ДОДАТОК А

Вихідний код розроблених методів

```
    class LightGCN(MessagePassing):

        def __init__(self):
            super(LightGCN, self).__init__(aggr='add
')

        def update(self, aggr_out, x):

            aggr_out = 0.7*aggr_out + 0.3*x

            return aggr_out

        def message(self, x_j, norm):

            return norm.view(-1, 1) * x_j

        def forward(self, x, edge_index):

            row, col = edge_index
            deg = degree(col)
            deg_inv_sqrt = deg.pow(-0.5)
            norm = deg_inv_sqrt[row] * deg_inv_sqrt[
col]


            return self.propagate(edge_index, x=x, n
orm=norm)
    class GNN(torch.nn.Module):

        def __init__(self, embedding_dim, num_nodes,
 num_playlists, num_layers):
            super(GNN, self).__init__()

            self.embedding_dim = embedding_dim
            self.num_nodes = num_nodes
            self.num_playlists = num_playlists
            self.num_layers = num_layers


            self.embeddings = torch.nn.Embedding(num
```

```
_embeddings=self.num_nodes, embedding_dim=self.embed
ding_dim)
            torch.nn.init.normal_(self.embeddings.we
ight, std=0.1)

            self.layers = torch.nn.ModuleList()
            for _ in range(self.num_layers):
                self.layers.append(LightGCN())

            self.sigmoid = torch.sigmoid

        def forward(self):
            raise NotImplementedError("forward() has
 not been implemented for the GNN class. Do not use"
)

        def gnn_propagation(self, edge_index_mp):

            x = self.embeddings.weight

            x_at_each_layer = [x]
            for i in range(self.num_layers):
                x = self.layers[i](x, edge_index_mp)
                x_at_each_layer.append(x)
            final_embs = torch.stack(x_at_each_layer
, dim=0).mean(dim=0)
            return final_embs

        def predict_scores(self, edge_index, embs):

            scores = embs[edge_index[0,:], :] * embs
[edge_index[1,:], :]
            scores = scores.sum(dim=1)
            scores = self.sigmoid(scores)
            return scores

        def calc_loss(self, data_mp, data_pos, data_
neg):


            final_embs = self.gnn_propagation(data_m
p.edge_index)


            pos_scores = self.predict_scores(data_po
```

```
s.edge_index, final_embs)
            neg_scores = self.predict_scores(data_ne
g.edge_index, final_embs)


            loss = -
torch.log(self.sigmoid(pos_scores -
 neg_scores)).mean()
            return loss

        def evaluation(self, data_mp, data_pos, k):

            final_embs = self.gnn_propagation(data_m
p.edge_index)


            unique_playlists = torch.unique_consecut
ive(data_pos.edge_index[0,:])
            playlist_emb = final_embs[unique_playlis
ts, :]


            song_emb = final_embs[self.num_playlists
:, :]


            ratings = self.sigmoid(torch.matmul(play
list_emb, song_emb.t()))



            result = recall_at_k(ratings.cpu(), k, s
elf.num_playlists, data_pos.edge_index.cpu(),
                                 unique_playlists.cp
u(), data_mp.edge_index.cpu())
            return result
    def recall_at_k(all_ratings, k, num_playlists, g
round_truth, unique_playlists, data_mp):

        known_edges = data_mp[:, data_mp[0,:] < num_p
laylists]


        playlist_to_idx_in_batch = {playlist: i for i
, playlist in enumerate(unique_playlists.tolist())}
```

```
        exclude_playlists, exclude_songs = [], []
        for i in range(known_edges.shape[1]):
            pl, song = known_edges[:,i].tolist()
            if pl in playlist_to_idx_in_batch:
                exclude_playlists.append(playlist_to_id
x_in_batch[pl])
                exclude_songs.append(song -
 num_playlists)
        all_ratings[exclude_playlists, exclude_songs]
 = -10000


        _, top_k = torch.topk(all_ratings, k=k, dim=1
)
        top_k += num_playlists


        ret = {}
        for i, playlist in enumerate(unique_playlists
):
            pos_songs = ground_truth[1, ground_truth[0
, :] == playlist]

            k_recs = top_k[i, :]
            recall = len(np.intersect1d(pos_songs, k_r
ecs)) / len(pos_songs)
            ret[playlist] = recall
        return ret
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	_75_ с.
	Інші документи	
2.	Презентаційні матеріали	_14_ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Количева П.А.			Дослідження методів побудови рекомендаційних систем на основі графових нейронних мереж	Шифр групи	Код напр./спец.
Перевірив		Волощук О.Б.				СШІм-20-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					