In [1]:

```
from __future__ import division, print_function, absolute_import, unicode_literals
```

In [2]:

```
# packages initialization

import torch
from torch.utils.data import TensorDataset, DataLoader
from torch import optim
import torch.nn as nn
import torch.nn.functional as F

from scipy import sparse
import numpy as np

import math
import heapq
import json
import os
```

Configuration

In [3]:

```
# training device
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
DEVICE
```

Out[3]:

```
device(type='cuda', index=0)
```

In [4]:

```
class Config(object):
    def __init__(self, embedding_size=6, epoch=30,
                 num_negatives=4, batch_size=512, lr=0.00005, drop_ratio=0.2, top_k=5):

        self.data_dir = './data/CAMRa2011/'
        self.embedding_size = embedding_size
        self.epoch = epoch
        self.num_negatives = num_negatives
        self.batch_size = batch_size
        self.lr = lr
        self.drop_ratio = drop_ratio
        self.top_k = top_k

    def export_json(self, file_path):

        config_dict = {
            'embedding_size': self.embedding_size,
            'epoch': self.epoch,
            'num_negatives': self.num_negatives,
            'batch_size': self.batch_size,
            'lr': self.lr,
            'drop_ratio': self.drop_ratio,
            'top_k': self.top_k
        }

        with open(file_path, 'w') as file:
            file.write(json.dumps(config_dict))
```

DataLoader

how to process dataset:

1. combine user train set with user test set, group train set with group test set
2. compute average user's rating count & average group's rating count
3. filter out examples which its users/groups has total rating count lower than 30% of avg
4. generate negative instances: pick items that haven't been interacted by users/groups
5. split into trainset & testset

or:

1. generate negative instances using items that haven't been interacted by users/groups
2. recalculate user number and group number and item number from both train & test set

In [5]:

```
class CAMRa2011Dataset(object):
    """CAMRa2011 dataset"""

    def __init__(self, dataset_dir):

        self.pathes = {
            'train': {
                'user': dataset_dir + "userRatingTrain.txt",
                'group': dataset_dir + "groupRatingTrain.txt"
            },
            'test': {
                'user': dataset_dir + "userRatingTest.txt",
                'user_negative': dataset_dir + "userRatingNegative.txt",
                'group': dataset_dir + "groupRatingTest.txt",
                'group_negative': dataset_dir + "groupRatingNegative.txt",
            },
            'group_user': dataset_dir + "groupMember.txt"
        }

        # number of items & users (needed by embedding layer)
        self.max_iid = 0

        # get the mapping of users and groups
        # format: {gid: [uid, uid, ..], gid: [uid, uid, ..], ...}
        self.group_members = self.get_group_user_mapping()

        # get interaction matrix from uid-iid training set
        # train_user_matrix[uid, iid] = [1 | 0]
        self.train_user_matrix = self.get_interaction_matrix(self.pathes['train']['use
r'])

        # format: [[uid, iid], [uid, iid], ...]
        # only pairs of users & items have interactions would appear in the list
        self.test_user_list = self.get_interaction_list(self.pathes['test']['user'])

        # format: [[uid, ...], [uid, ...], ...]
        # test_user_negative_list & test_user_list follow the same order
        # e.g. test_user_negative_list[0] is for test_user_list[0]
        self.test_user_negative_list = self.get_negatives(self.pathes['test']['user_neg
ative'])

        # get interaction matrix from gid-iid training set
        self.train_group_matrix = self.get_interaction_matrix(self.pathes['train']['gro
up'])

        # pairs of group & item to be tested
        self.test_group_list = self.get_interaction_list(self.pathes['test']['group'])

        self.test_group_negative_list = self.get_negatives(self.pathes['test']['group_n
egative'])


    def get_user_dataloader(self, batch_size=256, shuffle=True, num_negatives=6):

        users, positives_negatives = self.get_train_instances(self.train_user_matrix, n
um_negatives=num_negatives)
        dataset = TensorDataset(
            torch.tensor(users, dtype=torch.float),
            torch.tensor(positives_negatives, dtype=torch.float))
```

```
        loader = DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)

        return loader

    def get_group_dataloader(self, batch_size=256, shuffle=True, num_negatives=6):

        groups, positives_negatives = self.get_train_instances(self.train_group_matrix,
num_negatives=num_negatives)

        dataset = TensorDataset(
            torch.tensor(groups, dtype=torch.float),
            torch.tensor(positives_negatives, dtype=torch.float))

        loader = DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)

        return loader

    # get number of groups, users and items
    def get_sizes(self):
        group_size = len(self.group_members)

        num_user, _ = self.train_user_matrix.shape

        return (group_size, num_user, self.max_iid+1)

    def get_train_instances(self, interaction_matrix, num_negatives=6):

        users, positive_items, negative_items = [], [], []

        for (uid, iid) in interaction_matrix.keys():

            # positive instance
            for _ in range(num_negatives):

                # positive instances
                positive_items.append(iid)

                # negative instances ---> need to be fixed
                negative_iid = np.random.randint(self.max_iid+1)
                while (uid, negative_iid) in interaction_matrix:
                    negative_iid = np.random.randint(self.max_iid+1) # re-generate an n
egative iid
                negative_items.append(negative_iid)

                # users
                users.append(uid)

        positives_negatives = [[positive_iid, negative_iid] for positive_iid, negative_
iid in zip(positive_items, negative_items)]

        return users, positives_negatives


    def get_group_user_mapping(self):

        mapping = {}

        # read mapping file
        with open(self.pathes['group_user'], 'r') as file:
```

```
            line = file.readline().strip()
            while line != None and line != "":

                # sample line format: [gid] [uid 1],[uid 2],[uid 3],[uid 4]
                sequences = line.split(' ')
                gid = int(sequences[0])
                mapping[gid] = []
                for uid in sequences[1].split(','):
                    mapping[gid].append(int(uid))
                line = file.readline().strip()

        return mapping

    # parse all interactions in dataset to 2D sparse matrix
    def get_interaction_matrix(self, rating_file_path):

        # get number of users and items
        num_users, num_items = 0, 0
        with open(rating_file_path, "r") as file:

            line = file.readline()
            while line != None and line != "":
                arr = line.split(" ")
                uid, iid = int(arr[0]), int(arr[1])

                # update num_items if bigger iid appears
                num_items = max(num_items, iid)
                # update num_items if bigger iid appears
                num_users = max(num_users, uid)

                line = file.readline()

        # update the max iid / uid of the whole dataset
        self.max_iid = max(self.max_iid, num_items)

        # construct interaction matrix
        # dok_matrix: Dictionary Of Keys based sparse matrix, an efficient structure fo
r constructing sparse matrices incrementally.
        matrix = sparse.dok_matrix((num_users + 1, num_items + 1), dtype=np.float32) #
 iid and uid starts from 1
        with open(rating_file_path, "r") as file:
            line = file.readline()
            while line != None and line != "":
                arr = line.split(" ")
                if len(arr) > 2:
                    uid, iid, rating = int(arr[0]), int(arr[1]), int(arr[2])
                    if (rating > 0):
                        matrix[uid, iid] = 1.0
                else:
                    uid, iid = int(arr[0]), int(arr[1])
                    matrix[uid, iid] = 1.0
                line = file.readline()

        return matrix

    # parse all interactions in dataset to list
    def get_interaction_list(self, rating_file_path):

        interaction_list = []
        with open(rating_file_path, "r") as file:
            line = file.readline()
```

```
            while line != None and line != "":
                arr = line.split(" ")
                uid, iid = int(arr[0]), int(arr[1])
                interaction_list.append([uid, iid])

                # update max iid/uid if bigger iid/uid appears
                self.max_iid = max(self.max_iid, iid)

                line = file.readline()

        return interaction_list

    # parse negative sample lists for pairs in test set
    # negative samples: the items which never been interacted
    # the order of returned sample lists must be paired with test list
    def get_negatives(self, file_path):

        negative_samples_list = []

        with open(file_path, "r") as file:

            line = file.readline()
            while line != None and line != "":
                arr = line.split(" ")

                negative_iids = []
                for iid in arr[1:]:
                    negative_iids.append(int(iid))

                negative_samples_list.append(negative_iids)
                line = file.readline()

        return negative_samples_list
```

AGREE Model (Attentive Group Representation)

In [6]:

```
# embedding networks

class UserEmbeddingLayer(nn.Module):
    def __init__(self, num_users, embedding_dim):
        super(UserEmbeddingLayer, self).__init__()
        self.userEmbedding = nn.Embedding(num_users, embedding_dim)

    def forward(self, uids):
        user_embedded = self.userEmbedding(uids)
        return user_embedded

class ItemEmbeddingLayer(nn.Module):
    def __init__(self, num_items, embedding_dim):
        super(ItemEmbeddingLayer, self).__init__()
        self.itemEmbedding = nn.Embedding(num_items, embedding_dim)

    def forward(self, iids):
        item_embedded = self.itemEmbedding(iids)
        return item_embedded

class GroupEmbeddingLayer(nn.Module):
    def __init__(self, num_groups, embedding_dim):
        super(GroupEmbeddingLayer, self).__init__()
        self.groupEmbedding = nn.Embedding(num_groups, embedding_dim)

    def forward(self, gids):
        group_embedded = self.groupEmbedding(gids)
        return group_embedded
```

In [7]:

```
# attention network

class AttentionLayer(nn.Module):
    def __init__(self, embedding_dim, drop_ratio=0):
        super(AttentionLayer, self).__init__()

        self.linear1 = nn.Linear(embedding_dim, 16)
        self.linear2 = nn.Linear(16, 1)
        self.dropout = nn.Dropout(p=drop_ratio)


    def forward(self, x):

        x = F.relu(self.linear1(x))
        x = self.dropout(x)
        x = self.linear2(x)
        weights = F.softmax(x.view(1, -1), dim=1)

        return weights
```

In [8]:

```
# final layers of AGREE for prediction

class PredictLayer(nn.Module):
    def __init__(self, embedding_dim, drop_ratio=0):
        super(PredictLayer, self).__init__()

        self.linear1 = nn.Linear(embedding_dim, 8)
        self.dropout = nn.Dropout(p=drop_ratio)
        self.linear2 = nn.Linear(8, 1)

    def forward(self, x):

        x = F.relu(self.linear1(x))
        x = self.dropout(x)
        out = self.linear2(x)

        return out
```

In [9]:

```
# AGREE model

class AGREE(nn.Module):
    def __init__(self, num_users, num_items, num_groups, embedding_dim, group_member_ma
pping, drop_ratio):
        super(AGREE, self).__init__()

        self.user_embedding = UserEmbeddingLayer(num_users, embedding_dim)
        self.item_embedding = ItemEmbeddingLayer(num_items, embedding_dim)
        self.group_embedding = GroupEmbeddingLayer(num_groups, embedding_dim)
        self.attention = AttentionLayer(2 * embedding_dim, drop_ratio)
        self.predict = PredictLayer(3 * embedding_dim, drop_ratio)

        self.group_members = group_member_mapping

        self.num_users = num_users
        self.num_groups = num_groups
        self.num_items = num_items

        # initial model's parameters
        for m in self.modules():

            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=1) # normal distribution
            if isinstance(m, nn.Embedding):
                nn.init.xavier_normal_(m.weight) # Glorot initialization

    def forward(self, gids, uids, iids):

        # group prediction
        if (gids is not None) and (uids is None):
            output = self.group_forward(gids, iids)
        # user prediction
        else:
            output = self.user_forward(uids, iids)

        return output

    # group forwarding
    def group_forward(self, gids, iids):

        # group_embeds = Variable(torch.Tensor())
        group_embeddeds = torch.empty(0).to(DEVICE)

        # generate embedding vector for item
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # shape: (batch_
size, 1, embedding_dim)


        # get attentive group embedding
        for gid, iid in zip(gids, iids):
            member_uids = self.group_members[gid.item()]

            # generate user embedding vector
            member_embeddeds = self.user_embedding(
                torch.tensor(member_uids, dtype=torch.long).unsqueeze(dim=1).to(DEVICE
)) # shape: (num_member, 1, embedding_dim)
```

```
            # generate item embedding vector
            one_items = [iid.item() for _ in member_uids]
            one_item_embeddeds = self.item_embedding(
                torch.tensor(one_items, dtype=torch.long).unsqueeze(dim=1).to(DEVICE))
# shape: (num_member, 1, embedding_dim)

            # get attentive weights for each user-item pair
            user_item_embeddeds = torch.cat((member_embeddeds, one_item_embeddeds), dim
=2) # shape: (num_member, 1, 2 * embedding_dim)
            user_item_embeddeds = user_item_embeddeds.squeeze(dim=1) # shape: (num_memb
er, 2 * embedding_dim)
            attentive_weights = self.attention(user_item_embeddeds) # shape: (num_membe
r, 1)

            # aggregation
            member_embeddeds = member_embeddeds.squeeze(dim=1) # shape: (num_member, em
bedding_dim)
            aggregated_user_item_embeddeds = torch.matmul(attentive_weights, member_emb
eddeds) # shape: (1, embedding_dim)

            # generate group-item embedding vector
            group_embedded = self.group_embedding(torch.tensor([[gid.item()]], dtype=to
rch.long).to(DEVICE)) # shape: (1, embedding_dim)

            # group embedding = user embedding aggregation + group preference embedding
            aggregated_all_embedded = aggregated_user_item_embeddeds + group_embedded #
shape: (1, embedding_dim)

            aggregated_all_embedded = aggregated_all_embedded.squeeze(dim=0) # shape:
 (embedding_dim, )

            # append group embedding
            group_embeddeds = torch.cat((group_embeddeds, aggregated_all_embedded), dim
=0) # shape: (batch_size, embedding_dim)

        item_embeddeds = item_embeddeds.squeeze(dim=1) # shape: (batch_size, embedding_
dim)

        # element-wise product: group-item interaction
        interacted_embeddeds = torch.mul(group_embeddeds, item_embeddeds) # shape: (bat
ch_size, embedding_dim)

        # pooling: [group x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, group_embeddeds, item_embed
deds), dim=1) # shape: (batch_size, 3*embedding_dim)

        y = torch.sigmoid(self.predict(pooled_embeddeds))
        return y

    # user forwarding
    def user_forward(self, uids, iids):

        # uids.shape: (batch_size, )
        # iids.shape: (batch_size, )

        # generate user embedding vectors
        user_embeddeds = self.user_embedding(
            uids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # generate item embedding vectors
```

```
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # element-wise product: user-item interactions
        interacted_embeddeds = torch.mul(user_embeddeds, item_embeddeds) # (batch_size,
1, embedding_dim)

        # pooling: [user x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, user_embeddeds, item_embedd
eds), dim=2) # (batch_size, 1, 3 * embedding_dim)

        # reshape x from (batch_size, 1, 3 * embedding_dim) to (batch_size, 3 * embeddi
ng_dim)
        pooled_embeddeds = pooled_embeddeds.squeeze(dim=1)

        y = torch.sigmoid(self.predict(pooled_embeddeds))

        return y
```

Baseline Models

In [23]:

```
# GREE model: using average strategy to aggregate group members' preference

class GREE(nn.Module):
    def __init__(self, num_users, num_items, num_groups, embedding_dim, group_member_ma
pping, drop_ratio):
        super(GREE, self).__init__()

        self.user_embedding = UserEmbeddingLayer(num_users, embedding_dim)
        self.item_embedding = ItemEmbeddingLayer(num_items, embedding_dim)
        self.group_embedding = GroupEmbeddingLayer(num_groups, embedding_dim)
        self.predict = PredictLayer(3 * embedding_dim, drop_ratio)

        self.group_members = group_member_mapping

        self.num_users = num_users
        self.num_groups = num_groups
        self.num_items = num_items

        # initial model's parameters
        for m in self.modules():

            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=1) # normal distribution
            if isinstance(m, nn.Embedding):
                nn.init.xavier_normal_(m.weight) # Glorot initialization

    def forward(self, gids, uids, iids):

        # group prediction
        if (gids is not None) and (uids is None):
            output = self.group_forward(gids, iids)
        # user prediction
        else:
            output = self.user_forward(uids, iids)

        return output

    # group forwarding
    def group_forward(self, gids, iids):

        # group_embeds = Variable(torch.Tensor())
        group_embeddeds = torch.empty(0).to(DEVICE)

        # generate embedding vector for item
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # shape: (batch_
size, 1, embedding_dim)


        # get attentive group embedding
        for gid, iid in zip(gids, iids):
            member_uids = self.group_members[gid.item()]
            num_member = len(member_uids)

            # generate user embedding vector
            member_embeddeds = self.user_embedding(
                torch.tensor(member_uids, dtype=torch.long).unsqueeze(dim=1).to(DEVICE
)) # shape: (num_member, 1, embedding_dim)
```

```
            # generate item embedding vector
            one_items = [iid.item() for _ in member_uids]
            one_item_embeddeds = self.item_embedding(
                torch.tensor(one_items, dtype=torch.long).unsqueeze(dim=1).to(DEVICE))
# shape: (num_member, 1, embedding_dim)

            # get attentive weights for each user-item pair
            user_item_embeddeds = torch.cat((member_embeddeds, one_item_embeddeds), dim
=2) # shape: (num_member, 1, 2 * embedding_dim)
            user_item_embeddeds = user_item_embeddeds.squeeze(dim=1) # shape: (num_memb
er, 2 * embedding_dim)

            # uniform aggregation weights
            attentive_weights = torch.full(
                (1, num_member), (1/num_member), dtype=torch.float).to(DEVICE) # shape:
(1, num_member)

            # aggregation
            member_embeddeds = member_embeddeds.squeeze(dim=1) # shape: (num_member, em
bedding_dim)
            aggregated_user_item_embeddeds = torch.matmul(attentive_weights, member_emb
eddeds) # shape: (1, embedding_dim)

            # generate group-item embedding vector
            group_embedded = self.group_embedding(torch.tensor([[gid.item()]], dtype=to
rch.long).to(DEVICE)) # shape: (1, embedding_dim)

            # group embedding = user embedding aggregation + group preference embedding
            aggregated_all_embedded = aggregated_user_item_embeddeds + group_embedded #
shape: (1, embedding_dim)

            aggregated_all_embedded = aggregated_all_embedded.squeeze(dim=0) # shape:
 (embedding_dim, )

            # append group embedding
            group_embeddeds = torch.cat((group_embeddeds, aggregated_all_embedded), dim
=0) # shape: (batch_size, embedding_dim)

        item_embeddeds = item_embeddeds.squeeze(dim=1) # shape: (batch_size, embedding_
dim)

        # element-wise product: group-item interaction
        interacted_embeddeds = torch.mul(group_embeddeds, item_embeddeds) # shape: (bat
ch_size, embedding_dim)

        # pooling: [group x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, group_embeddeds, item_embed
deds), dim=1) # shape: (batch_size, 3*embedding_dim)

        y = torch.sigmoid(self.predict(pooled_embeddeds))
        return y

    # user forwarding
    def user_forward(self, uids, iids):

        # uids.shape: (batch_size, )
        # iids.shape: (batch_size, )

        # generate user embedding vectors
        user_embeddeds = self.user_embedding(
            uids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
```

```
 1, embedding_dim)

        # generate item embedding vectors
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # element-wise product: user-item interactions
        interacted_embeddeds = torch.mul(user_embeddeds, item_embeddeds) # (batch_size,
1, embedding_dim)

        # pooling: [user x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, user_embeddeds, item_embedd
eds), dim=2) # (batch_size, 1, 3 * embedding_dim)

        # reshape x from (batch_size, 1, 3 * embedding_dim) to (batch_size, 3 * embeddi
ng_dim)
        pooled_embeddeds = pooled_embeddeds.squeeze(dim=1)

        y = torch.sigmoid(self.predict(pooled_embeddeds))

        return y
```

In [28]:

```
# without any member aggregation, treat group as indivudual

class GNCF(nn.Module):
    def __init__(self, num_users, num_items, num_groups, embedding_dim, group_member_ma
pping, drop_ratio):
        super(GNCF, self).__init__()

        self.user_embedding = UserEmbeddingLayer(num_users, embedding_dim)
        self.item_embedding = ItemEmbeddingLayer(num_items, embedding_dim)
        self.group_embedding = GroupEmbeddingLayer(num_groups, embedding_dim)
        self.predict = PredictLayer(3 * embedding_dim, drop_ratio)

        self.group_members = group_member_mapping

        self.num_users = num_users
        self.num_groups = num_groups
        self.num_items = num_items

        # initial model's parameters
        for m in self.modules():

            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=1) # normal distribution
            if isinstance(m, nn.Embedding):
                nn.init.xavier_normal_(m.weight) # Glorot initialization

    def forward(self, gids, uids, iids):

        # group prediction
        if (gids is not None) and (uids is None):
            output = self.group_forward(gids, iids)
        # user prediction
        else:
            output = self.user_forward(uids, iids)

        return output

    # group forwarding
    def group_forward(self, gids, iids):

        # gids.shape: (batch_size, )
        # iids.shape: (batch_size, )

        # generate user embedding vectors
        group_embeddeds = self.group_embedding(
            gids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # generate item embedding vectors
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # element-wise product: user-item interactions
        interacted_embeddeds = torch.mul(group_embeddeds, item_embeddeds) # (batch_siz
e, 1, embedding_dim)

        # pooling: [user x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, group_embeddeds, item_embed
```

```
deds), dim=2) # (batch_size, 1, 3 * embedding_dim)

        # reshape x from (batch_size, 1, 3 * embedding_dim) to (batch_size, 3 * embeddi
ng_dim)
        pooled_embeddeds = pooled_embeddeds.squeeze(dim=1)

        y = torch.sigmoid(self.predict(pooled_embeddeds))

        return y

    # user forwarding
    def user_forward(self, uids, iids):

        # uids.shape: (batch_size, )
        # iids.shape: (batch_size, )

        # generate user embedding vectors
        user_embeddeds = self.user_embedding(
            uids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # generate item embedding vectors
        item_embeddeds = self.item_embedding(
            iids.clone().type(torch.long).unsqueeze(dim=1).to(DEVICE)) # (batch_size,
 1, embedding_dim)

        # element-wise product: user-item interactions
        interacted_embeddeds = torch.mul(user_embeddeds, item_embeddeds) # (batch_size,
1, embedding_dim)

        # pooling: [user x item, group, item]
        pooled_embeddeds = torch.cat((interacted_embeddeds, user_embeddeds, item_embedd
eds), dim=2) # (batch_size, 1, 3 * embedding_dim)

        # reshape x from (batch_size, 1, 3 * embedding_dim) to (batch_size, 3 * embeddi
ng_dim)
        pooled_embeddeds = pooled_embeddeds.squeeze(dim=1)

        y = torch.sigmoid(self.predict(pooled_embeddeds))

        return y
```

Evaluation

In [30]:

```
def evaluate_model(model, uid_piid_list, niids_list, K, input_type):
    """
    Evaluate the performance (Hit_Ratio, NDCG) of top-K recommendation
    Return: score of each test rating.
    """

    # uid_piid_list shape: (testset size, 2)
    # niids_list shape: (testset size, 100)

    testset_size = len(uid_piid_list)

    hits, ndcgs = [], []

    # for printing progress
    max_percentage = -1
    print("Evaluating: ", end="")
    for idx in range(testset_size):

        # print progress
        if idx%10 == 0:
            percentage = idx*100/testset_size
            if int(percentage/10) > max_percentage:
                max_percentage = int(percentage/10)
                print("{:.1f}%".format(percentage), end=" ")

        (hr, ndcg) = evaluate_one_example(model, uid_piid_list, niids_list, K, input_ty
pe, idx)

        hits.append(hr)
        ndcgs.append(ndcg)

    print("") # line-break

    return (hits, ndcgs)


def evaluate_one_example(model, uid_piid_list, niids_list, K, input_type, idx):

    uid, piid = uid_piid_list[idx]

    # items to be predicted
    iids = torch.tensor(niids_list[idx] + [piid], dtype=torch.long).to(DEVICE)
    uids = torch.full(iids.shape, uid, dtype=torch.long).to(DEVICE)

    # store prediction scores
    iid_scores = {}

    if input_type == 'group':
        predictions = model(uids, None, iids)
    elif input_type == 'user':
        predictions = model(None, uids, iids)

    for idx in range(len(iids)):
        iid = iids[idx]
        iid_scores[iid] = torch.flatten(predictions)[idx]

    # Evaluate top rank list
    top_k_iids = heapq.nlargest(K, iid_scores, key=iid_scores.get)
```

```
    hr = evaluate_HR(top_k_iids, piid)
    ndcg = evaluate_NDCG(top_k_iids, piid)

    return (hr, ndcg)

def evaluate_HR(top_k_iids, piid):

    for iid in top_k_iids:
        if iid == piid:
            return 1

    return 0

def evaluate_NDCG(top_k_iids, piid):

    for idx in range(len(top_k_iids)):
        iid = top_k_iids[idx]

        if iid == piid:
            return math.log(2) / math.log(idx+2)

    return 0
```

In [11]:

```
def evaluation(model, uid_piid_list, niids_list, K, input_type):

    model.eval()

    (hrs, ndcgs) = evaluate_model(model, uid_piid_list, niids_list, K, input_type)
    avg_hr, avg_ndcg = np.mean(hrs), np.mean(ndcgs)

    return avg_hr, avg_ndcg
```

Training

In [12]:

```
# model training procedure
def training(model, dataloader, epoch_id, config, input_type):

    # user trainning
    learning_rate = config.lr

    # lr decay: halve for every five epochs
    for _ in range(0, epoch_id, 5):
        learning_rate /= 2

    # create optimizer
    optimizer = optim.RMSprop(model.parameters(), lr=learning_rate)

    print("Epoch {}, lr = {}, input_type = {}".format(epoch_id, learning_rate, input_ty
pe))

    losses = []
    for batch_id, (uids, piids_niids) in enumerate(dataloader):

        if batch_id%10 == 0:
            print(".", end="")

        # Data Load
        p_iids = piids_niids[:, 0]
        n_iids = piids_niids[:, 1]

        # Forward
        if input_type == 'user':
            positive_predictions = model(None, uids, p_iids)
            negative_predictions = model(None, uids, n_iids)
        elif input_type == 'group':
            positive_predictions = model(uids, None, p_iids)
            negative_predictions = model(uids, None, n_iids)

        optimizer.zero_grad()

        # calculate loss
        loss = torch.mean((positive_predictions - negative_predictions -1) **2)

        # back propagation
        loss.backward()
        optimizer.step()

        # record loss history
        losses.append(loss.item())

    print("") # line-break

    return np.mean(losses)
```

Training Procedure

In [27]:

```
def start_training(model_name='AGREE', name="my_training", embedding_dim=6, train_epoch
=30,
        num_negatives=4, batch_size=512, lr=0.00005, drop_ratio=0.2, top_k=5):

    # create dir to store configs, models and training history
    training_directory = 'models/'+ name
    os.mkdir(training_directory)

    # create configuration object
    config = Config(embedding_size=embedding_dim, epoch=train_epoch,
        num_negatives=num_negatives, batch_size=batch_size, lr=lr, drop_ratio=drop_rati
o, top_k=top_k)

    # save configs to file
    config.export_json(training_directory + "/config.json")

    # load and parse dataset
    dataset = CAMRa2011Dataset(config.data_dir)
    num_group, num_user, num_item = dataset.get_sizes()
    print("number of groups: {}".format(num_group))
    print("number of users: {}".format(num_user))
    print("number of items: {}".format(num_item))

    # create model object
    if model_name == 'AGREE':
        model = AGREE(num_user, num_item, num_group, config.embedding_size,
                      dataset.group_members, config.drop_ratio).to(DEVICE)
        print("AGREE at embedding size: {}, epoch: {}, NDCG & HR: Top-{}"
              .format(config.embedding_size, config.epoch, config.top_k))
    elif model_name == 'GREE':
        model = GREE(num_user, num_item, num_group, config.embedding_size,
                      dataset.group_members, config.drop_ratio).to(DEVICE)
        print("GREE at embedding size: {}, epoch: {}, NDCG & HR: Top-{}"
              .format(config.embedding_size, config.epoch, config.top_k))

    elif model_name == 'GNCF':
        model = GNCF(num_user, num_item, num_group, config.embedding_size,
                      dataset.group_members, config.drop_ratio).to(DEVICE)
        print("GNCF at embedding size: {}, epoch: {}, NDCG & HR: Top-{}"
              .format(config.embedding_size, config.epoch, config.top_k))

    # start training
    all_history = []
    for epoch in range(config.epoch):

        history = { 'loss': {}, 'hr': {}, 'ndcg': {} }

        # set the model in train mode (some network like dropout will behave differentl
y in train mode / evaluaiton mode)
        model.train(mode=True)

        # train the model using user interactions
        user_loss = training(model,
                     dataset.get_user_dataloader(batch_size=config.batch_size, shuffle=
True, num_negatives=config.num_negatives),
                     epoch, config, 'user')
        history['loss']['user'] = user_loss

        # train the model using group & members interactions
```

```
        group_loss = training(model,
                              dataset.get_group_dataloader(batch_size=config.batch_size, shuf
fle=True, num_negatives=config.num_negatives),
                              epoch, config, 'group')
        history['loss']['group'] = group_loss

        print("Losses: {}".format(history['loss']))

        # evaluation
        avg_user_hr, avg_user_ndcg = evaluation(model, dataset.test_user_list, dataset.
test_user_negative_list, config.top_k, 'user')
        print("User-- average Top-{} Hit Rate: {:.4f}, average Top-{} NDCG: {:.4f}"
              .format(config.top_k, avg_user_hr, config.top_k, avg_user_ndcg))
        history['hr']['user'] = avg_user_hr
        history['ndcg']['user'] = avg_user_ndcg

        avg_group_hr, avg_group_ndcg = evaluation(model, dataset.test_group_list, datas
et.test_group_negative_list, config.top_k, 'group')
        print("Group-- average Top-{} Hit Rate: {:.4f}, average Top-{} NDCG: {:.4f}"
              .format(config.top_k, avg_group_hr, config.top_k, avg_group_ndcg))
        history['hr']['group'] = avg_group_hr
        history['ndcg']['group'] = avg_group_ndcg

        all_history.append(history)

        # save model to file
        torch.save(model.state_dict(), training_directory+ "/{}_e{}.pt".format(model_na
me, epoch))

        # save training & evaluation result to file
        with open(training_directory+"/history.json", 'w') as file:
                file.write(json.dumps(all_history))
```

Experiments

In [33]:

```
start_training(model_name="AGREE", name="AGREE-basic", embedding_dim=6, train_epoch=20,
        num_negatives=4, batch_size=1024, lr=0.00001, drop_ratio=0.1, top_k=5)
```

```
number of groups: 290
number of users: 602
number of items: 7710
AGREE at embedding size: 6, epoch: 20, NDCG & HR: Top-5
Epoch 0, lr = 1e-05, input_type = user
........................................
Epoch 0, lr = 1e-05, input_type = group
........................................
Losses: {'user': 0.995277810833274, 'group': 0.9872857651202317}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.1837, average Top-5 NDCG: 0.1187
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.1745, average Top-5 NDCG: 0.1131
Epoch 1, lr = 5e-06, input_type = user
........................................
Epoch 1, lr = 5e-06, input_type = group
........................................
Losses: {'user': 0.9788238283425774, 'group': 0.974697018696878}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.2801, average Top-5 NDCG: 0.1855
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2683, average Top-5 NDCG: 0.1739
Epoch 2, lr = 5e-06, input_type = user
........................................
Epoch 2, lr = 5e-06, input_type = group
........................................
Losses: {'user': 0.9673250745854062, 'group': 0.9623411878977233}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3668, average Top-5 NDCG: 0.2437
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3400, average Top-5 NDCG: 0.2269
Epoch 3, lr = 5e-06, input_type = user
........................................
Epoch 3, lr = 5e-06, input_type = group
........................................
Losses: {'user': 0.9540670881009484, 'group': 0.9489722084296264}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4385, average Top-5 NDCG: 0.2888
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3986, average Top-5 NDCG: 0.2647
Epoch 4, lr = 5e-06, input_type = user
........................................
Epoch 4, lr = 5e-06, input_type = group
........................................
Losses: {'user': 0.9391727210182347, 'group': 0.9339175225655778}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4937, average Top-5 NDCG: 0.3200
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4524, average Top-5 NDCG: 0.2974
Epoch 5, lr = 5e-06, input_type = user
........................................
Epoch 5, lr = 5e-06, input_type = group
........................................
Losses: {'user': 0.9229159982580888, 'group': 0.9174510856874946}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5296, average Top-5 NDCG: 0.3429
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4903, average Top-5 NDCG: 0.3167
Epoch 6, lr = 2.5e-06, input_type = user
........................................
Epoch 6, lr = 2.5e-06, input_type = group
```

```
.........................................
Losses: {'user': 0.9078296286563305, 'group': 0.9066189265305222}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5369, average Top-5 NDCG: 0.3488
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5028, average Top-5 NDCG: 0.3235
Epoch 7, lr = 2.5e-06, input_type = user
.........................................
Epoch 7, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.8987630789830974, 'group': 0.8979863447396934}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5439, average Top-5 NDCG: 0.3536
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5166, average Top-5 NDCG: 0.3298
Epoch 8, lr = 2.5e-06, input_type = user
.........................................
Epoch 8, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.8891153334207338, 'group': 0.8886372583523359}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5472, average Top-5 NDCG: 0.3570
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5241, average Top-5 NDCG: 0.3354
Epoch 9, lr = 2.5e-06, input_type = user
.........................................
Epoch 9, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.8793117732696184, 'group': 0.8792609033130464}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5528, average Top-5 NDCG: 0.3612
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5269, average Top-5 NDCG: 0.3399
Epoch 10, lr = 2.5e-06, input_type = user
.........................................
Epoch 10, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.8694367794750484, 'group': 0.8693506193269137}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5581, average Top-5 NDCG: 0.3652
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5331, average Top-5 NDCG: 0.3446
Epoch 11, lr = 1.25e-06, input_type = user
.........................................
Epoch 11, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8603582949605632, 'group': 0.8631893300415437}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5625, average Top-5 NDCG: 0.3676
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5372, average Top-5 NDCG: 0.3466
Epoch 12, lr = 1.25e-06, input_type = user
.........................................
Epoch 12, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8554055483444877, 'group': 0.858338885972289}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5671, average Top-5 NDCG: 0.3708
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5372, average Top-5 NDCG: 0.3469
Epoch 13, lr = 1.25e-06, input_type = user
```

```
.........................................
Epoch 13, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8500934556910866, 'group': 0.853113609511836}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5698, average Top-5 NDCG: 0.3728
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5393, average Top-5 NDCG: 0.3483
Epoch 14, lr = 1.25e-06, input_type = user
.........................................
Epoch 14, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8445355395703349, 'group': 0.8480155508534438}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5721, average Top-5 NDCG: 0.3742
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5434, average Top-5 NDCG: 0.3516
Epoch 15, lr = 1.25e-06, input_type = user
.........................................
Epoch 15, lr = 1.25e-06, input_type = group
.....
```

```
----------------------------------------------------------------------
-
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-33-6027a4be6cdc> in <module>
      1 start_training(model_name="AGREE", name="AGREE-basic", embedding_d
im=6, train_epoch=20,
----> 2         num_negatives=4, batch_size=1024, lr=0.00001, drop_ratio=
0.1, top_k=5)

<ipython-input-27-a8ec1f5685b4> in start_training(model_name, name, embedd
ing_dim, train_epoch, num_negatives, batch_size, lr, drop_ratio, top_k)
     56         group_loss = training(model,
     57                               dataset.get_group_dataloader(batch_size=co
nfig.batch_size, shuffle=True, num_negatives=config.num_negatives),
---> 58                               epoch, config, 'group')
     59         history['loss']['group'] = group_loss
     60

<ipython-input-12-dd2b9b754a82> in training(model, dataloader, epoch_id, c
onfig, input_type)
     30         elif input_type == 'group':
     31             positive_predictions = model(uids, None, p_iids)
---> 32             negative_predictions = model(uids, None, n_iids)
     33
     34         optimizer.zero_grad()

E:\ProgramData\Anaconda3\envs\attentive\lib\site-packages\torch\nn\modules
\module.py in __call__(self, *input, **kwargs)
    548             result = self._slow_forward(*input, **kwargs)
    549         else:
--> 550             result = self.forward(*input, **kwargs)
    551         for hook in self._forward_hooks.values():
    552             hook_result = hook(self, input, result)

<ipython-input-9-c0b068c83d00> in forward(self, gids, uids, iids)
     27         # group prediction
     28         if (gids is not None) and (uids is None):
---> 29             output = self.group_forward(gids, iids)
     30         # user prediction
     31         else:

<ipython-input-9-c0b068c83d00> in group_forward(self, gids, iids)
     61             user_item_embeddeds = torch.cat((member_embeddeds, one
_item_embeddeds), dim=2) # shape: (num_member, 1, 2 * embedding_dim)
     62             user_item_embeddeds = user_item_embeddeds.squeeze(dim=
1) # shape: (num_member, 2 * embedding_dim)
---> 63             attentive_weights = self.attention(user_item_embeddeds
) # shape: (num_member, 1)
     64
     65             # aggregation

E:\ProgramData\Anaconda3\envs\attentive\lib\site-packages\torch\nn\modules
\module.py in __call__(self, *input, **kwargs)
    548             result = self._slow_forward(*input, **kwargs)
    549         else:
--> 550             result = self.forward(*input, **kwargs)
    551         for hook in self._forward_hooks.values():
    552             hook_result = hook(self, input, result)

<ipython-input-7-154cbda41dd5> in forward(self, x)
```

```
     19
     20         x = F.relu(self.linear1(x))
---> 21         x = self.dropout(x)
     22         x = self.linear2(x)
     23         weights = F.softmax(x.view(1, -1), dim=1)

E:\ProgramData\Anaconda3\envs\attentive\lib\site-packages\torch\nn\modules
\module.py in __call__(self, *input, **kwargs)
    548             result = self._slow_forward(*input, **kwargs)
    549         else:
--> 550             result = self.forward(*input, **kwargs)
    551         for hook in self._forward_hooks.values():
    552             hook_result = hook(self, input, result)

E:\ProgramData\Anaconda3\envs\attentive\lib\site-packages\torch\nn\modules
\dropout.py in forward(self, input)
     52
     53     def forward(self, input):
---> 54         return F.dropout(input, self.p, self.training, self.inplac
e)
     55
     56

E:\ProgramData\Anaconda3\envs\attentive\lib\site-packages\torch\nn\functio
nal.py in dropout(input, p, training, inplace)
    934     return (_VF.dropout_(input, p, training)
    935             if inplace
--> 936             else _VF.dropout(input, p, training))
    937
    938

KeyboardInterrupt:
```

In [36]:

```
# AGREE: use more negative instance(5)
start_training(model_name="AGREE", name="AGREE-n5", embedding_dim=6, train_epoch=5,
        num_negatives=5, batch_size=1024, lr=0.00001, drop_ratio=0.1, top_k=5)
```

```
number of groups: 290
number of users: 602
number of items: 7710
AGREE at embedding size: 6, epoch: 5, NDCG & HR: Top-5
Epoch 0, lr = 1e-05, input_type = user
.....................................................
Epoch 0, lr = 1e-05, input_type = group
.....................................................
Losses: {'user': 0.9960147632759275, 'group': 0.9812008813158787}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.2412, average Top-5 NDCG: 0.1539
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2207, average Top-5 NDCG: 0.1453
Epoch 1, lr = 5e-06, input_type = user
.....................................................
Epoch 1, lr = 5e-06, input_type = group
.....................................................
Losses: {'user': 0.9632067706510833, 'group': 0.9575228952885539}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3588, average Top-5 NDCG: 0.2275
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3221, average Top-5 NDCG: 0.2107
Epoch 2, lr = 5e-06, input_type = user
.....................................................
Epoch 2, lr = 5e-06, input_type = group
.....................................................
Losses: {'user': 0.9425803198456982, 'group': 0.9371402949473386}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4362, average Top-5 NDCG: 0.2780
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3972, average Top-5 NDCG: 0.2596
Epoch 3, lr = 5e-06, input_type = user
.....................................................
Epoch 3, lr = 5e-06, input_type = group
.....................................................
Losses: {'user': 0.9206137925222979, 'group': 0.9141517029483608}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4934, average Top-5 NDCG: 0.3165
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4483, average Top-5 NDCG: 0.2946
Epoch 4, lr = 5e-06, input_type = user
.....................................................
Epoch 4, lr = 5e-06, input_type = group
.....................................................
Losses: {'user': 0.8985922698346963, 'group': 0.8892562928087265}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5329, average Top-5 NDCG: 0.3425
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4800, average Top-5 NDCG: 0.3182
```

In [37]:

```
# AGREE: use more negative instance(5)
start_training(model_name="AGREE", name="AGREE-n6", embedding_dim=6, train_epoch=5,
        num_negatives=6, batch_size=1024, lr=0.00001, drop_ratio=0.1, top_k=5)
```

```
number of groups: 290
number of users: 602
number of items: 7710
AGREE at embedding size: 6, epoch: 5, NDCG & HR: Top-5
Epoch 0, lr = 1e-05, input_type = user
.................................................................
Epoch 0, lr = 1e-05, input_type = group
.................................................................
Losses: {'user': 0.9891845682045308, 'group': 0.9679760510910658}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.2535, average Top-5 NDCG: 0.1566
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2317, average Top-5 NDCG: 0.1515
Epoch 1, lr = 5e-06, input_type = user
.................................................................
Epoch 1, lr = 5e-06, input_type = group
.................................................................
Losses: {'user': 0.9429591259033214, 'group': 0.9357978438285043}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3734, average Top-5 NDCG: 0.2374
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3414, average Top-5 NDCG: 0.2213
Epoch 2, lr = 5e-06, input_type = user
.................................................................
Epoch 2, lr = 5e-06, input_type = group
.................................................................
Losses: {'user': 0.9149598534695986, 'group': 0.9078478401078759}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4615, average Top-5 NDCG: 0.2940
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4290, average Top-5 NDCG: 0.2789
Epoch 3, lr = 5e-06, input_type = user
.................................................................
Epoch 3, lr = 5e-06, input_type = group
.................................................................
Losses: {'user': 0.8851720945137304, 'group': 0.8784487239351432}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5213, average Top-5 NDCG: 0.3339
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4793, average Top-5 NDCG: 0.3124
Epoch 4, lr = 5e-06, input_type = user
.................................................................
Epoch 4, lr = 5e-06, input_type = group
.................................................................
Losses: {'user': 0.8549167787338176, 'group': 0.847793330583558}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5512, average Top-5 NDCG: 0.3554
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5103, average Top-5 NDCG: 0.3318
```

In [35]:

```
start_training(model_name="GREE", name="GREE-basic", embedding_dim=6, train_epoch=15,
        num_negatives=4, batch_size=1024, lr=0.00001, drop_ratio=0.1, top_k=5)
```

```
number of groups: 290
number of users: 602
number of items: 7710
GREE at embedding size: 6, epoch: 15, NDCG & HR: Top-5
Epoch 0, lr = 1e-05, input_type = user
.........................................
Epoch 0, lr = 1e-05, input_type = group
.........................................
Losses: {'user': 0.9990028688509349, 'group': 0.9961803443307509}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.1030, average Top-5 NDCG: 0.0621
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.1069, average Top-5 NDCG: 0.0632
Epoch 1, lr = 5e-06, input_type = user
.........................................
Epoch 1, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9924351422955843, 'group': 0.9916884389594028}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.1741, average Top-5 NDCG: 0.1072
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.1634, average Top-5 NDCG: 0.0991
Epoch 2, lr = 5e-06, input_type = user
.........................................
Epoch 2, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9880384568218781, 'group': 0.9873522041606254}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.2439, average Top-5 NDCG: 0.1556
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2372, average Top-5 NDCG: 0.1448
Epoch 3, lr = 5e-06, input_type = user
.........................................
Epoch 3, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9829999331205879, 'group': 0.9823914458151577}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3276, average Top-5 NDCG: 0.2088
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2986, average Top-5 NDCG: 0.1895
Epoch 4, lr = 5e-06, input_type = user
.........................................
Epoch 4, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9774620044695158, 'group': 0.9767678700336794}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3894, average Top-5 NDCG: 0.2511
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3600, average Top-5 NDCG: 0.2293
Epoch 5, lr = 5e-06, input_type = user
.........................................
Epoch 5, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9714526415416797, 'group': 0.970530356409328}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4468, average Top-5 NDCG: 0.2852
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4166, average Top-5 NDCG: 0.2629
Epoch 6, lr = 2.5e-06, input_type = user
.........................................
Epoch 6, lr = 2.5e-06, input_type = group
```

```
.........................................
Losses: {'user': 0.9657218232580399, 'group': 0.966523562158857}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4648, average Top-5 NDCG: 0.2972
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4317, average Top-5 NDCG: 0.2744
Epoch 7, lr = 2.5e-06, input_type = user
.........................................
Epoch 7, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.9623192835181474, 'group': 0.9630779340153649}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4847, average Top-5 NDCG: 0.3103
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4510, average Top-5 NDCG: 0.2855
Epoch 8, lr = 2.5e-06, input_type = user
.........................................
Epoch 8, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.958669314258977, 'group': 0.9591552032635056}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4950, average Top-5 NDCG: 0.3176
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4607, average Top-5 NDCG: 0.2912
Epoch 9, lr = 2.5e-06, input_type = user
.........................................
Epoch 9, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.9548591577215653, 'group': 0.9552388760238008}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5063, average Top-5 NDCG: 0.3246
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4779, average Top-5 NDCG: 0.3009
Epoch 10, lr = 2.5e-06, input_type = user
.........................................
Epoch 10, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.9507652422928974, 'group': 0.9510902497503493}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5186, average Top-5 NDCG: 0.3314
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4890, average Top-5 NDCG: 0.3085
Epoch 11, lr = 1.25e-06, input_type = user
.........................................
Epoch 11, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.9472699487236599, 'group': 0.948622135078015}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5236, average Top-5 NDCG: 0.3349
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4938, average Top-5 NDCG: 0.3121
Epoch 12, lr = 1.25e-06, input_type = user
.........................................
Epoch 12, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.9451928235573408, 'group': 0.9461878892245477}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5279, average Top-5 NDCG: 0.3388
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5000, average Top-5 NDCG: 0.3157
Epoch 13, lr = 1.25e-06, input_type = user
```

```
.......................................
Epoch 13, lr = 1.25e-06, input_type = group
.......................................
Losses: {'user': 0.9426836839256898, 'group': 0.9441859741059561}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5319, average Top-5 NDCG: 0.3414
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5041, average Top-5 NDCG: 0.3186
Epoch 14, lr = 1.25e-06, input_type = user
.......................................
Epoch 14, lr = 1.25e-06, input_type = group
.......................................
Losses: {'user': 0.9404967940099179, 'group': 0.9419899081426953}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5359, average Top-5 NDCG: 0.3446
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5083, average Top-5 NDCG: 0.3212
```

In [34]:

```
start_training(model_name="GNCF", name="GNCF-basic", embedding_dim=6, train_epoch=15,
        num_negatives=4, batch_size=1024, lr=0.00001, drop_ratio=0.1, top_k=5)
```

```
number of groups: 290
number of users: 602
number of items: 7710
GNCF at embedding size: 6, epoch: 15, NDCG & HR: Top-5
Epoch 0, lr = 1e-05, input_type = user
.........................................
Epoch 0, lr = 1e-05, input_type = group
.........................................
Losses: {'user': 0.9970118863086133, 'group': 0.9893124192750373}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.1661, average Top-5 NDCG: 0.1045
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.1628, average Top-5 NDCG: 0.1049
Epoch 1, lr = 5e-06, input_type = user
.........................................
Epoch 1, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9820374517746321, 'group': 0.9780903872178526}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.2502, average Top-5 NDCG: 0.1633
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.2331, average Top-5 NDCG: 0.1512
Epoch 2, lr = 5e-06, input_type = user
.........................................
Epoch 2, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9718807365583337, 'group': 0.9672182902457223}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.3439, average Top-5 NDCG: 0.2214
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3041, average Top-5 NDCG: 0.1958
Epoch 3, lr = 5e-06, input_type = user
.........................................
Epoch 3, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9603572860189652, 'group': 0.9557785674287619}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4050, average Top-5 NDCG: 0.2651
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.3772, average Top-5 NDCG: 0.2383
Epoch 4, lr = 5e-06, input_type = user
.........................................
Epoch 4, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9476078310874835, 'group': 0.943191213537506}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4532, average Top-5 NDCG: 0.2975
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4221, average Top-5 NDCG: 0.2657
Epoch 5, lr = 5e-06, input_type = user
.........................................
Epoch 5, lr = 5e-06, input_type = group
.........................................
Losses: {'user': 0.9334794183344808, 'group': 0.9294579580527584}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.4887, average Top-5 NDCG: 0.3213
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4572, average Top-5 NDCG: 0.2861
Epoch 6, lr = 2.5e-06, input_type = user
.........................................
Epoch 6, lr = 2.5e-06, input_type = group
```

```
.........................................
Losses: {'user': 0.9206731371257616, 'group': 0.9207429987232701}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5033, average Top-5 NDCG: 0.3297
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4731, average Top-5 NDCG: 0.2951
Epoch 7, lr = 2.5e-06, input_type = user
.........................................
Epoch 7, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.9130956407815423, 'group': 0.913189791632888}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5209, average Top-5 NDCG: 0.3390
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4814, average Top-5 NDCG: 0.3012
Epoch 8, lr = 2.5e-06, input_type = user
.........................................
Epoch 8, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.9049237210908798, 'group': 0.9056857033651702}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5336, average Top-5 NDCG: 0.3458
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4869, average Top-5 NDCG: 0.3076
Epoch 9, lr = 2.5e-06, input_type = user
.........................................
Epoch 9, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.896492890689684, 'group': 0.8980668430425682}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5442, average Top-5 NDCG: 0.3518
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.4938, average Top-5 NDCG: 0.3129
Epoch 10, lr = 2.5e-06, input_type = user
.........................................
Epoch 10, lr = 2.5e-06, input_type = group
.........................................
Losses: {'user': 0.8876545547893445, 'group': 0.8899965561977049}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5508, average Top-5 NDCG: 0.3566
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5021, average Top-5 NDCG: 0.3202
Epoch 11, lr = 1.25e-06, input_type = user
.........................................
Epoch 11, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8801597685508379, 'group': 0.8853112544332232}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5561, average Top-5 NDCG: 0.3599
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5062, average Top-5 NDCG: 0.3224
Epoch 12, lr = 1.25e-06, input_type = user
.........................................
Epoch 12, lr = 1.25e-06, input_type = group
.........................................
Losses: {'user': 0.8756168919788072, 'group': 0.8808863700922925}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5621, average Top-5 NDCG: 0.3627
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5083, average Top-5 NDCG: 0.3230
Epoch 13, lr = 1.25e-06, input_type = user
```

```
.......................................
Epoch 13, lr = 1.25e-06, input_type = group
.......................................
Losses: {'user': 0.8714240897165556, 'group': 0.8767652773803054}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5651, average Top-5 NDCG: 0.3636
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5138, average Top-5 NDCG: 0.3260
Epoch 14, lr = 1.25e-06, input_type = user
.......................................
Epoch 14, lr = 1.25e-06, input_type = group
.......................................
Losses: {'user': 0.8665758738255883, 'group': 0.872523874517471}
Evaluating: 0.0% 10.3% 20.3% 30.2% 40.2% 50.2% 60.1% 70.1% 80.1% 90.0%
User-- average Top-5 Hit Rate: 0.5661, average Top-5 NDCG: 0.3658
Evaluating: 0.0% 10.3% 20.0% 30.3% 40.0% 50.3% 60.0% 70.3% 80.0% 90.3%
Group-- average Top-5 Hit Rate: 0.5179, average Top-5 NDCG: 0.3286
```