encoder.fit_transform(df.item_id)
df['category']= category_encoder.fit_transform(df.category.apply(str))
df.head()
```

	session_id	timestamp	item_id	category
0	9	2014-04-06T11:26:24.127Z	3496	0
1	9	2014-04-06T11:28:54.654Z	3496	0
2	9	2014-04-06T11:29:13.479Z	3496	0
102	171	2014-04-03T17:45:25.575Z	10049	0
103	171	2014-04-03T17:45:33.177Z	10137	0

[13]
```
# Encode item and category id in purchase dataset
buy_df = buy_df.loc[buy_df.session_id.isin(df.session_id)]
buy_df['item_id'] = item_encoder.transform(buy_df.item_id)
buy_df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
  This is separate from the ipykernel package so we can avoid doing imports until
```

	session_id	timestamp	item_id	price	quantity
46	489491	2014-04-06T12:41:34.047Z	12633	1046	4
47	489491	2014-04-06T12:41:34.091Z	12634	627	2
61	70353	2014-04-06T10:55:06.086Z	14345	41783	1
62	489671	2014-04-03T15:48:37.392Z	12489	4188	1
63	489671	2014-04-03T15:59:35.495Z	12489	4188	1

[14]
```
# Get item dictionary with grouping by session
buy_item_dict = dict(buy_df.groupby('session_id')['item_id'].apply(list))
buy_item_dict
```

```
2284789: [526, 1412, 13283, 13287],
2289552: [13164, 13168],
2290431: [13498],
2290698: [11827, 11824],
2291707: [15758, 15758],
2292108: [18460],
2292426: [11208],
2292499: [13156, 13108],
2293224: [7964, 7965],
2294903: [13494],
2295932: [13110],
2300511: [13167, 13164, 9691, 13286],
2300966: [14062, 1121],
2301182: [12974, 10465, 1819],
2301826: [13164, 13167, 13286, 13166],
2302889: [2125],
2303773: [13166, 13164, 13290, 13288, 15921],
2305544: [13164, 13168],
2306951: [13164, 13164, 13286, 13168, 13166, 13167, 13285],
2307524: [13164, 13286, 13167],
2310501: [13168, 8393, 13168],
2311333: [13285, 13165, 13168],
2312773: [13164, 13286, 9699],
2316132: [7877],
2317759: [14951, 7439, 7439, 14951],
2318429: [18460],
2318877: [11221, 12355],
2319826: [13279, 13285, 13164],
2320679: [13108],
2322056: [12869, 13510],
2322739: [3718, 8278, 10962],
2323802: [18460],
2324881: [13279, 10606, 10607],
2325937: [18460],
2326769: [12826, 13832],
2330123: [13603, 8531],
2330138: [13286, 13164],
2331598: [13165, 13285, 13066],
2332823: [12791, 11662, 12362, 11839, 13712]
```

Сборка выборки для обучения

```
[15]  1 # Transform df into tensor data
      2 def transform_dataset(df, buy_item_dict):
      3     data_list = []
      4 
      5     # Group by session
      6     grouped = df.groupby('session_id')
      7     for session_id, group in tqdm(grouped):
      8         le = LabelEncoder()
      9         sess_item_id = le.fit_transform(group.item_id)
     10         group = group.reset_index(drop=True)
     11         group['sess_item_id'] = sess_item_id
     12 
     13         #get input features
     14         node_features = group.loc[group.session_id==session_id,
     15                                   ['sess_item_id','item_id','category']].sort_values('sess_item_id')[['item_id','category']].drop_duplicates().values
     16         node_features = torch.LongTensor(node_features).unsqueeze(1)
     17         target_nodes = group.sess_item_id.values[1:]
     18         source_nodes = group.sess_item_id.values[:-1]
     19 
     20         edge_index = torch.tensor([source_nodes,
     21                                    target_nodes], dtype=torch.long)
     22         x = node_features
     23 
     24         #get result
     25         if session_id in buy_item_dict:
     26             positive_indices = le.transform(buy_item_dict[session_id])
     27             label = np.zeros(len(node_features))
     28             label[positive_indices] = 1
     29         else:
     30             label = [0] * len(node_features)
     31 
     32         y = torch.FloatTensor(label)
     33 
     34         data = Data(x=x, edge_index=edge_index, y=y)
     35 
     36         data_list.append(data)
     37 
     38     return data_list
     39 
     40 # Pytorch class for creating datasets
     41 class YooChooseDataset(InMemoryDataset):
     42     def __init__(self, root, transform=None, pre_transform=None):
     43         super(YooChooseDataset, self).__init__(root, transform, pre_transform)
     44         self.data, self.slices = torch.load(self.processed_paths[0])
     45 
     46     @property
     47     def raw_file_names(self):
     48         return []
     49 
     50     @property
     51     def processed_file_names(self):
     52         return [BASE_DIR+'yoochoose_click_binary_100000_sess.dataset']
     53 
     54     def download(self):
     55         pass
     56 
     57     def process(self):
     58         data_list = transform_dataset(df, buy_item_dict)
     59 
     60         data, slices = self.collate(data_list)
     61         torch.save((data, slices), self.processed_paths[0])
```

```
 1 # Prepare dataset
 2 dataset = YooChooseDataset('./')
```

```
Processing...
  0%|          | 0/50000 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please consider converting the list to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at  .../torch/csr
100%|██████████| 50000/50000 [02:59<00:00, 279.07it/s]
Done!
```

Разделение выборки

```
[17]  1 # train_test_split
      2 dataset = dataset.shuffle()
      3 one_tenth_length = int(len(dataset) * 0.1)
      4 train_dataset = dataset[:one_tenth_length * 8]
      5 val_dataset = dataset[one_tenth_length*8:one_tenth_length * 9]
      6 test_dataset = dataset[one_tenth_length*9:]
      7 len(train_dataset), len(val_dataset), len(test_dataset)
```

```
(40000, 5000, 5000)
```

```
[18]  1 # Load dataset into PyG loaders
      2 batch_size= 512
      3 train_loader = DataLoader(train_dataset, batch_size=batch_size)
      4 val_loader = DataLoader(val_dataset, batch_size=batch_size)
      5 test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
```

```
[19]  1 # Load dataset into PyG loaders
      2 num_items = df.item_id.nunique() +1
      3 num_categories = df.category.nunique()+1
      4 num_items , num_categories
```

```
(18461, 109)
```

Настройка модели для обучения

```
[47]  1 embed_dim = 128
      2 from torch_geometric.nn import GraphConv, TopKPooling, GatedGraphConv, SAGEConv, SGConv
      3 from torch_geometric.nn import global_mean_pool as gap, global_max_pool as gmp
      4 import torch.nn.functional as F
      5
      6 class Net(torch.nn.Module):
      7     def __init__(self):
      8         super(Net, self).__init__()
      9         # Model Structure
     10         self.conv1 = GraphConv(embed_dim * 2, 128)
     11         self.pool1 = TopKPooling(128, ratio=0.9)
     12         self.conv2 = GraphConv(128, 128)
     13         self.pool2 = TopKPooling(128, ratio=0.9)
     14         self.conv3 = GraphConv(128, 128)
     15         self.pool3 = TopKPooling(128, ratio=0.9)
     16         self.item_embedding = torch.nn.Embedding(num_embeddings=num_items, embedding_dim=embed_dim)
     17         self.category_embedding = torch.nn.Embedding(num_embeddings=num_categories, embedding_dim=embed_dim)
     18         self.lin1 = torch.nn.Linear(256, 256)
     19         self.lin2 = torch.nn.Linear(256, 128)
     20         self.bn1 = torch.nn.BatchNorm1d(128)
     21         self.bn2 = torch.nn.BatchNorm1d(64)
     22         self.act1 = torch.nn.ReLU()
     23         self.act2 = torch.nn.ReLU()
     24
     25     # Forward step of a model
     26     def forward(self, data):
     27         x, edge_index, batch = data.x, data.edge_index, data.batch
     28
     29         item_id = x[:,:,0]
     30         category = x[:,:,1]
     31
     32
     33         emb_item = self.item_embedding(item_id).squeeze(1)
     34         emb_category = self.category_embedding(category).squeeze(1)
     35
     36         x = torch.cat([emb_item, emb_category], dim=1)
     37         # print(x.shape)
     38         x = F.relu(self.conv1(x, edge_index))
     39         # print(x.shape)
     40         r = self.pool1(x, edge_index, None, batch)
     41         # print(r)
     42         x, edge_index, _, batch, _, _ = self.pool1(x, edge_index, None, batch)
     43         x1 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)
     44
     45         x = F.relu(self.conv2(x, edge_index))
     46
     47         x, edge_index, _, batch, _, _ = self.pool2(x, edge_index, None, batch)
     48         x2 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)
     49
     50         x = F.relu(self.conv3(x, edge_index))
     51
     52         x, edge_index, _, batch, _, _ = self.pool3(x, edge_index, None, batch)
     53         x3 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)
     54
     55         x = x1 + x2 + x3
     56
     57         x = self.lin1(x)
     58         x = self.act1(x)
     59         x = self.lin2(x)
     60         x = F.dropout(x, p=0.5, training=self.training)
     62
     63         outputs = []
     64         for i in range(x.size(0)):
     65             output = torch.matmul(emb_item[data.batch == i], x[i,:])
     66
     67             outputs.append(output)
     68
     69         x = torch.cat(outputs, dim=0)
     70         x = torch.sigmoid(x)
     71
     72         return x
```

```
# Enable CUDA computing
device = torch.device('cuda')
model = Net().to(device)
# Choose optimizer and criterion for learning
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criti = torch.nn.BCELoss()
```

```
import torch.optim as optim
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.1)
```

```
# Train function
def train():
    model.train()

    loss_all = 0
    for data in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        output = model(data)

        label = data.y.to(device)
        loss = criti(output, label)
        loss.backward()
        loss_all += data.num_graphs * loss.item()
        optimizer.step()
        scheduler.step()
    return loss_all / len(train_dataset)
```

```
# Evaluate result of a model
from sklearn.metrics import roc_auc_score
def evaluate(loader):
    model.eval()

    predictions = []
    labels = []

    with torch.no_grad():
        for data in loader:

            data = data.to(device)
            pred = model(data).detach().cpu().numpy()

            label = data.y.detach().cpu().numpy()
            predictions.append(pred)
            labels.append(label)

    predictions = np.hstack(predictions)
    labels = np.hstack(labels)

    return roc_auc_score(labels, predictions)
```

```
# Train a model
NUM_EPOCHS =  50#@param { type: "integer" }
for epoch in tqdm(range(NUM_EPOCHS)):
    loss = train()
    train_acc = evaluate(train_loader)
    val_acc = evaluate(val_loader)
    test_acc = evaluate(test_loader)
    print('Epoch: {:03d}, Loss: {:.5f}, Train Auc: {:.5f}, Val Auc: {:.5f}, Test Auc: {:.5f}'.
          format(epoch, loss, train_acc, val_acc, test_acc))
```

NUM_EPOCHS: 50

```
  2%|▏         | 1/50 [00:37<30:45, 37.67s/it]Epoch: 000, Loss: 0.71581, Train Auc: 0.51631, Val Auc: 0.51478, Test Auc: 0.51530
  4%|▍         | 2/50 [01:14<29:56, 37.42s/it]Epoch: 001, Loss: 0.66867, Train Auc: 0.52383, Val Auc: 0.51480, Test Auc: 0.51697
  6%|▌         | 3/50 [01:52<29:16, 37.37s/it]Epoch: 002, Loss: 0.65921, Train Auc: 0.52405, Val Auc: 0.51494, Test Auc: 0.51847
  8%|▊         | 4/50 [02:29<28:42, 37.45s/it]Epoch: 003, Loss: 0.65895, Train Auc: 0.52417, Val Auc: 0.51503, Test Auc: 0.51867
 10%|▉         | 5/50 [03:06<27:58, 37.30s/it]Epoch: 004, Loss: 0.65802, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 12%|█▏        | 6/50 [03:44<27:22, 37.33s/it]Epoch: 005, Loss: 0.65719, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 14%|█▍        | 7/50 [04:21<26:40, 37.23s/it]Epoch: 006, Loss: 0.65781, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 16%|█▌        | 8/50 [04:58<26:04, 37.26s/it]Epoch: 007, Loss: 0.65548, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 18%|█▊        | 9/50 [05:35<25:22, 37.13s/it]Epoch: 008, Loss: 0.65784, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 20%|██        | 10/50 [06:12<24:43, 37.10s/it]Epoch: 009, Loss: 0.65710, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 22%|██▏       | 11/50 [06:49<24:11, 37.23s/it]Epoch: 010, Loss: 0.65933, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 24%|██▍       | 12/50 [07:26<23:31, 37.15s/it]Epoch: 011, Loss: 0.65728, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 26%|██▌       | 13/50 [08:03<22:51, 37.08s/it]Epoch: 012, Loss: 0.65700, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 28%|██▊       | 14/50 [08:41<22:20, 37.25s/it]Epoch: 013, Loss: 0.65825, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 30%|███       | 15/50 [09:18<21:42, 37.21s/it]Epoch: 014, Loss: 0.65664, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 32%|███▏      | 16/50 [09:55<21:04, 37.18s/it]Epoch: 015, Loss: 0.65738, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 34%|███▍      | 17/50 [10:32<20:25, 37.13s/it]Epoch: 016, Loss: 0.65639, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 36%|███▌      | 18/50 [11:10<19:52, 37.25s/it]Epoch: 017, Loss: 0.65919, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 38%|███▊      | 19/50 [11:47<19:12, 37.18s/it]Epoch: 018, Loss: 0.65605, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 40%|████      | 20/50 [12:24<18:37, 37.26s/it]Epoch: 019, Loss: 0.65833, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
 42%|████▏     | 21/50 [13:02<18:03, 37.36s/it]Epoch: 020, Loss: 0.65868, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 44%|████▍     | 22/50 [13:39<17:20, 37.16s/it]Epoch: 021, Loss: 0.65708, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 46%|████▌     | 23/50 [14:15<16:38, 37.00s/it]Epoch: 022, Loss: 0.65783, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 48%|████▊     | 24/50 [14:52<15:59, 36.89s/it]Epoch: 023, Loss: 0.65689, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 50%|█████     | 25/50 [15:29<15:22, 36.89s/it]Epoch: 024, Loss: 0.65795, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 52%|█████▏    | 26/50 [16:05<14:44, 36.84s/it]Epoch: 025, Loss: 0.65761, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 54%|█████▍    | 27/50 [16:42<14:05, 36.76s/it]Epoch: 026, Loss: 0.65736, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 56%|█████▌    | 28/50 [17:19<13:33, 36.99s/it]Epoch: 027, Loss: 0.65718, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 58%|█████▊    | 29/50 [17:57<12:57, 37.03s/it]Epoch: 028, Loss: 0.65881, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 62%|██████▏   | 31/50 [19:11<11:44, 37.07s/it]Epoch: 030, Loss: 0.65868, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 64%|██████▍   | 32/50 [19:48<11:05, 36.98s/it]Epoch: 031, Loss: 0.65885, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 66%|██████▌   | 33/50 [20:24<10:27, 36.93s/it]Epoch: 032, Loss: 0.65600, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 68%|██████▊   | 34/50 [21:01<09:49, 36.87s/it]Epoch: 033, Loss: 0.65817, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 70%|███████   | 35/50 [21:38<09:12, 36.81s/it]Epoch: 034, Loss: 0.65811, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 72%|███████▏  | 36/50 [22:15<08:35, 36.86s/it]Epoch: 035, Loss: 0.65961, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 74%|███████▍  | 37/50 [22:51<07:58, 36.83s/it]Epoch: 036, Loss: 0.65804, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 76%|███████▌  | 38/50 [23:28<07:21, 36.79s/it]Epoch: 037, Loss: 0.65696, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 78%|███████▊  | 39/50 [24:05<06:43, 36.70s/it]Epoch: 038, Loss: 0.65739, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 80%|████████  | 40/50 [24:41<06:06, 36.67s/it]Epoch: 039, Loss: 0.65704, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 82%|████████▏ | 41/50 [25:18<05:29, 36.63s/it]Epoch: 040, Loss: 0.65618, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 84%|████████▍ | 42/50 [25:54<04:52, 36.61s/it]Epoch: 041, Loss: 0.65725, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 86%|████████▌ | 43/50 [26:31<04:16, 36.58s/it]Epoch: 042, Loss: 0.65865, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 88%|████████▊ | 44/50 [27:08<03:39, 36.66s/it]Epoch: 043, Loss: 0.65803, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 90%|█████████ | 45/50 [27:44<03:02, 36.58s/it]Epoch: 044, Loss: 0.65916, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 92%|█████████▏| 46/50 [28:21<02:26, 36.66s/it]Epoch: 045, Loss: 0.65736, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 94%|█████████▍| 47/50 [28:58<01:50, 36.75s/it]Epoch: 046, Loss: 0.65851, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 96%|█████████▌| 48/50 [29:34<01:13, 36.64s/it]Epoch: 047, Loss: 0.65533, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
 98%|█████████▊| 49/50 [30:11<00:36, 36.68s/it]Epoch: 048, Loss: 0.65820, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51864
100%|██████████| 50/50 [30:48<00:00, 36.97s/it]Epoch: 049, Loss: 0.65629, Train Auc: 0.52417, Val Auc: 0.51506, Test Auc: 0.51865
```

Проверка результата с помощью примеров

[53]
```
# Подход №1 - из датасета
evaluate(DataLoader(test_dataset[40:60], batch_size=10))
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
0.5416666666666666
```

[54]
```
# Подход №2 - через создание сессии покупок
test_df = pd.DataFrame([
      [-1, 15219, 0],
      [-1, 15431, 0],
      [-1, 14371, 0],
      [-1, 15745, 0],
      [-2, 14594, 0],
      [-2, 16972, 11],
      [-2, 16943, 0],
      [-3, 17284, 0]
], columns=['session_id', 'item_id', 'category'])

test_data = transform_dataset(test_df, buy_item_dict)
test_data = DataLoader(test_data, batch_size=1)

with torch.no_grad():
    model.eval()
    for data in test_data:
        data = data.to(device)
        pred = model(data).detach().cpu().numpy()

        print(data, pred)
```

```
100%|██████████| 3/3 [00:00<00:00, 209.15it/s]DataBatch(x=[1, 1, 2], edge_index=[2, 0], y=[1], batch=[1], ptr=[2]) [0.38533816]
DataBatch(x=[3, 1, 2], edge_index=[2, 2], y=[3], batch=[3], ptr=[2]) [0.4903999 0.4756465 0.5093336]
DataBatch(x=[4, 1, 2], edge_index=[2, 3], y=[4], batch=[4], ptr=[2]) [0.5351139  0.32785496 0.44564673 0.71506095]

/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
```