ML-1M DL, consider sex

In [1]:
```
# import io
# import os
import math
import copy
import pickle
# import zipfile
# from textwrap import wrap
from pathlib import Path
from itertools import zip_longest
from collections import defaultdict
# from urllib.error import URLError
# from urllib.request import urlopen

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold

import torch
from torch import nn
from torch import optim
from torch.nn import functional as F
from torch.optim.lr_scheduler import _LRScheduler

from time import time
from collections import defaultdict

%matplotlib inline
```

In [2]:
```
def set_random_seed(state=1):
    gens = (np.random.seed, torch.manual_seed, torch.cuda.manual_seed)
    for set_state in gens:
        set_state(state)


RANDOM_STATE = 1
set_random_seed(RANDOM_STATE)
```

In [3]:
```
# load preprocessed df
df = pd.read_csv("ml-1m_dl.csv")
print(df.shape)
df.head()
```

```
(1000209, 9)
```

Out[3]:

	movie_id	movie_title	user_id	age	sex	occupation	rating	sex_index	age_index
0	1	Toy Story (1995)	1	1	F	10	5	0	0
1	48	Pocahontas (1995)	1	1	F	10	5	0	0
2	150	Apollo 13 (1995)	1	1	F	10	5	0	0
3	260	Star Wars: Episode IV - A New Hope (1977)	1	1	F	10	4	0	0
4	527	Schindler's List (1993)	1	1	F	10	5	0	0

In [4]:
```
# load dataset
datasets = pickle.load(open('ml-1m_dl.pkl','rb'))
datasets['val'][1]
```

Out[4]:
```
630120    4.0
229398    5.0
758377    3.0
159240    5.0
254252    4.0
         ...
875199    4.0
743921    4.0
527163    4.0
623363    3.0
120098    3.0
Name: rating, Length: 200042, dtype: float32
```

In [5]:
```
n_users = 6040
n_movies = 3706
dataset_sizes = {'train': 800167, 'val': 200042}
```

Define the network

In [6]:
```
# only consider sex
class EmbeddingNetGender(nn.Module):
    def __init__(self, n_users, n_movies, n_factors=50, embedding_dropout=0.02, hidden=10, dropouts=0.2, g_fa
ctor=25):
        super().__init__()
        hidden = get_list(hidden)
        dropouts = get_list(dropouts)
        n_last = hidden[-1]
        def gen_layers(n_in):
            """
            A generator that yields a sequence of hidden layers and
            their activations/dropouts.

            Note that the function captures `hidden` and `dropouts`
            values from the outer scope.
            """
            nonlocal hidden, dropouts
            assert len(dropouts) <= len(hidden)
            for n_out, rate in zip_longest(hidden, dropouts):
                yield nn.Linear(n_in, n_out)
                yield nn.ReLU()
                if rate is not None and rate > 0.:
                    yield nn.Dropout(rate)
                n_in = n_out


        self.u = nn.Embedding(n_users+1 , n_factors) # hard code
        self.m = nn.Embedding(4000, n_factors)  # hardcode

        self.g_factor = g_factor
        self.g = nn.Embedding(2, self.g_factor)


        self.drop = nn.Dropout(embedding_dropout)
        self.hidden = nn.Sequential(*list(gen_layers(n_factors * 2 + self.g_factor)))
        self.fc = nn.Linear(n_last, 1)
        self._init()
    def forward(self, users, movies, genders, minmax=None):
        uu = self.u(users)
        gg = self.g(genders)
        mm = self.m(movies)
        features = torch.cat([uu, gg, mm], dim=1)
        x = self.drop(features)
        x = self.hidden(x)
        out = torch.sigmoid(self.fc(x))
        if minmax is not None:
            min_rating, max_rating = minmax
            out = out*(max_rating - min_rating + 1) + min_rating - 0.5
        return out

    def _init(self):
        def init(m):
            if type(m) == nn.Linear:
                torch.nn.init.xavier_uniform_(m.weight)
                m.bias.data.fill_(0.01)
        self.u.weight.data.uniform_(-0.05, 0.05)
        self.m.weight.data.uniform_(-0.05, 0.05)
        self.hidden.apply(init)
        init(self.fc)

def get_list(n):
    if isinstance(n, (int, float)):
        return [n]
    elif hasattr(n, '__iter__'):
        return list(n)
    raise TypeError('layers configuraiton should be a single number or a list of numbers')
```

In [7]:
```
# test
testnet = EmbeddingNetGender(n_users, n_movies, n_factors=150, hidden=100, dropouts=0.5, g_factor=25)
print(testnet)
testnet = EmbeddingNetGender(n_users, n_movies, n_factors=150, hidden=[100, 200, 300], dropouts=[0.25, 0.5],
g_factor=25)
print(testnet)
```

```
EmbeddingNetGender(
  (u): Embedding(6041, 150)
  (m): Embedding(4000, 150)
  (g): Embedding(2, 25)
  (drop): Dropout(p=0.02, inplace=False)
  (hidden): Sequential(
    (0): Linear(in_features=325, out_features=100, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
  )
  (fc): Linear(in_features=100, out_features=1, bias=True)
)
EmbeddingNetGender(
  (u): Embedding(6041, 150)
  (m): Embedding(4000, 150)
  (g): Embedding(2, 25)
  (drop): Dropout(p=0.02, inplace=False)
  (hidden): Sequential(
    (0): Linear(in_features=325, out_features=100, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.25, inplace=False)
    (3): Linear(in_features=100, out_features=200, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=200, out_features=300, bias=True)
    (7): ReLU()
  )
  (fc): Linear(in_features=300, out_features=1, bias=True)
)
```

Helper functions

In [8]:
```
# batch-wise data iterator
class ReviewsIterator:
    def __init__(self, X, y, batch_size=32, shuffle=True):
        X, y = np.asarray(X), np.asarray(y)

        if shuffle:
            index = np.random.permutation(X.shape[0])
            X, y = X[index], y[index]

        self.X = X
        self.y = y
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.n_batches = int(math.ceil(X.shape[0] // batch_size))
        self._current = 0

    def __iter__(self):
        return self

    def __next__(self):
        return self.next()

    def next(self):
        if self._current >= self.n_batches:
            raise StopIteration()
        k = self._current
        self._current += 1
        bs = self.batch_size
        return self.X[k*bs:(k + 1)*bs], self.y[k*bs:(k + 1)*bs]


def batches(X, y, bs=32, shuffle=True):
    for xb, yb in ReviewsIterator(X, y, bs, shuffle):
        xb = torch.LongTensor(xb)
        yb = torch.FloatTensor(yb)
        yield xb, yb.view(-1, 1)
```

In [9]:
```
class CyclicLR(_LRScheduler):
    def __init__(self, optimizer, schedule, last_epoch=-1):
        assert callable(schedule)
        self.schedule = schedule
        super().__init__(optimizer, last_epoch)
    def get_lr(self):
        return [self.schedule(self.last_epoch, lr) for lr in self.base_lrs]

def triangular(step_size, max_lr, method='triangular', gamma=0.99):

    def scheduler(epoch, base_lr):
        period = 2 * step_size
        cycle = math.floor(1 + epoch/period)
        x = abs(epoch/step_size - 2*cycle + 1)
        delta = (max_lr - base_lr)*max(0, (1 - x))

        if method == 'triangular':
            pass  # we've already done
        elif method == 'triangular2':
            delta /= float(2 ** (cycle - 1))
        elif method == 'exp_range':
            delta *= (gamma**epoch)
        else:
            raise ValueError('unexpected method: %s' % method)

        return base_lr + delta

    return scheduler

def cosine(t_max, eta_min=0):

    def scheduler(epoch, base_lr):
        t = epoch % t_max
        return eta_min + (base_lr - eta_min)*(1 + math.cos(math.pi*t/t_max))/2

    return scheduler

def plot_lr(schedule, label):
    ts = list(range(1000))
    y = [schedule(t, 0.001) for t in ts]
    plt.plot(ts, y, label=label)
```

In [10]:
```
def train_model(datasets, model, lr, wd, bs, n_epochs, patience):
    minmax = (1.0, 5.0)
    device = torch.device('cpu')

    # Training
    no_improvements = 0
    best_loss = np.inf
    best_weights = None
    history = []
    lr_history = []

    start_time = time()
    model.to(device)
    criterion = nn.MSELoss(reduction='sum')
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=wd)
    iterations_per_epoch = int(math.ceil(dataset_sizes['train'] // bs))
    scheduler = CyclicLR(optimizer, cosine(t_max=iterations_per_epoch * 2, eta_min=lr/10))

    start_time = time()

    for epoch in range(n_epochs):
        stats = {'epoch': epoch + 1, 'total': n_epochs}

        for phase in ('train', 'val'):
            training = phase == 'train'
            running_loss = 0.0
            n_batches = 0

            for batch in batches(*datasets[phase], shuffle=training, bs=bs):
                x_batch, y_batch = [b.to(device) for b in batch] # [2000, 4], [2000, 1]
                optimizer.zero_grad()
                # compute gradients only during 'train' phase
                with torch.set_grad_enabled(training):
                    outputs = model(x_batch[:, 0], x_batch[:, 1], x_batch[:, 2], minmax)
                    loss = criterion(outputs, y_batch)
                    # don't update weights and rates when in 'val' phase
                    if training:
                        loss.backward()
                        optimizer.step()
                        scheduler.step()
                        lr_history.extend(scheduler.get_lr())
                running_loss += loss.item()

            epoch_loss = running_loss / dataset_sizes[phase]
            stats[phase] = epoch_loss

            # early stopping: save weights of the best model so far
            if phase == 'val':
                if epoch_loss < best_loss:
                    print('loss improvement on epoch: %d' % (epoch + 1))
                    best_loss = epoch_loss
                    best_weights = copy.deepcopy(model.state_dict())
                    no_improvements = 0
                else:
                    no_improvements += 1

        history.append(stats)
        cost_time = (time() - start_time) / 60.
        print('[{:03d}/{:03d}]|train {:.4f}|val {:.4f}|Time {:.2f}mins'.format(
                                                                        stats['epoch'], stats['total'],
                                                                        stats['train'], stats['val'], cost_ti
me))
        if no_improvements >= patience:
            print('early stopping after epoch {:03d}'.format(stats['epoch']))
            break
    return best_weights
```

In [16]:
```
def get_result_df(datasets, model, best_weights, bs, save_path=None):
    '''
    model_parameter1_best.weights
    '''
    minmax = (1.0, 5.0)
    device = torch.device('cpu')
    model.load_state_dict(best_weights)
    groud_truth, predictions = [], []

    val_size = len(datasets['val'][0])
    # print("Total val size: {}".format(val_size))

    with torch.no_grad():
        for batch in batches(*datasets['val'], shuffle=False, bs=bs):
            x_batch, y_batch = [b.to(device) for b in batch]
            outputs = model(x_batch[:, 0], x_batch[:, 1], x_batch[:, 2], minmax)
            groud_truth.extend(y_batch.tolist())
            predictions.extend(outputs.tolist())

        last_num = val_size % bs
        # print("Last num: {}".format(last_num))
        dataset_last = (datasets['val'][0][-last_num:], datasets['val'][1][-last_num:])
        # print("Last dataset: {}".format(len(dataset_last[0])))
        for batch in batches(*dataset_last, shuffle=False, bs=1):
            x_batch, y_batch = [b.to(device) for b in batch]
            outputs = model(x_batch[:, 0], x_batch[:, 1], x_batch[:, 2], minmax)
            groud_truth.extend(y_batch.tolist())
            predictions.extend(outputs.tolist())

    groud_truth = np.asarray(groud_truth).ravel()
    predictions = np.asarray(predictions).ravel()

    assert(predictions.shape[0] == val_size)

    final_loss = np.sqrt(np.mean((predictions - groud_truth)**2))
    print("RMSE: {:.4f}".format(final_loss))

    df_final = pd.DataFrame(datasets['val'][0])[['user_id','movie_id']]
    df_final['truth'] = datasets['val'][1]
    df_final['pred'] = predictions

    if save_path is not None:
        print("Save weight to:{}".format(save_path))
        with open(save_path, 'wb') as file:
            pickle.dump(best_weights, file)

    return df_final # note that here the sex and age is not included
```

In [12]:
```
def get_precision_recall(df_final, k=10, threshold=3.5):
    # map prediction to each user --> similar to top n
    # {id:(pred, truth)}
    user_pred_truth = defaultdict(list)
    for row in df_final.itertuples():
        _, user_id, movie_id, truth, pred = row
        user_pred_truth[user_id].append((pred, truth))

    precisions = dict()
    recalls = dict()

    for user_id, user_ratings in user_pred_truth.items():
        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Number of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                              for (est, true_r) in user_ratings[:k])

        # Precision@K: Proportion of recommended items that are relevant
        precisions[user_id] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1

        # Recall@K: Proportion of relevant items that are recommended
        recalls[user_id] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

    # mean precision and recall
    mean_precision = sum(prec for prec in precisions.values()) / len(precisions)
    mean_recall = sum(rec for rec in recalls.values()) / len(recalls)
    print("Prec10 {:.4f}|Rec10 {:.4f}".format(mean_precision, mean_recall))

# get topn
def get_top_n(df_final, n=10):
    """
    key: user_id
    value: his top 10 highest movies as well as ratings
    """
    # map predictions to each user
    top_n = defaultdict(list)
    for row in df_final.itertuples():
        _, user_id, movie_id, truth, pred = row
        top_n[user_id].append((movie_id, pred))

    # sort the pred for each user
    for user_id, pred_ratings in top_n.items():
        pred_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[user_id] = pred_ratings[:n]
    return top_n

# i = 0
# top_n = get_top_n(df_final, n=10)
# for user_id, pred_ratings in top_n.items():
#     print("User id: {}".format(user_id))
#     for (movie_id, rating) in pred_ratings:
#         print("Movie {:<5d}|Rating {:.2f}".format(movie_id, rating))
#     print("----- -----")
#     i += 1
#     if i > 0:
#         break

def get_train_pred_top10(df, df_final, datasets, user_id=1635, n=10):
    """
    df: the original df --> contain movie name
    df_final: the final df with predicted ratings
    datasets: the datasets with training and testing datasets
    user_id: the user id to be queried
    n: top_n
    """
    # step 1, get top n from df_final
    top_n = get_top_n(df_final, n)
    assert(user_id in top_n), "user_id {} is not in testing data, try another user such as 1635".format(user_
id)
    pred_ratings = top_n[user_id]

    # step 2: user information
    user = df[df['user_id'] == user_id]
    age = list(set(user['age']))[0]
    sex = list(set(user['sex']))[0]
    info = "User {}, age {}, {}, ".format(user_id, age, sex)

    # step 2, build df_train
    df_train = pd.DataFrame(datasets['train'][0])
    df_train['rating'] = datasets['train'][1]
```

```
    # step 3, find all movies user_id has been rated 5
    # df_refined = df_train[(df_train['user_id'] == user_id) & (df_train['rating'] == 5)]
    df_refined = df_train[df_train['user_id'] == user_id]
    movie_id_sets_train = set(df_refined['movie_id'])
    info = "{} has rated {} movies in training set\n".format(info, len(movie_id_sets_train))
    print(info)

    # step 4: get the top n
    print("===== ===== ===== ===== ===== =====")
    print("\nTop {} recommendations\n".format(n))
    for (movie_id, rating) in pred_ratings:
        movie_name = list(set(df[df['movie_id'] == movie_id]['movie_title']))[0]
        info = "ID {:<4d}|Rating {:2f}|{}".format(movie_id, rating, movie_name)
        if movie_id in movie_id_sets_train:
            info = "{}, but this movie has been rated during training!!!".format(info)
        print(info)
```

Now let's begin

para1

In [13]:
```
n_factors = 150
hidden = [500,500,500]
embedding_dropout = 0.05
dropouts = [0.5,0.5,0.25]
# training
lr = 1e-3
wd = 1e-5
bs =2000
n_epochs = 100
patience = 10
g_factor = 25

model = EmbeddingNetGender(n_users, n_movies,
                       n_factors=n_factors, hidden=hidden, dropouts=dropouts, g_factor=g_factor)
best_weights = train_model(datasets, model, lr, wd, bs, n_epochs, patience)
```

```
loss improvement on epoch: 1
[001/100]|train 0.9231|val 0.8216|Time 0.99mins
loss improvement on epoch: 2
[002/100]|train 0.8008|val 0.8064|Time 1.89mins
loss improvement on epoch: 3
[003/100]|train 0.8052|val 0.8025|Time 2.90mins
loss improvement on epoch: 4
[004/100]|train 0.7732|val 0.7919|Time 3.86mins
loss improvement on epoch: 5
[005/100]|train 0.7789|val 0.7845|Time 4.87mins
loss improvement on epoch: 6
[006/100]|train 0.7426|val 0.7752|Time 5.83mins
loss improvement on epoch: 7
[007/100]|train 0.7510|val 0.7732|Time 6.76mins
loss improvement on epoch: 8
[008/100]|train 0.7168|val 0.7663|Time 7.74mins
[009/100]|train 0.7309|val 0.7678|Time 8.66mins
loss improvement on epoch: 10
[010/100]|train 0.7010|val 0.7647|Time 9.66mins
[011/100]|train 0.7171|val 0.7656|Time 10.61mins
[012/100]|train 0.6874|val 0.7655|Time 11.61mins
[013/100]|train 0.7039|val 0.7671|Time 12.53mins
[014/100]|train 0.6745|val 0.7672|Time 13.48mins
[015/100]|train 0.6916|val 0.7671|Time 14.52mins
[016/100]|train 0.6611|val 0.7730|Time 15.59mins
[017/100]|train 0.6787|val 0.7735|Time 16.71mins
[018/100]|train 0.6471|val 0.7765|Time 17.63mins
[019/100]|train 0.6647|val 0.7750|Time 18.59mins
[020/100]|train 0.6332|val 0.7840|Time 19.57mins
early stopping after epoch 020
```

In [17]:
```
df_final = get_result_df(datasets, model, best_weights, bs, save_path='withSex_noAge_para1.weights')
get_precision_recall(df_final, k=10, threshold=3.5)
```

```
RMSE: 0.8747
Save weight to:withSex_noAge_para1.weights
Prec10 0.7987|Rec10 0.5624
```

para 2

In [19]:
```
n_factors = 150
hidden = [500] * 3
g_factor = 50

embedding_dropout = 0.05
dropouts = [0.5,0.5,0.25]
# training
lr = 1e-3
wd = 1e-5
bs =2000
n_epochs = 100
patience = 10

model = EmbeddingNetGender(n_users, n_movies,
                        n_factors=n_factors, hidden=hidden, dropouts=dropouts, g_factor=g_factor)
best_weights = train_model(datasets, model, lr, wd, bs, n_epochs, patience)
df_final = get_result_df(datasets, model, best_weights, bs, save_path='withSex_noAge_para2.weights')
get_precision_recall(df_final, k=10, threshold=3.5)
```

```
loss improvement on epoch: 1
[001/100]|train 0.9426|val 0.8267|Time 0.51mins
loss improvement on epoch: 2
[002/100]|train 0.8052|val 0.8088|Time 1.02mins
loss improvement on epoch: 3
[003/100]|train 0.8096|val 0.8069|Time 1.54mins
loss improvement on epoch: 4
[004/100]|train 0.7790|val 0.7964|Time 2.11mins
loss improvement on epoch: 5
[005/100]|train 0.7863|val 0.7915|Time 2.67mins
loss improvement on epoch: 6
[006/100]|train 0.7544|val 0.7825|Time 3.21mins
loss improvement on epoch: 7
[007/100]|train 0.7626|val 0.7768|Time 3.73mins
loss improvement on epoch: 8
[008/100]|train 0.7298|val 0.7711|Time 4.22mins
loss improvement on epoch: 9
[009/100]|train 0.7413|val 0.7691|Time 4.77mins
loss improvement on epoch: 10
[010/100]|train 0.7120|val 0.7668|Time 5.26mins
loss improvement on epoch: 11
[011/100]|train 0.7271|val 0.7664|Time 5.71mins
loss improvement on epoch: 12
[012/100]|train 0.6992|val 0.7662|Time 6.21mins
loss improvement on epoch: 13
[013/100]|train 0.7151|val 0.7655|Time 6.70mins
loss improvement on epoch: 14
[014/100]|train 0.6884|val 0.7645|Time 7.22mins
[015/100]|train 0.7043|val 0.7647|Time 7.71mins
[016/100]|train 0.6774|val 0.7680|Time 8.19mins
[017/100]|train 0.6934|val 0.7655|Time 8.69mins
[018/100]|train 0.6660|val 0.7689|Time 9.19mins
[019/100]|train 0.6833|val 0.7699|Time 9.63mins
[020/100]|train 0.6551|val 0.7723|Time 10.09mins
[021/100]|train 0.6720|val 0.7749|Time 10.55mins
[022/100]|train 0.6440|val 0.7776|Time 11.03mins
[023/100]|train 0.6617|val 0.7770|Time 11.62mins
[024/100]|train 0.6340|val 0.7803|Time 12.17mins
early stopping after epoch 024
RMSE: 0.8744
Save weight to:withSex_noAge_para2.weights
Prec10 0.7996|Rec10 0.5600
```

para 3-4 vulcan

In [ ]:
```
# para 3 --> vulcan
n_factors = 200
hidden = [500] * 3
g_factor = 25

# para 4 --> vulcan
n_factors = 200
hidden = [500] * 3
g_factor = 50
```

para 5-7

In [20]:
```
n_factors = 200
hidden = [750] * 3
g_factor = 25

embedding_dropout = 0.05
dropouts = [0.5,0.5,0.25]
# training
lr = 1e-3
wd = 1e-5
bs =2000
n_epochs = 100
patience = 10

model = EmbeddingNetGender(n_users, n_movies,
                       n_factors=n_factors, hidden=hidden, dropouts=dropouts, g_factor=g_factor)
best_weights = train_model(datasets, model, lr, wd, bs, n_epochs, patience)
df_final = get_result_df(datasets, model, best_weights, bs, save_path='withSex_noAge_para5.weights')
get_precision_recall(df_final, k=10, threshold=3.5)
```

```
loss improvement on epoch: 1
[001/100]|train 0.9203|val 0.8173|Time 0.71mins
loss improvement on epoch: 2
[002/100]|train 0.7966|val 0.8034|Time 1.46mins
loss improvement on epoch: 3
[003/100]|train 0.8014|val 0.7974|Time 2.18mins
loss improvement on epoch: 4
[004/100]|train 0.7661|val 0.7866|Time 2.88mins
loss improvement on epoch: 5
[005/100]|train 0.7710|val 0.7833|Time 3.58mins
loss improvement on epoch: 6
[006/100]|train 0.7310|val 0.7690|Time 4.26mins
loss improvement on epoch: 7
[007/100]|train 0.7417|val 0.7648|Time 4.98mins
loss improvement on epoch: 8
[008/100]|train 0.7067|val 0.7620|Time 5.68mins
[009/100]|train 0.7224|val 0.7641|Time 6.47mins
loss improvement on epoch: 10
[010/100]|train 0.6909|val 0.7613|Time 7.22mins
[011/100]|train 0.7072|val 0.7624|Time 7.91mins
[012/100]|train 0.6753|val 0.7636|Time 8.62mins
[013/100]|train 0.6931|val 0.7653|Time 9.29mins
[014/100]|train 0.6587|val 0.7675|Time 10.06mins
[015/100]|train 0.6770|val 0.7665|Time 10.78mins
[016/100]|train 0.6418|val 0.7717|Time 11.46mins
[017/100]|train 0.6606|val 0.7700|Time 12.11mins
[018/100]|train 0.6247|val 0.7781|Time 12.87mins
[019/100]|train 0.6446|val 0.7748|Time 13.65mins
[020/100]|train 0.6075|val 0.7838|Time 14.50mins
early stopping after epoch 020
RMSE: 0.8722
Save weight to:withSex_noAge_para5.weights
Prec10 0.7983|Rec10 0.5599
```

In [21]:
```
n_factors = 200
hidden = [750] * 3
g_factor = 50

embedding_dropout = 0.05
dropouts = [0.5,0.5,0.25]
# training
lr = 1e-3
wd = 1e-5
bs =2000
n_epochs = 100
patience = 10

model = EmbeddingNetGender(n_users, n_movies,
                       n_factors=n_factors, hidden=hidden, dropouts=dropouts, g_factor=g_factor)
best_weights = train_model(datasets, model, lr, wd, bs, n_epochs, patience)
df_final = get_result_df(datasets, model, best_weights, bs, save_path='withSex_noAge_para6.weights')
get_precision_recall(df_final, k=10, threshold=3.5)
```

```
loss improvement on epoch: 1
[001/100]|train 0.9351|val 0.8205|Time 0.74mins
loss improvement on epoch: 2
[002/100]|train 0.7982|val 0.8045|Time 1.46mins
loss improvement on epoch: 3
[003/100]|train 0.8043|val 0.8019|Time 2.16mins
loss improvement on epoch: 4
[004/100]|train 0.7711|val 0.7903|Time 2.89mins
loss improvement on epoch: 5
[005/100]|train 0.7772|val 0.7842|Time 3.57mins
loss improvement on epoch: 6
[006/100]|train 0.7398|val 0.7718|Time 4.22mins
loss improvement on epoch: 7
[007/100]|train 0.7494|val 0.7686|Time 4.87mins
loss improvement on epoch: 8
[008/100]|train 0.7141|val 0.7631|Time 5.54mins
[009/100]|train 0.7292|val 0.7652|Time 6.21mins
loss improvement on epoch: 10
[010/100]|train 0.6991|val 0.7620|Time 6.88mins
[011/100]|train 0.7163|val 0.7621|Time 7.54mins
[012/100]|train 0.6862|val 0.7622|Time 8.20mins
[013/100]|train 0.7037|val 0.7626|Time 8.87mins
[014/100]|train 0.6739|val 0.7663|Time 9.52mins
[015/100]|train 0.6918|val 0.7651|Time 10.20mins
[016/100]|train 0.6611|val 0.7692|Time 10.89mins
[017/100]|train 0.6801|val 0.7636|Time 11.54mins
[018/100]|train 0.6473|val 0.7723|Time 12.19mins
[019/100]|train 0.6663|val 0.7732|Time 12.86mins
[020/100]|train 0.6332|val 0.7797|Time 13.56mins
early stopping after epoch 020
RMSE: 0.8738
Save weight to:withSex_noAge_para6.weights
Prec10 0.7997|Rec10 0.5639
```

In [22]:
```
n_factors = 200
hidden = [750] * 3
g_factor = 100

embedding_dropout = 0.05
dropouts = [0.5,0.5,0.25]
# training
lr = 1e-3
wd = 1e-5
bs =2000
n_epochs = 100
patience = 10

model = EmbeddingNetGender(n_users, n_movies,
                       n_factors=n_factors, hidden=hidden, dropouts=dropouts, g_factor=g_factor)
best_weights = train_model(datasets, model, lr, wd, bs, n_epochs, patience)
df_final = get_result_df(datasets, model, best_weights, bs, save_path='withSex_noAge_para7.weights')
get_precision_recall(df_final, k=10, threshold=3.5)
```

```
loss improvement on epoch: 1
[001/100]|train 0.9757|val 0.8273|Time 0.66mins
loss improvement on epoch: 2
[002/100]|train 0.8037|val 0.8085|Time 1.32mins
loss improvement on epoch: 3
[003/100]|train 0.8091|val 0.8084|Time 1.99mins
loss improvement on epoch: 4
[004/100]|train 0.7793|val 0.7973|Time 2.70mins
loss improvement on epoch: 5
[005/100]|train 0.7884|val 0.7962|Time 3.42mins
loss improvement on epoch: 6
[006/100]|train 0.7592|val 0.7865|Time 4.16mins
loss improvement on epoch: 7
[007/100]|train 0.7695|val 0.7855|Time 4.89mins
loss improvement on epoch: 8
[008/100]|train 0.7400|val 0.7784|Time 5.67mins
loss improvement on epoch: 9
[009/100]|train 0.7523|val 0.7762|Time 6.46mins
loss improvement on epoch: 10
[010/100]|train 0.7232|val 0.7713|Time 7.21mins
[011/100]|train 0.7369|val 0.7719|Time 7.95mins
loss improvement on epoch: 12
[012/100]|train 0.7093|val 0.7687|Time 8.65mins
[013/100]|train 0.7248|val 0.7743|Time 9.39mins
loss improvement on epoch: 14
[014/100]|train 0.6986|val 0.7676|Time 10.14mins
[015/100]|train 0.7140|val 0.7678|Time 10.88mins
[016/100]|train 0.6878|val 0.7679|Time 11.58mins
[017/100]|train 0.7043|val 0.7698|Time 12.32mins
[018/100]|train 0.6783|val 0.7690|Time 13.02mins
[019/100]|train 0.6952|val 0.7687|Time 13.75mins
[020/100]|train 0.6684|val 0.7729|Time 14.47mins
[021/100]|train 0.6856|val 0.7745|Time 15.22mins
[022/100]|train 0.6591|val 0.7756|Time 15.98mins
[023/100]|train 0.6770|val 0.7769|Time 16.73mins
[024/100]|train 0.6502|val 0.7764|Time 17.49mins
early stopping after epoch 024
RMSE: 0.8756
Save weight to:withSex_noAge_para7.weights
Prec10 0.7967|Rec10 0.5614
```

---

para 8-10 mars10

para 11-13