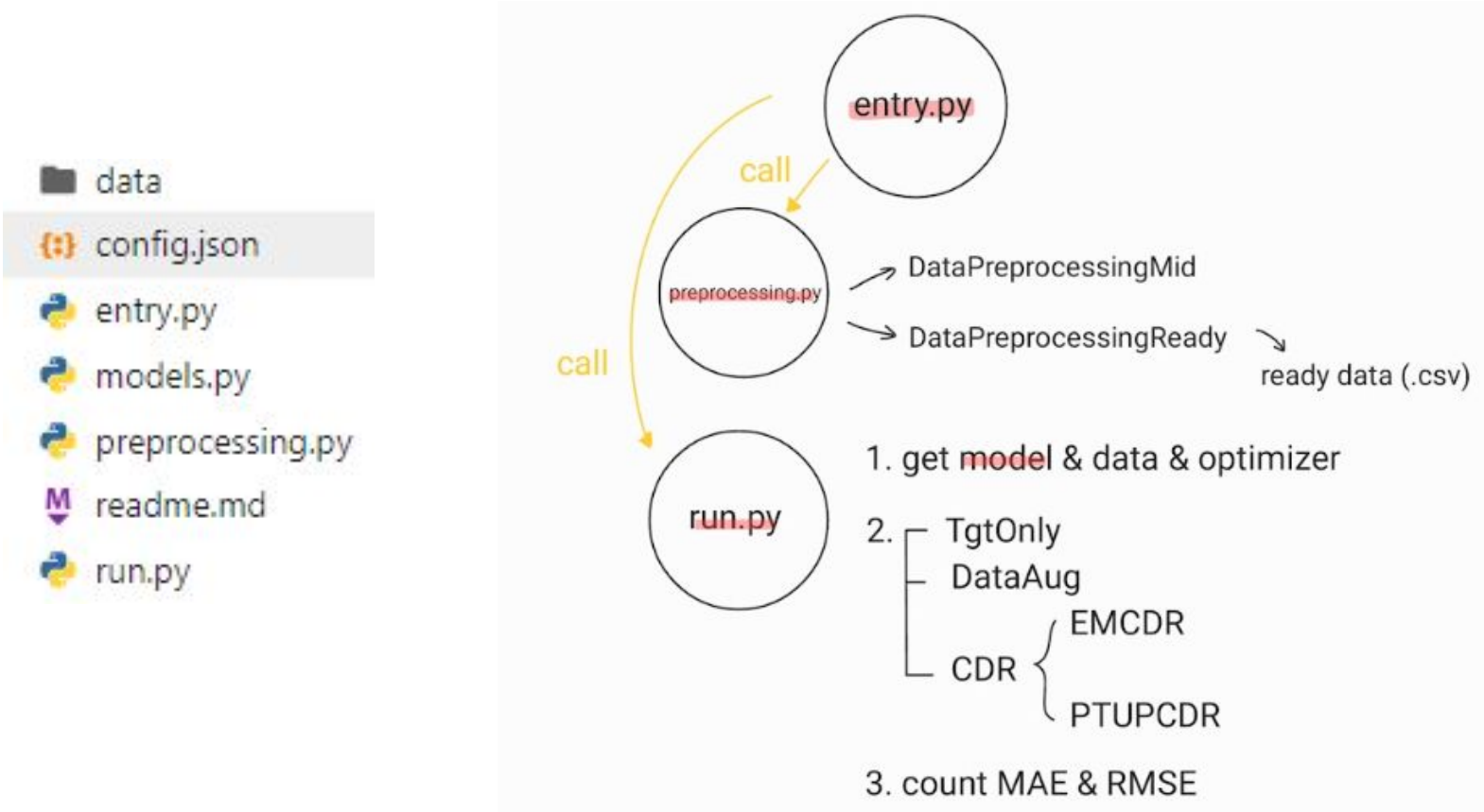
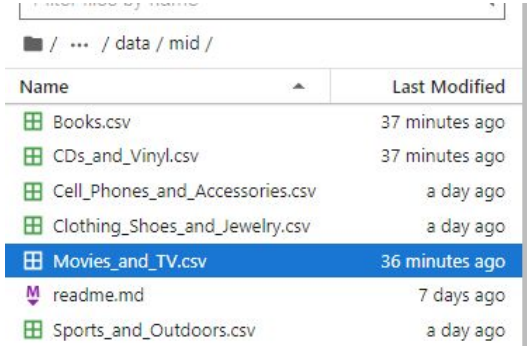
PTUPCDR


data
config.json
entry.py
models.py
preprocessing.py
readme.md
run.py
entry.py
call
preprocessing.py
DataPreprocessingMid
DataPreprocessingReady
ready data (.csv)
call
run.py
1. get model & data & optimizer
2.
TgtOnly
DataAug
CDR
EMCDR
PTUPCDR
3. count MAE & RMSE

```python
class DataPreprocessingMid():    #Fuction目標->處理raw data，提取所需要的欄位資料(line 21)
    def __init__(self,
                 root,
                 dealing):
        self.root = root
        self.dealing = dealing

    def main(self):
        print('Parsing ' + self.dealing + ' Mid...')
        re = []
        with gzip.open(self.root + 'raw/reviews_' + self.dealing + '_5.json.gz', 'rb') as f:
            for line in tqdm.tqdm(f, smoothing=0, mininterval=1.0):
                line = json.loads(line)
                re.append([line['reviewerID'], line['asin'], line['overall']])
        re = pd.DataFrame(re, columns=['uid', 'iid', 'y'])
        print(self.dealing + ' Mid Done.')
        re.to_csv(self.root + 'mid/' + self.dealing + '.csv', index=0)
        return re    #type(re) = dataframe

```

{'reviewerID': 'ADZPIG9QOCDG5', 'asin': '0005019281', 'reviewerName': 'Alice L. Larson "alice-loves-books"', 'helpful': [0, 0], 'reviewText': 'This is a charming version of the classic Dic
ken\'s tale. Henry Winkler makes a good showing as the "Scrooge" character. Even though you know what will happen this version has enough of a change to make it better that average. If you
love A Christmas Carol in any version, then you will love this.', 'overall': 4.0, 'summary': 'good version of a classic', 'unixReviewTime': 1203984000, 'reviewTime': '02 26, 2008'}



	uid	iid	y
1	ADZPIG9QOCDG5	0005019281	4.0
2	A35947ZP82G7JH	0005019281	3.0
3	A3UORV8A9D5L2E	0005019281	3.0
4	A1VKW06X1O2X7V	0005019281	5.0
5	A3R27T4HADWFFJ	0005019281	4.0
6	A2L0G56BNOTX6S	0005019281	5.0
7	A5NYUBEKXFLX5	0005019281	5.0
8	A2DJ8B8GE4V2VD	0005019281	5.0
9	AWF2S3UNW9UA0	0005019281	5.0
10	A3O4UUT83DG3OU	0005019281	5.0

Example

```python
class DataPreprocessingReady():     #Fuction目標->進行數據映射&拆分
    def __init__(self,
                 root,
                 src_tgt_pairs,
                 task,
                 ratio):
        self.root = root
        self.src = src_tgt_pairs[task]['src']
        self.tgt = src_tgt_pairs[task]['tgt']
        self.ratio = ratio

    def read_mid(self, field):
        path = self.root + 'mid/' + field + '.csv'
        re = pd.read_csv(path)
        return re
```

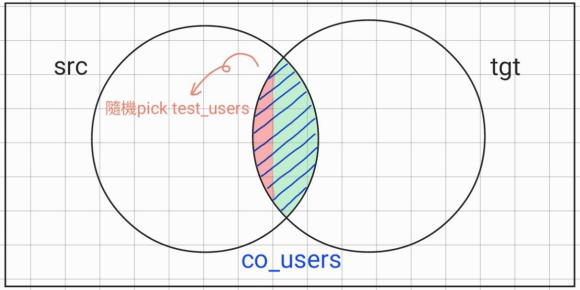
```python
    def main(self):
        src = self.read_mid(self.src)
        tgt = self.read_mid(self.tgt)
        src, tgt = self.mapper(src, tgt)
        train_src, train_tgt, train_meta, test = self.split(src, tgt)
        self.save(train_src, train_tgt, train_meta, test)
```

```
▼ root:
    use_cuda: 0
    root: "./data/"
  ▼ src_tgt_pairs:
    ▼ 1:
        src: "Movies_and_TV"
        tgt: "CDs_and_Vinyl"
        uid: 181187
        iid: 114495
        batchsize_src: 256
        batchsize_tgt: 256
        batchsize_meta: 128
        batchsize_map: 64
        batchsize_test: 128
    ▶ 2:
    ▶ 3:
    ▶ 4:
    ▶ 5:
    ▶ 6:
    emb_dim: 10
    meta_dim: 50
    num_fields: 2
    wd: 0
```

```
43      def mapper(self, src, tgt):
44          print('Source inters: {}, uid: {}, iid: {}.'.format(len(src), len(set(src.uid)), len(set(src.iid))))
45          print('Target inters: {}, uid: {}, iid: {}.'.format(len(tgt), len(set(tgt.uid)), len(set(tgt.iid))))
46          co_uid = set(src.uid) & set(tgt.uid)
47          all_uid = set(src.uid) | set(tgt.uid)
48          print('All uid: {}, Co uid: {}.'.format(len(all_uid), len(co_uid)))
49          uid_dict = dict(zip(all_uid, range(len(all_uid))))
50              #建立user_id字典，將所有user_id重新編號為某一個範圍內的連續整數
51          iid_dict_src = dict(zip(set(src.iid), range(len(set(src.iid)))))
52          iid_dict_tgt = dict(zip(set(tgt.iid), range(len(set(src.iid)), len(set(src.iid)) + len(set(tgt.iid)))))
53              #例如:0~10000為src_iid，10001~20000為tgt_iid
54          src.uid = src.uid.map(uid_dict)
55          src.iid = src.iid.map(iid_dict_src)
56          tgt.uid = tgt.uid.map(uid_dict)
57          tgt.iid = tgt.iid.map(iid_dict_tgt)
58              #根據字典重新編號
59          return src, tgt
60
```

```
91      def main(self):
92          src = self.read_mid(self.src)
93          tgt = self.read_mid(self.tgt)
94          src, tgt = self.mapper(src, tgt)
95          train_src, train_tgt, train_meta, test = self.split(src, tgt)
96          self.save(train_src, train_tgt, train_meta, test)
97
```

```python
def split(self, src, tgt):
    print('All iid: {}.'.format(len(set(src.iid) | set(tgt.iid))))
    src_users = set(src.uid.unique())
    tgt_users = set(tgt.uid.unique())
    co_users = src_users & tgt_users
    test_users = set(random.sample(co_users, round(self.ratio[1] * len(co_users))))
        #在co_users中隨機選擇測試者
    train_src = src
    train_tgt = tgt[tgt['uid'].isin(tgt_users - test_users)]
    test = tgt[tgt['uid'].isin(test_users)]
    pos_seq_dict = self.get_history(src, co_users)
    train_meta = tgt[tgt['uid'].isin(co_users - test_users)]
    train_meta['pos_seq'] = train_meta['uid'].map(pos_seq_dict)
    test['pos_seq'] = test['uid'].map(pos_seq_dict)
    return train_src, train_tgt, train_meta, test

```



train_src
train_tgt = tgt\test_users
train_meta → cross domain, 將src的history結合tgt
→ uid (tgt), iid (tgt), pos_seq (src history)
test = test_users
→ uid (tgt), iid (tgt), pos_seq (src history)

```python
def main(self):
    src = self.read_mid(self.src)
    tgt = self.read_mid(self.tgt)
    src, tgt = self.mapper(src, tgt)
    train_src, train_tgt, train_meta, test = self.split(src, tgt)
    self.save(train_src, train_tgt, train_meta, test)

```

```
70      def split(self, src, tgt):
71          print('All iid: {}.'.format(len(set(src.iid) | set(tgt.iid))))
72          src_users = set(src.uid.unique())
73          tgt_users = set(tgt.uid.unique())
74          co_users = src_users & tgt_users
75          test_users = set(random.sample(co_users, round(self.ratio[1] * len(co_users))))
76              #在co_users中隨機選擇測試者
77          train_src = src
78          train_tgt = tgt[tgt['uid'].isin(tgt_users - test_users)]
79          test = tgt[tgt['uid'].isin(test_users)]
80          pos_seq_dict = self.get_history(src, co_users)
81          train_meta = tgt[tgt['uid'].isin(co_users - test_users)]
82          train_meta['pos_seq'] = train_meta['uid'].map(pos_seq_dict)
83          test['pos_seq'] = test['uid'].map(pos_seq_dict)
84          return train_src, train_tgt, train_meta, test
85
```

```
def get_history(self, data, uid_set):
    pos_seq_dict = {}
    for uid in tqdm.tqdm(uid_set):
        pos = data[(data.uid == uid) & (data.y > 3)].iid.values.tolist()
            #將user評分大於3的item提出
        pos_seq_dict[uid] = pos
    return pos_seq_dict
        #key->uid，value->list of item(score>3)
```

```
91      def main(self):
92          src = self.read_mid(self.src)
93          tgt = self.read_mid(self.tgt)
94          src, tgt = self.mapper(src, tgt)
95          train_src, train_tgt, train_meta, test = self.split(src, tgt)
96          self.save(train_src, train_tgt, train_meta, test)
97
```

```
87      def save(self, train_src, train_tgt, train_meta, test):
88          output_root = self.root + 'ready/_' + str(int(self.ratio[0] * 10)) + '_' + str(int(self.ratio[1] * 10)) + \
89                        '/tgt_' + self.tgt + '_src_' + self.src
90          if not os.path.exists(output_root):
91              os.makedirs(output_root)
92          print(output_root)
93          train_src.to_csv(output_root + '/train_src.csv', sep=',', header=None, index=False)
94          train_tgt.to_csv(output_root + '/train_tgt.csv', sep=',', header=None, index=False)
95          train_meta.to_csv(output_root +  '/train_meta.csv', sep=',', header=None, index=False)
96          test.to_csv(output_root + '/test.csv', sep=',', header=None, index=False)
```

```
91      def main(self):
92          src = self.read_mid(self.src)
93          tgt = self.read_mid(self.tgt)
94          src, tgt = self.mapper(src, tgt)
95          train_src, train_tgt, train_meta, test = self.split(src, tgt)
96          self.save(train_src, train_tgt, train_meta, test)
97
```

train_meta.csv

1	78500	57587	5.0	[5792, 16996, 28090, 46910, 30196, 42645, 47841, 36596, 37029, 37682, 731, 32285, 9091, 35279, 15085, 22292, 24932, 19265, 46831, 31656, 49212, 48538, 49602, 8438, 361…
2	154356	57587	5.0	[5792, 18495, 25542, 46110, 45285, 8516, 6717, 28564, 8873, 36893, 26119, 23542, 49731, 36675, 5472, 24193, 36572, 9978, 40471, 17446, 43588, 8798, 48625, 8729, 30703, …
3	139666	105458	5.0	[30564, 29719, 1370, 46422, 34932, 17782, 39342, 18122, 23066, 29347, 32187, 37453, 41283, 18783, 4390]
4	114220	105458	5.0	[32697, 30115, 9248, 24152]
5	49823	105458	4.0	[25235, 32697, 44806, 43412, 44694, 20661]
6	40484	105458	3.0	[10684, 30295, 41199, 36115, 33048, 40755, 40653, 2841, 35116, 23519, 38808, 39167, 14682, 14429, 9472, 10617, 39492, 10097, 28688, 6583, 41293, 44795, 7854, 28662]
7	103496	105458	5.0	[35870, 37709, 44694, 35624, 36792, 35623, 11449, 9512, 26203]
8	40697	105458	3.0	[48836, 14477, 22399, 1829, 3404, 11264]
9	8742	91432	3.0	[49972, 10291, 42972, 36328, 31126, 23359, 14810, 32347, 41347, 27777]
10	35398	91432	4.0	[32159, 11923, 4283, 32651, 25015, 14909, 11575, 33271, 11699, 43319]
11	28891	91432	5.0	[32159, 7247, 14527, 22149, 33689, 1046, 19095, 37677, 45619, 13393, 27879, 43941, 43419, 21172, 11805, 34020, 31976, 18764, 33577, 30406, 8886, 19821, 28914]
12	76854	91432	5.0	[32159, 35411, 25389, 192, 19895, 19254, 10785]
13	135702	91432	3.0	[27930, 2179, 23522, 7066, 39639, 5501, 27497, 2656, 27214, 511, 41347, 35294, 27285, 38439, 27201, 38475, 1604, 7929, 35293, 2787, 34127, 38121, 4598, 29816, 17937, 30…
14	54346	91432	5.0	[27707, 32159, 15447, 42373, 38403, 26164, 41347, 26149]
15	138826	91432	5.0	[32159, 22399, 34916, 48529, 20629, 49486, 41347, 5754, 39151, 23100, 32120, 11661, 15731, 4665, 31636, 37587, 15883]

test.csv

1	35864	105458	5.0	[49789, 37755, 35724, 43989, 48625, 10684, 24230, 40817, 24446, 32871, 37760, 19029, 33, 20730, 30648, 20668, 20238, 6987, 18413, 14704, 27961, 20318, 40755, 25981, 20245…
2	131594	105458	1.0	[42990, 36786, 31475, 13319, 25442, 44594]
3	129019	91432	5.0	[27707, 32159, 26119, 43097, 8038, 1940, 17534, 22719, 34710, 7401, 334, 24416, 14229, 39521, 24580, 36814, 30565, 14799, 5806, 36366, 43490, 4120, 45076, 43632, 45306, 29…
4	165842	91432	5.0	[32159, 44705, 7408, 4950, 37340, 48260, 32958, 12420, 27178, 3502, 14954, 12836, 6799, 4606, 31377, 17428, 24059, 40676, 48544, 37299]
5	99559	91432	5.0	[27707, 32159, 27389, 3404, 16284, 44041, 31534, 34394, 31636, 32083, 37587, 573, 37293]
6	140687	91432	4.0	[32159, 41347, 43412, 31636, 37587, 44694, 36792]
7	156715	91432	5.0	[32159, 49730, 49731, 29782, 30302, 18875, 46743, 22091, 47744, 26164, 24241, 30199, 1225, 36626, 5339, 15092, 41734, 37587, 8715, 27081, 28809, 37293, 20569, 25354, 2074…
8	129472	83528	5.0	[25871, 10159, 15784, 40300, 15844, 28836, 13584]
9	119555	83528	5.0	[25871, 1223, 3035, 2841]
10	105159	83528	5.0	[25871, 1007, 16256, 40793, 14429]
11	22620	83528	5.0	[25871, 267, 14805]
12	177129	64660	5.0	[11257, 11893, 17308, 14096, 5361, 46567, 2992, 27846, 16825, 49335, 29061, 44819, 28784, 2603]
13	27028	73856	5.0	[18495, 9205, 29752, 47904, 38403, 39389, 273, 43444, 45703, 7698, 33384, 25729, 37271, 32365]
14	63861	73856	4.0	[18495, 39712, 44097, 14847, 30216, 33241, 31919, 42701, 15917, 36617, 32191, 7638, 41213, 33384, 28537, 9700, 6261, 13436, 32181, 46367, 6019, 10705, 39018, 22014, 35909…
15	127389	59066	5.0	[6962, 43977, 8818, 18617, 24092, 37755, 19213, 8189, 16543, 5153, 10291, 1344, 10684, 48260, 4738, 49551, 8101, 12540, 3010, 27360, 43091, 13655, 32267, 25381, 37965, 440…

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
161     def get_data(self):
162         print('========Reading data========')
163         # retrieve a data_loader
164         data_src = self.read_log_data(self.src_path, self.batchsize_src)
165         print('src {} iter / batchsize = {} '.format(len(data_src), self.batchsize_src))
166 
167         # retrieve a data_loader
168         data_tgt = self.read_log_data(self.tgt_path, self.batchsize_tgt)
169         print('tgt {} iter / batchsize = {} '.format(len(data_tgt), self.batchsize_tgt))
170 
171         # retrieve a data_loader
172         data_meta = self.read_log_data(self.meta_path, self.batchsize_meta, history=True)
173         print('meta {} iter / batchsize = {} '.format(len(data_meta), self.batchsize_meta))
174 
175         # retrieve a data_loader of unique_uid data
176         data_map = self.read_map_data()
177         print('map {} iter / batchsize = {} '.format(len(data_map), self.batchsize_map))
178 
179         # retrieve a  concat(src,tgt) feature and label
180         data_aug = self.read_aug_data()
181         print('aug {} iter / batchsize = {} '.format(len(data_aug), self.batchsize_aug))
182 
183         # retrieve a data_loader
184         data_test = self.read_log_data(self.test_path, self.batchsize_test, history=True)
185         print('test {} iter / batchsize = {} '.format(len(data_test), self.batchsize_test))
186 
187         return data_src, data_tgt, data_meta, data_map, data_aug, data_test
188 
```

```python
    def read_log_data(self, path, batchsize, history=False):
        if not history:
            cols = ['uid', 'iid', 'y']
            x_col = ['uid', 'iid']
            y_col = ['y']
            data = pd.read_csv(path, header=None)
            data.columns = cols
            X = torch.tensor(data[x_col].values, dtype=torch.long)
                #轉換成tensor類型for pytorch
                #tensor是用來表示在向量、純量或其他張量間線性關係的多線性函數，也是在深度學習/機器學習運算的基本元素
            y = torch.tensor(data[y_col].values, dtype=torch.long)
            if self.use_cuda:
                #如果系統具有NVIDIA的GPU且已安裝CUDA，則利用GPU加速計算
                X = X.cuda()
                y = y.cuda()
            dataset = TensorDataset(X, y)
                #打包
            data_iter = DataLoader(dataset, batchsize, shuffle=True)
                #批量處理及洗牌
                #DataLoader的好處
            return data_iter
        else:
            #需處理pos_seq的情況
            data = pd.read_csv(path, header=None)
            cols = ['uid', 'iid', 'y', 'pos_seq']
            x_col = ['uid', 'iid']
            y_col = ['y']
            data.columns = cols
            pos_seq = keras.preprocessing.sequence.pad_sequences(data.pos_seq.map(self.seq_extractor), maxlen=20, padding='post')
                #self.seq_extractor轉成純數字列表，並填充序列、截斷(maxlen=20)，確保長度一致
            pos_seq = torch.tensor(pos_seq, dtype=torch.long)
            id_fea = torch.tensor(data[x_col].values, dtype=torch.long)
            X = torch.cat([id_fea, pos_seq], dim=1)
                #沿著dim=1的維度生成特徵矩陣X，組合兩種不同的特徵
            y = torch.tensor(data[y_col].values, dtype=torch.long)
            if self.use_cuda:
                X = X.cuda()
                y = y.cuda()
            dataset = TensorDataset(X, y)
            data_iter = DataLoader(dataset, batchsize, shuffle=True)
            return data_iter

```

```python
    def seq_extractor(self, x):
        x = x.rstrip(']').lstrip('[').split(', ')
            #得到純數字序列，例如1, 2, 3, 4
        for i in range(len(x)):
            try:
                x[i] = int(x[i])
            except:
                x[i] = self.iid_all
        return np.array(x)
            #type = Numpy

```

```
161     def get_data(self):
162         print('========Reading data========')
163         # retrieve a data_loader
164         data_src = self.read_log_data(self.src_path, self.batchsize_src)
165         print('src {} iter / batchsize = {} '.format(len(data_src), self.batchsize_src))
166 
167         # retrieve a data_loader
168         data_tgt = self.read_log_data(self.tgt_path, self.batchsize_tgt)
169         print('tgt {} iter / batchsize = {} '.format(len(data_tgt), self.batchsize_tgt))
170 
171         # retrieve a data_loader
172         data_meta = self.read_log_data(self.meta_path, self.batchsize_meta, history=True)
173         print('meta {} iter / batchsize = {} '.format(len(data_meta), self.batchsize_meta))
174 
175         # retrieve a data_loader of unique_uid data
176         data_map = self.read_map_data()
177         print('map {} iter / batchsize = {} '.format(len(data_map), self.batchsize_map))
178 
179         # retrieve a  concat(src,tgt) feature and label
180         data_aug = self.read_aug_data()
181         print('aug {} iter / batchsize = {} '.format(len(data_aug), self.batchsize_aug))
182 
183         # retrieve a data_loader
184         data_test = self.read_log_data(self.test_path, self.batchsize_test, history=True)
185         print('test {} iter / batchsize = {} '.format(len(data_test), self.batchsize_test))
186 
187         return data_src, data_tgt, data_meta, data_map, data_aug, data_test
188 
```

EMCDR

```
102     def read_map_data(self):
103         cols = ['uid', 'iid', 'y', 'pos_seq']
104         data = pd.read_csv(self.meta_path, header=None)
105         data.columns = cols
106         X = torch.tensor(data['uid'].unique(), dtype=torch.long)
107             #獲取uid唯一值
108         y = torch.tensor(np.array(range(X.shape[0])), dtype=torch.long)
109             #創建和X相同長度的數組，數值是X tensor的索引
110         if self.use_cuda:
111             X = X.cuda()
112             y = y.cuda()
113         dataset = TensorDataset(X, y)
114         data_iter = DataLoader(dataset, self.batchsize_map, shuffle=True)
115         return data_iter
116
```

```
 98         elif stage == 'train_map':#訓練映射模型階段->目的是將源領域的用戶嵌入映射到目標領域的用戶嵌入
 99             src_emb = self.src_model.uid_embedding(x.unsqueeze(1)).squeeze()
100             src_emb = self.mapping.forward(src_emb)
101             tgt_emb = self.tgt_model.uid_embedding(x.unsqueeze(1)).squeeze()
102             return src_emb, tgt_emb
```

```
161     def get_data(self):
162         print('========Reading data========')
163         # retrieve a data_loader
164         data_src = self.read_log_data(self.src_path, self.batchsize_src)
165         print('src {} iter / batchsize = {} '.format(len(data_src), self.batchsize_src))
166 
167         # retrieve a data_loader
168         data_tgt = self.read_log_data(self.tgt_path, self.batchsize_tgt)
169         print('tgt {} iter / batchsize = {} '.format(len(data_tgt), self.batchsize_tgt))
170 
171         # retrieve a data_loader
172         data_meta = self.read_log_data(self.meta_path, self.batchsize_meta, history=True)
173         print('meta {} iter / batchsize = {} '.format(len(data_meta), self.batchsize_meta))
174 
175         # retrieve a data_loader of unique_uid data
176         data_map = self.read_map_data()
177         print('map {} iter / batchsize = {} '.format(len(data_map), self.batchsize_map))
178 
179         # retrieve a  concat(src,tgt) feature and label
180         data_aug = self.read_aug_data()
181         print('aug {} iter / batchsize = {} '.format(len(data_aug), self.batchsize_aug))
182 
183         # retrieve a data_loader
184         data_test = self.read_log_data(self.test_path, self.batchsize_test, history=True)
185         print('test {} iter / batchsize = {} '.format(len(data_test), self.batchsize_test))
186 
187         return data_src, data_tgt, data_meta, data_map, data_aug, data_test
188 
```

```
    def read_aug_data(self):
        cols_train = ['uid', 'iid', 'y']
        x_col = ['uid', 'iid']
        y_col = ['y']
        src = pd.read_csv(self.src_path, header=None)
        src.columns = cols_train
        tgt = pd.read_csv(self.tgt_path, header=None)
        tgt.columns = cols_train

        X_src = torch.tensor(src[x_col].values, dtype=torch.long)
        y_src = torch.tensor(src[y_col].values, dtype=torch.long)
        X_tgt = torch.tensor(tgt[x_col].values, dtype=torch.long)
        y_tgt = torch.tensor(tgt[y_col].values, dtype=torch.long)
        X = torch.cat([X_src, X_tgt])
        y = torch.cat([y_src, y_tgt])
        if self.use_cuda:
            X = X.cuda()
            y = y.cuda()
        dataset = TensorDataset(X, y)
        data_iter = DataLoader(dataset, self.batchsize_aug, shuffle=True)

        return data_iter

```

▼ 1:
- src: "Movies_and_TV"
- tgt: "CDs_and_Vinyl"
- uid: 181187
- iid: 114495
- batchsize_src: 256
- batchsize_tgt: 256
- batchsize_meta: 128
- batchsize_map: 64
- batchsize_test: 128

▼ 2:
- src: "Books"
- tgt: "Movies_and_TV"
- uid: 690240
- iid: 418034
- batchsize_src: 512
- batchsize_tgt: 512
- batchsize_meta: 512
- batchsize_map: 128
- batchsize_test: 256

▼ 3:
- src: "Books"
- tgt: "CDs_and_Vinyl"
- uid: 662188
- iid: 432425
- batchsize_src: 512
- batchsize_tgt: 512
- batchsize_meta: 512
- batchsize_map: 128
- batchsize_test: 256

'tgt_mae': 4.445978164672852,
'tgt_rmse': 5.13602876663208,
'aug_mae': 1.4493868350982666,
'aug_rmse': 1.9287296533584595,
'emcdr_mae': 1.2723376750946045,
'emcdr_rmse': 1.5767518281936646,
'ptupcdr_mae': 1.1183274984359741,
'ptupcdr_rmse': 1.4527101516723633

'tgt_mae': 4.31486701965332,
'tgt_rmse': 4.923935890197754,
'aug_mae': 3.681924343109131,
'aug_rmse': 4.509251594543457,
'emcdr_mae': 3.9957830905914307,
'emcdr_rmse': 4.214810371398926,
'ptupcdr_mae': 2.1298327445983887,
'ptupcdr_rmse': 2.755762815475464

▼ 4:
- src: "Clothing_Shoes_and_Jewelry"
- tgt: "Cell_Phones_and_Accessories"
- uid: 64735
- iid: 33462
- batchsize_src: 128
- batchsize_tgt: 128
- batchsize_meta: 128
- batchsize_map: 64
- batchsize_test: 128

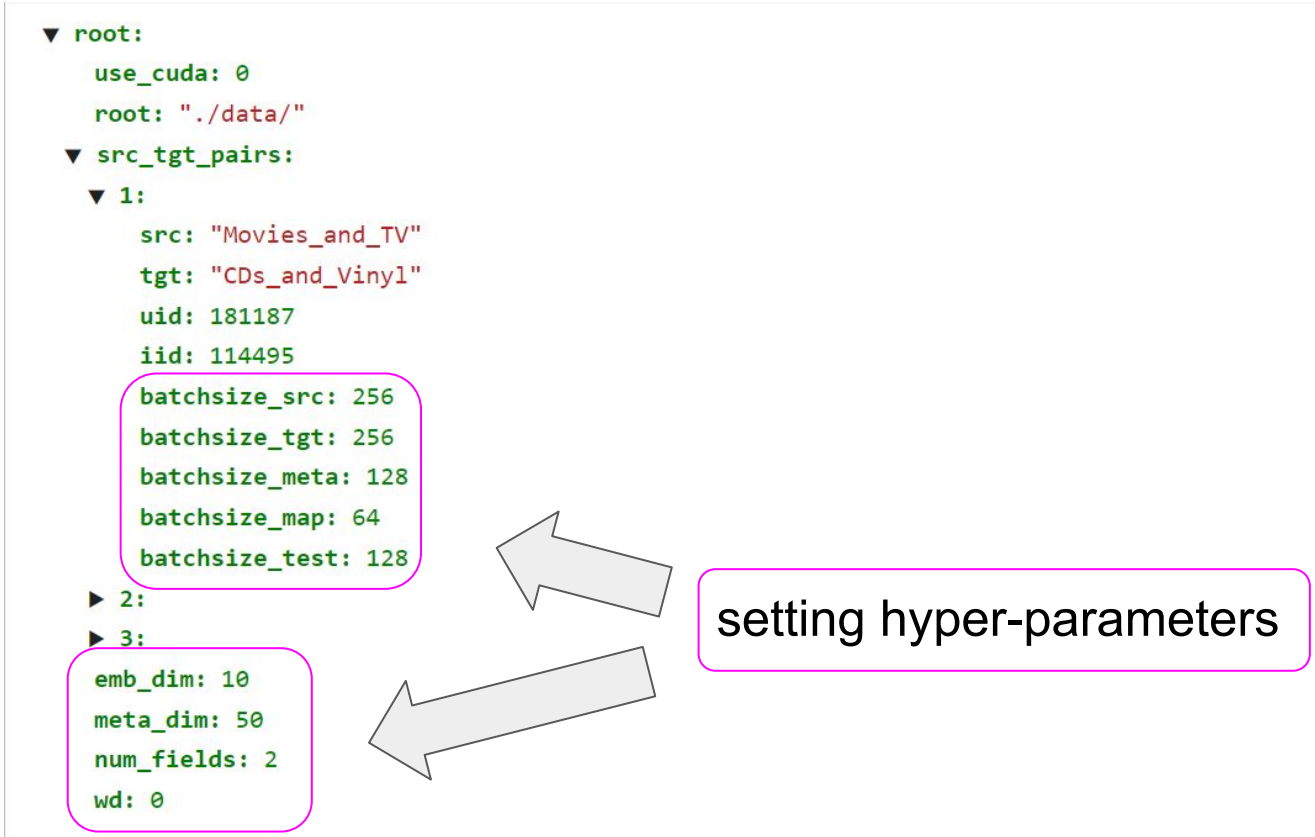
▼ 5:
- src: "Cell_Phones_and_Accessories"
- tgt: "Sports_and_Outdoors"
- uid: 60621
- iid: 28786
- batchsize_src: 128
- batchsize_tgt: 128
- batchsize_meta: 128
- batchsize_map: 64
- batchsize_test: 128

▼ 6:
- src: "Sports_and_Outdoors"
- tgt: "Clothing_Shoes_and_Jewelry"
- uid: 71077
- iid: 41390
- batchsize_src: 128
- batchsize_tgt: 128
- batchsize_meta: 128
- batchsize_map: 64
- batchsize_test: 128

entry

```python
if __name__ == '__main__':
    config_path = 'config.json'
    args, config = prepare(config_path)
    # assign gpu
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    # 指令para --process_data_mid=True，則執行以下
    if args.process_data_mid:
        for dealing in ['Books', 'CDs_and_Vinyl', 'Movies_and_TV']:
            DataPreprocessingMid(config['root'], dealing).main()
    # 指令para --process_data_ready=True，則執行以下
    if args.process_data_ready:
        # 實驗採3個冷啟動比例
        for ratio in [[0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]:
            # 實驗有3個task
            for task in ['1', '2', '3']:
                # task_1：src:"Movies_and_TV"; tgt:"CDs_and_Vinyl"
                # task_2：src:"Books"        ; tgt:"Movies_and_TV"
                # task_3：src:"Books"        ; tgt:"CDs_and_Vinyl"
                DataPreprocessingReady(config['root'], config['src_tgt_pairs'], task, ratio).main()
    # output result and para(task, model, ratio...)
    print('task:{}; model:{}; ratio:{}; epoch:{}; lr:{}; gpu:{}; seed:{};'.
          format(args.task, args.base_model, args.ratio, args.epoch, args.lr, args.gpu, args.seed))

    # arg.process_data_mid和 arg.process_data_ready，則Run(config).main()
    # 並且issue error
    if not args.process_data_mid and not args.process_data_ready:
        Run(config).main()

```

初始化


root:
use_cuda: 0
root: "./data/"
src_tgt_pairs:
1:
src: "Movies_and_TV"
tgt: "CDs_and_Vinyl"
uid: 181187
iid: 114495
batchsize_src: 256
batchsize_tgt: 256
batchsize_meta: 128
batchsize_map: 64
batchsize_test: 128
2:
3:
emb_dim: 10
meta_dim: 50
num_fields: 2
wd: 0
Filter…
setting hyper-parameters

```python
# 準備訓練要用到的參數，如：seed, lr, epoch, model_base
# 將paramter記錄在config(未寫回config.json，因為會更改參數)
def prepare(config_path):
    # argparse：命令列剖析模組
    parser = argparse.ArgumentParser()
    # add_argument：加入選項參數
    # 使用方法：entry.py --arg_name parameter
    # 先加入命令，後加入if實作命令
    parser.add_argument('--process_data_mid', default=0)
    parser.add_argument('--process_data_ready', default=0)
    parser.add_argument('--task', default='1')
    parser.add_argument('--base_model', default='MF')
    parser.add_argument('--seed', type=int, default=2020)
    parser.add_argument('--ratio', default=[0.8, 0.2])
    parser.add_argument('--gpu', default='0')
    parser.add_argument('--epoch', type=int, default=10)
    parser.add_argument('--lr', type=float, default=0.01)
    args = parser.parse_args()

    # assign seed
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)
```

增加命令選項

```
35      with open(config_path, 'r') as f:
36
37          config = json.load(f)
38          # 把arg接到的parameter assign to config
39          config['base_model'] = args.base_model
40          config['task'] = args.task
41          config['ratio'] = args.ratio
42          config['epoch'] = args.epoch
43          config['lr'] = args.lr
44          '''
45          印出來trace
46          import pprint
47          pprint.pprint(config)
48          '''
49      return args, config
50
```

接收並setting hyper-parameter

```python
if __name__ == '__main__':
    config_path = 'config.json'
    args, config = prepare(config_path)
    # assign gpu
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    # 指令para --process_data_mid=True，則執行以下
    if args.process_data_mid:
        for dealing in ['Books', 'CDs_and_Vinyl', 'Movies_and_TV']:
            DataPreprocessingMid(config['root'], dealing).main()
    # 指令para --process_data_ready=True，則執行以下
    if args.process_data_ready:
        # 實驗採3個冷啟動比例
        for ratio in [[0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]:
            # 實驗有3個task
            for task in ['1', '2', '3']:
                # task_1：src:"Movies_and_TV"; tgt:"CDs_and_Vinyl"
                # task_2：src:"Books"        ; tgt:"Movies_and_TV"
                # task_3：src:"Books"        ; tgt:"CDs_and_Vinyl"
                DataPreprocessingReady(config['root'], config['src_tgt_pairs'], task, ratio).main()
    # output result and para(task, model, ratio...)
    print('task:{}; model:{}; ratio:{}; epoch:{}; lr:{}; gpu:{}; seed:{};'.
          format(args.task, args.base_model, args.ratio, args.epoch, args.lr, args.gpu, args.seed))

    # arg.process_data_mid和 arg.process_data_ready，則Run(config).main()
    # 並且issue error
    if not args.process_data_mid and not args.process_data_ready:
        Run(config).main()

```

Preprocessing(json_to _csv)

```python
if __name__ == '__main__':
    config_path = 'config.json'
    args, config = prepare(config_path)
    # assign gpu
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    # 指令para --process_data_mid=True，則執行以下
    if args.process_data_mid:
        for dealing in ['Books', 'CDs_and_Vinyl', 'Movies_and_TV']:
            DataPreprocessingMid(config['root'], dealing).main()
    # 指令para --process_data_ready=True，則執行以下
    if args.process_data_ready:
        # 實驗採3個冷啟動比例
        for ratio in [[0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]:
            # 實驗有3個task
            for task in ['1', '2', '3']:
                # task_1：src:"Movies_and_TV"; tgt:"CDs_and_Vinyl"
                # task_2：src:"Books"        ; tgt:"Movies_and_TV"
                # task_3：src:"Books"        ; tgt:"CDs_and_Vinyl"
                DataPreprocessingReady(config['root'], config['src_tgt_pairs'], task, ratio).main()
    # output result and para(task, model, ratio...)
    print('task:{}; model:{}; ratio:{}; epoch:{}; lr:{}; gpu:{}; seed:{};'.
          format(args.task, args.base_model, args.ratio, args.epoch, args.lr, args.gpu, args.seed))

    # arg.process_data_mid和 arg.process_data_ready，則Run(config).main()
    # 並且issue error
    if not args.process_data_mid and not args.process_data_ready:
        Run(config).main()

```

Preprocessing(split data)

```python
if __name__ == '__main__':
    config_path = 'config.json'
    args, config = prepare(config_path)
    # assign gpu
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    # 指令para --process_data_mid=True，則執行以下
    if args.process_data_mid:
        for dealing in ['Books', 'CDs_and_Vinyl', 'Movies_and_TV']:
            DataPreprocessingMid(config['root'], dealing).main()
    # 指令para --process_data_ready=True，則執行以下
    if args.process_data_ready:
        # 實驗採3個冷啟動比例
        for ratio in [[0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]:
            # 實驗有3個task
            for task in ['1', '2', '3']:
                # task_1：src:"Movies_and_TV"; tgt:"CDs_and_Vinyl"
                # task_2：src:"Books"        ; tgt:"Movies_and_TV"
                # task_3：src:"Books"        ; tgt:"CDs_and_Vinyl"
                DataPreprocessingReady(config['root'], config['src_tgt_pairs'], task, ratio).main()
    # output result and para(task, model, ratio...)
    print('task:{}; model:{}; ratio:{}; epoch:{}; lr:{}; gpu:{}; seed:{};'.
          format(args.task, args.base_model, args.ratio, args.epoch, args.lr, args.gpu, args.seed))

    # arg.process_data_mid和 arg.process_data_ready，則Run(config).main()
    # 並且issue error
    if not args.process_data_mid and not args.process_data_ready:
        Run(config).main()
```

```python
parser.add_argument('--process_data_mid', default=0)
parser.add_argument('--process_data_ready', default=0)
```

```python
if __name__ == '__main__':
    config_path = 'config.json'
    args, config = prepare(config_path)
    # assign gpu
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    # 指令para --process_data_mid=True，則執行以下
    if args.process_data_mid:
        for dealing in ['Books', 'CDs_and_Vinyl', 'Movies_and_TV']:
            DataPreprocessingMid(config['root'], dealing).main()
    # 指令para --process_data_ready=True，則執行以下
    if args.process_data_ready:
        # 實驗採3個冷啟動比例
        for ratio in [[0.8, 0.2], [0.5, 0.5], [0.2, 0.8]]:
            # 實驗有3個task
            for task in ['1', '2', '3']:
                # task_1：src:"Movies_and_TV"; tgt:"CDs_and_Vinyl"
                # task_2：src:"Books"        ; tgt:"Movies_and_TV"
                # task_3：src:"Books"        ; tgt:"CDs_and_Vinyl"
                DataPreprocessingReady(config['root'], config['src_tgt_pairs'], task, ratio).main()
    # output result and para(task, model, ratio...)
    print('task:{}; model:{}; ratio:{}; epoch:{}; lr:{}; gpu:{}; seed:{};'.
          format(args.task, args.base_model, args.ratio, args.epoch, args.lr, args.gpu, args.seed))

    # arg.process_data_mid和 arg.process_data_ready，則Run(config).main()
    # 並且issue error
    if not args.process_data_mid and not args.process_data_ready:
        Run(config).main()

```

Run

```
    # use config to init
    def __init__(self,
                 config
                 ):
        self.use_cuda = config['use_cuda']
        self.base_model = config['base_model']
        self.root = config['root']
        self.ratio = config['ratio']
        self.task = config['task']
        self.src = config['src_tgt_pairs'][self.task]['src']
        self.tgt = config['src_tgt_pairs'][self.task]['tgt']
        self.uid_all = config['src_tgt_pairs'][self.task]['uid']
        self.iid_all = config['src_tgt_pairs'][self.task]['iid']
        self.batchsize_src = config['src_tgt_pairs'][self.task]['batchsize_src']
        self.batchsize_tgt = config['src_tgt_pairs'][self.task]['batchsize_tgt']
        self.batchsize_meta = config['src_tgt_pairs'][self.task]['batchsize_meta']
        self.batchsize_map = config['src_tgt_pairs'][self.task]['batchsize_map']
        self.batchsize_test = config['src_tgt_pairs'][self.task]['batchsize_test']
        self.batchsize_aug = self.batchsize_src

        self.epoch = config['epoch']
        self.emb_dim = config['emb_dim']
        self.meta_dim = config['meta_dim']
        self.num_fields = config['num_fields']
        self.lr = config['lr']
        # weight dacay：抑制更新參數的幅度
        self.wd = config['wd']

```

Setting Hyper-Parameters

```
39          self.input_root = self.root + 'ready/_' + str(int(self.ratio[0] * 10)) + '_' + str(int(self.ratio[1] *
    10)) + \
40              '/tgt_' + self.tgt + '_src_' + self.src
41          self.src_path = self.input_root + '/train_src.csv'
42          self.tgt_path = self.input_root + '/train_tgt.csv'
43          self.meta_path = self.input_root + '/train_meta.csv'
44          self.test_path = self.input_root + '/test.csv'
45
46          self.results = {'tgt_mae': 10, 'tgt_rmse': 10,
47                          'aug_mae': 10, 'aug_rmse': 10,
48                          'emcdr_mae': 10, 'emcdr_rmse': 10,
49                          'ptupcdr_mae': 10, 'ptupcdr_rmse': 10}
50
```

Setting File Path

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
189     # choose model and set hyper para
190     # put model into gpu
191     def get_model(self):
192         if self.base_model == 'MF':
193             model = MFBasedModel(self.uid_all, self.iid_all, self.num_fields, self.emb_dim, self.meta_dim)
194         elif self.base_model == 'DNN':
195             model = DNNBasedModel(self.uid_all, self.iid_all, self.num_fields, self.emb_dim, self.meta_dim)
196         elif self.base_model == 'GMF':
197             model = GMFBasedModel(self.uid_all, self.iid_all, self.num_fields, self.emb_dim, self.meta_dim)
198         else:
199             raise ValueError('Unknown base model: ' + self.base_model)
200         return model.cuda() if self.use_cuda else model
201
```

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
    def get_data(self):
        print('========Reading data========')
        # retrieve a data_loader
        data_src = self.read_log_data(self.src_path, self.batchsize_src)
        print('src {} iter / batchsize = {} '.format(len(data_src), self.batchsize_src))

        # retrieve a data_loader
        data_tgt = self.read_log_data(self.tgt_path, self.batchsize_tgt)
        print('tgt {} iter / batchsize = {} '.format(len(data_tgt), self.batchsize_tgt))

        # retrieve a data_loader
        data_meta = self.read_log_data(self.meta_path, self.batchsize_meta, history=True)
        print('meta {} iter / batchsize = {} '.format(len(data_meta), self.batchsize_meta))

        # retrieve a data_loader of unique_uid data
        data_map = self.read_map_data()
        print('map {} iter / batchsize = {} '.format(len(data_map), self.batchsize_map))

        # retrieve a  concat(src,tgt) feature and label
        data_aug = self.read_aug_data()
        print('aug {} iter / batchsize = {} '.format(len(data_aug), self.batchsize_aug))

        # retrieve a data_loader
        data_test = self.read_log_data(self.test_path, self.batchsize_test, history=True)
        print('test {} iter / batchsize = {} '.format(len(data_test), self.batchsize_test))

        return data_src, data_tgt, data_meta, data_map, data_aug, data_test

```

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
202     # define the optimization algo
203     # return optimizer
204     def get_optimizer(self, model):
205         # weight_decay：權值衰減
206         optimizer_src = torch.optim.Adam(params=model.src_model.parameters(), lr=self.lr, weight_decay=self.wd)
207         optimizer_tgt = torch.optim.Adam(params=model.tgt_model.parameters(), lr=self.lr, weight_decay=self.wd)
208         optimizer_meta = torch.optim.Adam(params=model.meta_net.parameters(), lr=self.lr, weight_decay=self.wd)
209         optimizer_aug = torch.optim.Adam(params=model.aug_model.parameters(), lr=self.lr, weight_decay=self.wd)
210         optimizer_map = torch.optim.Adam(params=model.mapping.parameters(), lr=self.lr, weight_decay=self.wd)
211         return optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map
212
```

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

```
262     # target domain predict target
263     def TgtOnly(self, model, data_tgt, data_test, criterion, optimizer):
264         print('=========TgtOnly========')
265         # train i_th epoch
266         for i in range(self.epoch):
267             self.train(data_tgt, model, criterion, optimizer, i, stage='train_tgt')
268             mae, rmse = self.eval_mae(model, data_test, stage='test_tgt')
269             self.update_results(mae, rmse, 'tgt')
270             print('MAE: {} RMSE: {}'.format(mae, rmse))
271
```

```
272     # concat domain predict target
273     # CMF : combining the data from different domains into a single domain
274     def DataAug(self, model, data_aug, data_test, criterion, optimizer):
275         print('=========DataAug========')
276         for i in range(self.epoch):
277             self.train(data_aug, model, criterion, optimizer, i, stage='train_aug')
278             mae, rmse = self.eval_mae(model, data_test, stage='test_aug')
279             self.update_results(mae, rmse, 'aug')
280             print('MAE: {} RMSE: {}'.format(mae, rmse))
281
```

```
282     # source domain (with bridge) to predict target
283     def CDR(self, model, data_src, data_map, data_meta, data_test,
284             criterion, optimizer_src, optimizer_map, optimizer_meta):
285         print('=====CDR Pretraining=====')
286         for i in range(self.epoch):
287             self.train(data_src, model, criterion, optimizer_src, i, stage='train_src')
288         # use mapped data
289         print('==========EMCDR==========')
290         for i in range(self.epoch):
291             self.train(data_map, model, criterion, optimizer_map, i, stage='train_map', mapping=True)
292             mae, rmse = self.eval_mae(model, data_test, stage='test_map')
293             self.update_results(mae, rmse, 'emcdr')
294             print('MAE: {} RMSE: {}'.format(mae, rmse))
295         # use meta-network
296         print('==========PTUPCDR==========')
297         for i in range(self.epoch):
298             self.train(data_meta, model, criterion, optimizer_meta, i, stage='train_meta')
299             mae, rmse = self.eval_mae(model, data_test, stage='test_meta')
300             self.update_results(mae, rmse, 'ptupcdr')
301             print('MAE: {} RMSE: {}'.format(mae, rmse))
```

```python
def train(self, data_loader, model, criterion, optimizer, epoch, stage, mapping=False):
    print('Training Epoch {}:'.format(epoch + 1))
    # 轉為train mode
    model.train()
    for X, y in tqdm.tqdm(data_loader, smoothing=0, mininterval=1.0):
        if mapping:
            src_emb, tgt_emb = model(X, stage)
            loss = criterion(src_emb, tgt_emb)
        else:
            # 將data傳入model進行forward propagation
            pred = model(X, stage)
            # 計算loss
            loss = criterion(pred, y.squeeze().float())
        # 清空前一次的gradient   # gradient是會被累加的
        model.zero_grad()
        # 根據loss進行back propagation，計算gradient
        loss.backward()
        # 做gradient descent：minimize cost
        optimizer.step()

```

```python
    # 評估loss，但不做梯度更新
    # stage：target_only/data_aug/CDR
    def eval_mae(self, model, data_loader, stage):
        print('Evaluating MAE:')
        # evaluate model
        model.eval()
        # init two list to save data
        targets, predicts = list(), list()
        # define loss fn_1
        loss = torch.nn.L1Loss()
        # define loss fn_2
        mse_loss = torch.nn.MSELoss()
        # with torch.no_grad()不更新網路，單純只是想看訓練成果
        with torch.no_grad():
            for X, y in tqdm.tqdm(data_loader, smoothing=0, mininterval=1.0):
                pred = model(X, stage)
                # squeeze()：把shape中維度為1的去掉 ex: (1,10,1) => (10,)
                # y.shape：(1,num) => (num,)
                # extend()：擴展原本的data
                targets.extend(y.squeeze(1).tolist())
                predicts.extend(pred.tolist())
        # put into tensor to compute loss
        targets = torch.tensor(targets).float()
        predicts = torch.tensor(predicts)
        # item()：從tensor中，提取值(一個)
        # tolist()：從tensor中，提取值(多個)
        return loss(targets, predicts).item(), torch.sqrt(mse_loss(targets, predicts)).item()
```

```python
    # 該次訓練loss降低才進行loss更新
    def update_results(self, mae, rmse, phase):
        if mae < self.results[phase + '_mae']:
            self.results[phase + '_mae'] = mae
        if rmse < self.results[phase + '_rmse']:
            self.results[phase + '_rmse'] = rmse

```

```
303     def main(self):
304         # select model base
305         model = self.get_model()
306         # to get the feature&label for training&testing
307         data_src, data_tgt, data_meta, data_map, data_aug, data_test = self.get_data()
308         # setting optimizer
309         optimizer_src, optimizer_tgt, optimizer_meta, optimizer_aug, optimizer_map = self.get_optimizer(model)
310         # define loss fn
311         criterion = torch.nn.MSELoss()
312 
313         self.TgtOnly(model, data_tgt, data_test, criterion, optimizer_tgt)
314         self.DataAug(model, data_aug, data_test, criterion, optimizer_aug)
315         self.CDR(model, data_src, data_map, data_meta, data_test,
316                  criterion, optimizer_src, optimizer_map, optimizer_meta)
317         print(self.results)
318 
```

Model & Task & Method

Model & Task

Base model：MF、GMF、DNN

Default model：MF

Task 1：src:"Movies_and_TV"
tgt:"CDs_and_Vinyl"

Method Recall

Method 1:TgtOnly

dataset:data_tgt

```
data = iter(data_tgt)
data=next(data)
data
```

```
[tensor([[531395, 371407],
         [ 49588, 426575],
         [104237, 407764],
         ...,
         [292819, 391696],
         [526207, 413474],
         [234893, 404452]])
 tensor([[5],
         [1],
         [5],
         [4], ...
```

x

y

Method 2:DataAug

dataset:data_aug

```
data = iter(data_aug)
data=next(data)
data
```

```
[tensor([[425180, 214823],
         [481117,  16437],
         [563813, 382574],
         ...,
         [ 57258, 217462],
         [137569,  60092],
         [564711,  80671]])
 tensor([[5],
         [4],
         [5],
         [4], ...
```

x

y

Method 3:EMCDR

dataset:data_map

```
[tensor([364408, 618349, 273959, 583998, 515446,  36777,    818, 551203, 177986,
         410248, 555612, 348576, 516055, 408447, 634158, 640424, 395709, 476407,
          90335, 450789, 417384, 625158, 627948, 651370, 199784, 275033, 106711,
         621408, 358198, 279081, 613760, 171501, 591534,  30217, 480342, 496777,
         289900, 654404, 251172, 448324, 126973, 488534, 148834, 188309, 282262,
         449049, 593003, 456452, 201995, 374996, 355249, 251938, 632162,  9027,
         198343, 239050, 140000, 600475, 550231, 378486, 175127, 592806, 141627,
         576313, 320390,   7625,  76089, 258530, 210046, 307675, 440334, 170018,
         446501, 227740, 345519,  24785, 489155, 540799, 506280, 187686,  74463,
          27670, 435067, 398096,  94348,  55950, 183873, 505665,  73481, 157834,
         306699, 399139, 415888, 585164, 388361, 567720, 261715,  78329, 154318,
         493518,   5188, 183295, 263162, 457179,  25825, 160237, 246181, 387601,
         568675, 211712,   9190, 486501, 412853, 370683, 291117,  20757, 292268,
         151023,  13551, 117958, 334831, 292193, 305649, 296379, 541247, 577556,
         555974, 103166]),
 tensor([3178, 2042, 2622,  307, 2151, 1103, 2689, 1608, 3025, 3126, 1909, 1530,
          405, 2445,  803, 2437, 3201, 2285,  330, 1096, 3003, 3214,  495, 2955,
         1666,   31, 1270,  890,  349,  464, 1049,  227,   64, 2589, 1173,  790,
         1162, 2124, 2887,   91, 3243, 1060, 1153, 1751,   84, 2683, 1276,  308,
         1519,  290, 1805,  311, 3032, 2120, 3237, 2988,  664, 1556, 2312, 1589,
         2930, 1609, 1921, 1134, 2135, 1360,  703, 1582,  411, 2131, 1175, 2350,
          273, 2327,   56, 2759, 2176, 2361,  628, 1230, 2612, 1491,  259, 1799,
         3053,  519, 2877, 2696,  724, 1047, 1195,  454, 2885, 2414, 3097, 1353,
         1731,  195, 2603, 2152,   88,  288, 1536,  300,  493, 2476, 2072, 2588,
          406,  144, 2659, 1235,  769,  306, 2298, 3002,  269, 1906, 2927, 3073,
         2262, 2233, 1531,  276, 2692, 1954, 1842,  901])]
```

x

y

Method 4:PTUPCDR

dataset:data_meta

```
[tensor([[495379, 379439, 270174,  ...,      0,      0,
          [455067, 397011,  73671,  ...,      0,      0,
          [ 87415, 389303, 133201,  ...,      0,      0,
          ...,
          [200071, 423889,  88036,  ..., 328965,  58215,  73
          [450789, 430616,  68651,  ..., 189326, 146900,
          [336870, 408102, 187249,  ...,      0,      0,
 tensor([[5],
         [5],
         [2],
         [5],    ...
```

x

y

Model Initialization

models.py

```python
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)
```

```python
class LookupEmbedding(torch.nn.Module):

    def __init__(self, uid_all, iid_all, emb_dim):
        super().__init__()
        self.uid_embedding = torch.nn.Embedding(uid_all, emb_dim)
        self.iid_embedding = torch.nn.Embedding(iid_all + 1, emb_dim)

    def forward(self, x):
        uid_emb = self.uid_embedding(x[:, 0].unsqueeze(1))
        iid_emb = self.iid_embedding(x[:, 1].unsqueeze(1))
        emb = torch.cat([uid_emb, iid_emb], dim=1)
        return emb
```

```python
class MetaNet(torch.nn.Module):
    def __init__(self, emb_dim, meta_dim):
        super().__init__()
        self.event_K = torch.nn.Sequential(torch.nn.Linear(emb_dim, emb_dim), torch.nn.ReLU(),
                                           torch.nn.Linear(emb_dim, 1, False))
        self.event_softmax = torch.nn.Softmax(dim=1)
        self.decoder = torch.nn.Sequential(torch.nn.Linear(emb_dim, meta_dim), torch.nn.ReLU(),
                                           torch.nn.Linear(meta_dim, emb_dim * emb_dim))

    def forward(self, emb_fea, seq_index):
        mask = (seq_index == 0).float()
        event_K = self.event_K(emb_fea)
        t = event_K - torch.unsqueeze(mask, 2) * 1e8
        att = self.event_softmax(t)
        his_fea = torch.sum(att * emb_fea, 1)
        output = self.decoder(his_fea)
        return output.squeeze(1)
```

Method 1:TgtOnly

run.py

```
def TgtOnly(self, model, data_tgt, data_test, criterion, optimizer):
    print('=========TgtOnly========')
    for i in range(self.epoch):
        self.train(data_tgt, model, criterion, optimizer, i, stage='train_tgt')
        mae, rmse = self.eval_mae(model, data_test, stage='test_tgt')
        self.update_results(mae, rmse, 'tgt')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
```

models.py

```
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)
```

```
    elif stage in ['train_tgt', 'test_tgt']:
        emb = self.tgt_model.forward(x)
        x = torch.sum(emb[:, 0, :] * emb[:, 1, :], dim=1)
        return x
```

ui * vj

$$\min_{\boldsymbol{u},\boldsymbol{v}} \frac{1}{|\mathcal{R}|} \sum_{r_{ij} \in \mathcal{R}} (r_{ij} - \boldsymbol{u}_i \boldsymbol{v}_j)^2,$$

Method 2:Data Augmentation

run.py

```
def DataAug(self, model, data_aug, data_test, criterion, optimizer):
    print('=========DataAug========')
    for i in range(self.epoch):
        self.train(data_aug, model, criterion, optimizer, i, stage='train_aug')
        mae, rmse = self.eval_mae(model, data_test, stage='test_aug')
        self.update_results(mae, rmse, 'aug')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
```

models.py

```
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)


    elif stage in ['train_aug', 'test_aug']:
        emb = self.aug_model.forward(x)
        x = torch.sum(emb[:, 0, :] * emb[:, 1, :], dim=1)
        return x
```

ui * vj

$$\min_{\boldsymbol{u},\boldsymbol{v}} \frac{1}{|\mathcal{R}|} \sum_{r_{ij} \in \mathcal{R}} (r_{ij} - \boldsymbol{u}_i \boldsymbol{v}_j)^2,$$

Method 3:EMCDR

run.py

```
print('==========EMCDR==========')
for i in range(self.epoch):
    self.train(data_map, model, criterion, optimizer_map, i, stage='train_map', mapping=True)
    mae, rmse = self.eval_mae(model, data_test, stage='test_map')
    self.update_results(mae, rmse, 'emcdr')
    print('MAE: {} RMSE: {}'.format(mae, rmse))
```

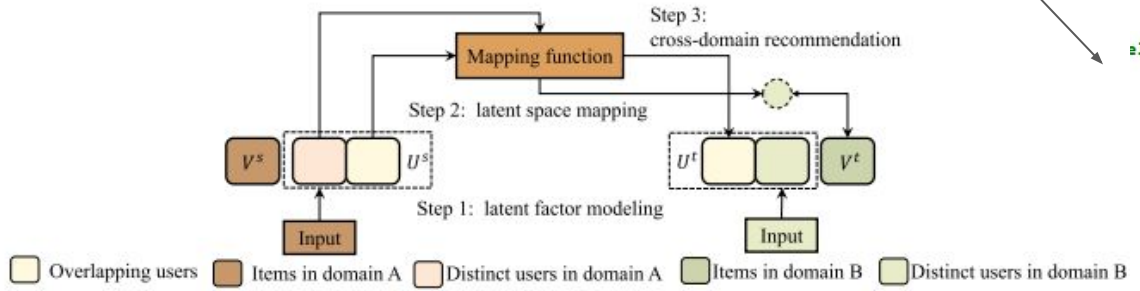


Fig. 7. The schematic diagram of embedding and mapping.

models.py

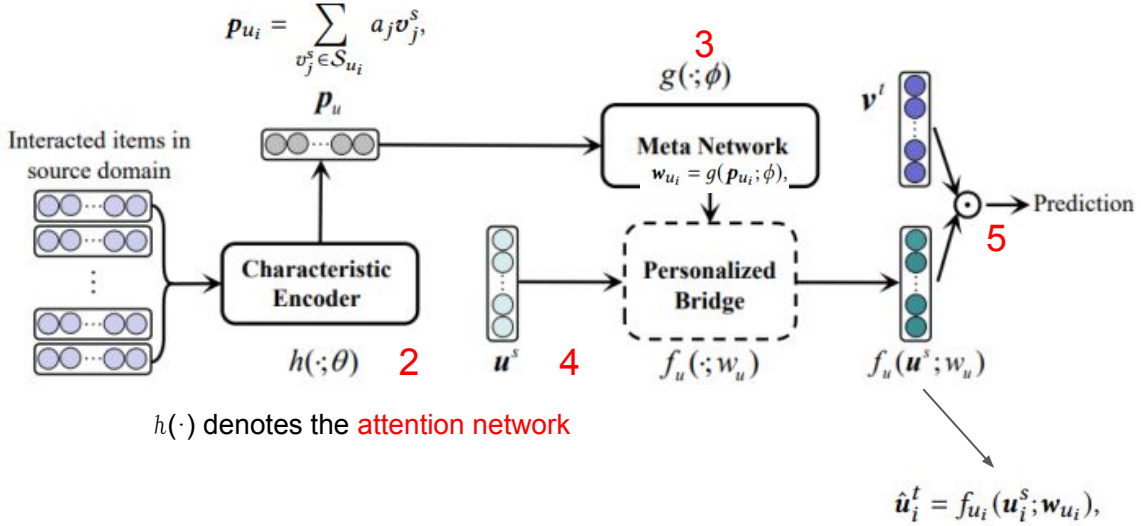
```
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)

    elif stage == 'train_map':
        src_emb = self.src_model.uid_embedding(x.unsqueeze(1)).squeeze()
        src_emb = self.mapping.forward(src_emb)
        tgt_emb = self.tgt_model.uid_embedding(x.unsqueeze(1)).squeeze()
        return src_emb, tgt_emb
```

Method 4:PTUPCDR

Algorithm 1 Personalized Transfer of User Preferences for CDR (PTUPCDR)

Input: $\mathcal{U}^s, \mathcal{U}^t, \mathcal{V}^s, \mathcal{V}^t, \mathcal{U}^o, \mathcal{R}^s, \mathcal{R}^t$
Input: Meta network g_ϕ.
Input: Characteristic encoder h_θ.
Pre-training Stage:
1. Learning a source model which contains $\boldsymbol{u}^s, \boldsymbol{v}^s$.
2. Learning a target model which contains $\boldsymbol{u}^t, \boldsymbol{v}^t$.
Meta Stage:
3. Learning a characteristic encoder h_θ and a meta network g_ϕ by minimizing Equation (7).
Initialization Stage:
4. For a cold-start user u^t in the target domain, we use the transformed embedding $f_{u_i}(\boldsymbol{u}_i^s; \boldsymbol{w}_{u_i})$ as the user's initialized embedding in the target domain.

source domain by $\mathcal{S}_{u_i} = \{v_{t_1}^s, v_{t_2}^s, \cdots, v_{t_n}^s\}$, where n denotes the number of interacted items and $v_{t_n}^s$ denotes the interacted item in the source domain at timestamp t_n.



$h(\cdot)$ denotes the attention network

PTUPCDR : Pretraining Stage

Algorithm 1 Personalized Transfer of User Preferences for CDR (PTUPCDR)

Input: $\mathcal{U}^s, \mathcal{U}^t, \mathcal{V}^s, \mathcal{V}^t, \mathcal{U}^o, \mathcal{R}^s, \mathcal{R}^t$
Input: Meta network g_ϕ.
Input: Characteristic encoder h_θ.
Pre-training Stage:
1. Learning a source model which contains $\boldsymbol{u}^s, \boldsymbol{v}^s$.
2. Learning a target model which contains $\boldsymbol{u}^t, \boldsymbol{v}^t$.
Meta Stage:
3. Learning a characteristic encoder h_θ and a meta network g_ϕ by minimizing Equation (7).
Initialization Stage:
4. For a cold-start user u^t in the target domain, we use the transformed embedding $f_{u_i}(\boldsymbol{u}_i^s; \boldsymbol{w}_{u_i})$ as the user's initialized embedding in the target domain.

source domain by $\mathcal{S}_{u_i} = \{v_{t_1}^s, v_{t_2}^s, \cdots, v_{t_n}^s\}$, where n denotes the number of interacted items and $v_{t_n}^s$ denotes the interacted item in the source domain at timestamp t_n.

run.py

```
def CDR(self, model, data_src, data_map, data_meta, data_test,
        criterion, optimizer_src, optimizer_map, optimizer_meta):
    print('=====CDR Pretraining=====')#optimize the embedding from source domain
    for i in range(self.epoch):
        self.train(data_src, model, criterion, optimizer_src, i, stage='train_src')
    print('==========EMCDR==========')
    for i in range(self.epoch):
        self.train(data_map, model, criterion, optimizer_map, i, stage='train_map', mapping=True)
        mae, rmse = self.eval_mae(model, data_test, stage='test_map')
        self.update_results(mae, rmse, 'emcdr')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
    print('==========PTUPCDR==========')
    for i in range(self.epoch):
        self.train(data_meta, model, criterion, optimizer_meta, i, stage='train_meta')
        mae, rmse = self.eval_mae(model, data_test, stage='test_meta')
        self.update_results(mae, rmse, 'ptupcdr')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
```

PTUPCDR : Pretraining Stage

models.py

```
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)

    def forward(self, x, stage):
        if stage == 'train_src':
            emb = self.src_model.forward(x)
            x = torch.sum(emb[:, 0, :] * emb[:, 1, :], dim=1)
```

ui * vj

```
class LookupEmbedding(torch.nn.Module):

    def __init__(self, uid_all, iid_all, emb_dim):
        super().__init__()
        self.uid_embedding = torch.nn.Embedding(uid_all, emb_dim)
        self.iid_embedding = torch.nn.Embedding(iid_all + 1, emb_dim)

    def forward(self, x):
        uid_emb = self.uid_embedding(x[:, 0].unsqueeze(1))
        iid_emb = self.iid_embedding(x[:, 1].unsqueeze(1))
        emb = torch.cat([uid_emb, iid_emb], dim=1)
        return emb
```

$$\min_{\boldsymbol{u},\boldsymbol{v}} \frac{1}{|\mathcal{R}|} \sum_{r_{ij} \in \mathcal{R}} (r_{ij} - \boldsymbol{u}_i \boldsymbol{v}_j)^2,$$

PTUPCDR : Meta Stage & Initialization Stage

Algorithm 1 Personalized Transfer of User Preferences for CDR (PTUPCDR)

Input: $\mathcal{U}^s, \mathcal{U}^t, \mathcal{V}^s, \mathcal{V}^t, \mathcal{U}^o, \mathcal{R}^s, \mathcal{R}^t$
Input: Meta network g_ϕ.
Input: Characteristic encoder h_θ.
Pre-training Stage:
1. Learning a source model which contains $\boldsymbol{u}^s, \boldsymbol{v}^s$.
2. Learning a target model which contains $\boldsymbol{u}^t, \boldsymbol{v}^t$.
Meta Stage:
3. Learning a characteristic encoder h_θ and a meta network g_ϕ by minimizing Equation (7).
Initialization Stage:
4. For a cold-start user u^t in the target domain, we use the transformed embedding $f_{u_i}(\boldsymbol{u}_i^s; \boldsymbol{w}_{u_i})$ as the user's initialized embedding in the target domain.

source domain by $\mathcal{S}_{u_i} = \{v_{t_1}^s, v_{t_2}^s, \cdots, v_{t_n}^s\}$, where n denotes the number of interacted items and $v_{t_n}^s$ denotes the interacted item in the source domain at timestamp t_n.

run.py

```
def CDR(self, model, data_src, data_map, data_meta, data_test,
        criterion, optimizer_src, optimizer_map, optimizer_meta):
    print('=====CDR Pretraining=====')#optimize the embedding from source domain
    for i in range(self.epoch):
        self.train(data_src, model, criterion, optimizer_src, i, stage='train_src')
    print('==========EMCDR==========')
    for i in range(self.epoch):
        self.train(data_map, model, criterion, optimizer_map, i, stage='train_map', mapping=True)
        mae, rmse = self.eval_mae(model, data_test, stage='test_map')
        self.update_results(mae, rmse, 'emcdr')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
    print('==========PTUPCDR==========')
    for i in range(self.epoch):
        self.train(data_meta, model, criterion, optimizer_meta, i, stage='train_meta')
        mae, rmse = self.eval_mae(model, data_test, stage='test_meta')
        self.update_results(mae, rmse, 'ptupcdr')
        print('MAE: {} RMSE: {}'.format(mae, rmse))
```

PTUPCDR : Meta Stage & Initialization Stage

models.py

```python
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)


    def forward(self, x, stage):

        elif stage in ['train_meta', 'test_meta']:
            iid_emb = self.tgt_model.iid_embedding(x[:, 1].unsqueeze(1))
            uid_emb_src = self.src_model.uid_embedding(x[:, 0].unsqueeze(1))
            ufea = self.src_model.iid_embedding(x[:, 2:])
            mapping = self.meta_net.forward(ufea, x[:, 2:]).view(-1, self.emb_dim, self.emb_dim)
            uid_emb = torch.bmm(uid_emb_src, mapping)
            emb = torch.cat([uid_emb, iid_emb], 1)
            output = torch.sum(emb[:, 0, :] * emb[:, 1, :], dim=1)
            return output
```

```python
class LookupEmbedding(torch.nn.Module):

    def __init__(self, uid_all, iid_all, emb_dim):
        super().__init__()
        self.uid_embedding = torch.nn.Embedding(uid_all, emb_dim)
        self.iid_embedding = torch.nn.Embedding(iid_all + 1, emb_dim)

    def forward(self, x):
        uid_emb = self.uid_embedding(x[:, 0].unsqueeze(1))
        iid_emb = self.iid_embedding(x[:, 1].unsqueeze(1))
        emb = torch.cat([uid_emb, iid_emb], dim=1)
        return emb
```

uid	iid	pos_seq
x[: , 0]	x[: , 1]	x[: , 2 :]

```
[tensor([[495379, 379439, 270174, ...,      0,      0,      0],
         [455067, 397011,  73671, ...,      0,      0,      0],
         [ 87415, 389303, 133201, ...,      0,      0,      0],
         ...,
         [200071, 423889,  88036, ..., 328965,  58215,  73458],
         [450789, 430616,  68651, ..., 189326, 146900,      0],
         [336870, 408102, 187249, ...,      0,      0,      0]]),
 tensor([[5],
         [5],
         [2],
         [5],
```

PTUPCDR : Meta Stage & Initialization Stage

models.py

```python
class MFBasedModel(torch.nn.Module):# self.uid_all, self.iid_all, self.num_
    def __init__(self, uid_all, iid_all, num_fields, emb_dim, meta_dim_0):
        super().__init__()
        self.num_fields = num_fields
        self.emb_dim = emb_dim
        self.src_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.tgt_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.aug_model = LookupEmbedding(uid_all, iid_all, emb_dim)
        self.meta_net = MetaNet(emb_dim, meta_dim_0)
        self.mapping = torch.nn.Linear(emb_dim, emb_dim, False)
```

```python
    def forward(self, x, stage):

        elif stage in ['train_meta', 'test_meta']:
            iid_emb = self.tgt_model.iid_embedding(x[:, 1].unsqueeze(1))
            uid_emb_src = self.src_model.uid_embedding(x[:, 0].unsqueeze(1))
            ufea = self.src_model.iid_embedding(x[:, 2:])
            mapping = self.meta_net.forward(ufea, x[:, 2:]).view(-1, self.emb_dim, self.emb_dim)
            uid_emb = torch.bmm(uid_emb_src, mapping)
            emb = torch.cat([uid_emb, iid_emb], 1)
            output = torch.sum(emb[:, 0, :] * emb[:, 1, :], dim=1)
            return output
```

predicted ui * vj

```python
class MetaNet(torch.nn.Module):
    def __init__(self, emb_dim, meta_dim):
        super().__init__()
        self.event_K = torch.nn.Sequential(torch.nn.Linear(emb_dim, emb_dim), torch.nn.ReLU(),
                                           torch.nn.Linear(emb_dim, 1, False))
        self.event_softmax = torch.nn.Softmax(dim=1)
        self.decoder = torch.nn.Sequential(torch.nn.Linear(emb_dim, meta_dim), torch.nn.ReLU(),
                                           torch.nn.Linear(meta_dim, emb_dim * emb_dim))

    def forward(self, emb_fea, seq_index):
        mask = (seq_index == 0).float()
        event_K = self.event_K(emb_fea)
        t = event_K - torch.unsqueeze(mask, 2) * 1e8
        att = self.event_softmax(t)
        his_fea = torch.sum(att * emb_fea, 1)
        output = self.decoder(his_fea)
        return output.squeeze(1)
```

$$a'_j = h(\boldsymbol{v}_j; \theta),$$

$$a_j = \frac{\exp(a'_j)}{\sum_{v_l^s \in \mathcal{S}_{u_i}} \exp(a'_l)},$$

$$\boldsymbol{p}_{u_i} = \sum_{v_j^s \in \mathcal{S}_{u_i}} a_j \boldsymbol{v}_j^s,$$

$$\boldsymbol{w}_{u_i} = g(\boldsymbol{p}_{u_i}; \phi),$$

With the personalized bridge function, we can obtain the personalized transformed user's embeddings:

$$\hat{\boldsymbol{u}}_i^t = f_{u_i}(\boldsymbol{u}_i^s; \boldsymbol{w}_{u_i}), \tag{5}$$

$$\min_{\theta,\phi} \frac{1}{|\mathcal{R}_o^t|} \sum_{r_{ij} \in \mathcal{R}_o^t} (r_{ij} - f_{u_i}(\boldsymbol{u}_i^s; \boldsymbol{w}_{u_i}) \boldsymbol{v}_j)^2, \tag{7}$$

where $\mathcal{R}_o^t = \{r_{ij} | u_i \in \mathcal{U}^o, v_j \in \mathcal{V}^t\}$ denotes the interactions of overlapping users in the target domain.