推荐系统：打卡地点推荐

I - 问题界定

主业务问题：基于用户的社交数据和地点打卡数据，向用户推荐可能感兴趣的地点。

- 数据准备

In [1]:

```
import pandas as pd
import numpy as np
```

1.数据检视

In [2]:

```
# 用户社交网络数据
social_data = pd.read_csv('Gowalla_edges.txt', sep='\t', header=None, names=['u','v'])
social_data
```

Out[2]:

	u	v
0	0	1
1	0	2
2	0	3
3	0	4
4	0	5
...	...	...
1900649	196586	196539
1900650	196587	196540
1900651	196588	196540
1900652	196589	196547
1900653	196590	196561

1900654 rows × 2 columns

In [3]:

```
# 用户数量
social_data['u'].nunique()
```

Out[3]:

196591

In [4]:

```
# 社交数据的描述性统计
social_data.describe(include='all')
```

Out[4]:

	u	v
count	1.900654e+06	1.900654e+06
mean	5.100774e+04	5.100774e+04
std	5.010527e+04	5.010527e+04
min	0.000000e+00	0.000000e+00
25%	7.399000e+03	7.399000e+03
50%	3.734000e+04	3.734000e+04
75%	8.212000e+04	8.212000e+04
max	1.965900e+05	1.965900e+05

In [5]:

```
# 用户打卡数据
behavior_data = pd.read_csv('Gowalla_totalCheckins.txt', sep='\t',
                            header=None, names=['u','time','x','y','p'])
behavior_data
```

Out[5]:

	u	time	x	y	p
0	0	2010-10-19T23:55:27Z	30.235909	-97.795140	22847
1	0	2010-10-18T22:17:43Z	30.269103	-97.749395	420315
2	0	2010-10-17T23:42:03Z	30.255731	-97.763386	316637
3	0	2010-10-17T19:26:05Z	30.263418	-97.757597	16516
4	0	2010-10-16T18:50:42Z	30.274292	-97.740523	5535878
...	...	...	...	...	...
6442887	196578	2010-06-11T13:32:26Z	51.742988	-0.488065	906885
6442888	196578	2010-06-11T13:26:45Z	51.746492	-0.490780	965121
6442889	196578	2010-06-11T13:26:34Z	51.741916	-0.496729	1174322
6442890	196585	2010-10-08T21:01:49Z	50.105516	8.571525	471724
6442891	196585	2010-10-07T17:39:18Z	50.027812	8.785098	4555073

6442892 rows × 5 columns

In [6]:

```
# 打卡用户数量
behavior_data['u'].nunique()
```

Out[6]:

107092

In [7]:

```
# 打卡地点数量
behavior_data['p'].nunique()
```

Out[7]:

1280969

In [8]:

```
# 打卡数据的描述性统计
behavior_data.describe(include='all')
```

Out[8]:

	u	time	x	y	p
count	6.442892e+06	6442892	6.442892e+06	6.442892e+06	6.442892e+06
unique	NaN	5561957	NaN	NaN	NaN
top	NaN	2010-10-08T17:50:58Z	NaN	NaN	NaN
freq	NaN	8	NaN	NaN	NaN
mean	6.043642e+04	NaN	4.052177e+01	-4.744338e+01	7.257161e+05
std	5.427504e+04	NaN	1.476714e+01	6.636130e+01	9.501359e+05
min	0.000000e+00	NaN	-9.000000e+01	-1.763086e+02	8.904000e+03
25%	1.104800e+04	NaN	3.340766e+01	-9.767548e+01	1.125330e+05
50%	4.229500e+04	NaN	3.988993e+01	-7.806955e+01	4.249405e+05
75%	1.071570e+05	NaN	5.125089e+01	1.113241e+01	9.775450e+05
max	1.965850e+05	NaN	4.056585e+02	1.774625e+02	5.977757e+06

2.数据清理

In [9]:

```
# 社交数据的重复情况
social_data.duplicated().sum()
```

Out[9]:

0

In [12]:

```
# 打卡数据的重复情况
behavior_data.duplicated().sum()
```

Out[12]:

0

In [11]:

```
# 删除重复数据
behavior_data.drop_duplicates(inplace=True)
```

3.数据转换

In [13]:

```
# time由文本型转换为时间类型
from datetime import datetime

behavior_data['time'] = behavior_data['time'].apply(
    lambda x: datetime.strptime(x[:10],'%Y-%m-%d')
)
behavior_data
```

Out[13]:

	u	time	x	y	p
0	0	2010-10-19	30.235909	-97.795140	22847
1	0	2010-10-18	30.269103	-97.749395	420315
2	0	2010-10-17	30.255731	-97.763386	316637
3	0	2010-10-17	30.263418	-97.757597	16516
4	0	2010-10-16	30.274292	-97.740523	5535878
...	...	...	...	...	...
6442887	196578	2010-06-11	51.742988	-0.488065	906885
6442888	196578	2010-06-11	51.746492	-0.490780	965121
6442889	196578	2010-06-11	51.741916	-0.496729	1174322
6442890	196585	2010-10-08	50.105516	8.571525	471724
6442891	196585	2010-10-07	50.027812	8.785098	4555073

6442291 rows × 5 columns

In [14]:

```
# 保存处理后的打卡数据
import pickle

with open('behavior_data.pkl','wb') as f:
    pickle.dump(behavior_data, f)
```

In [15]:

```
# 导入用户打卡数据
with open('behavior_data.pkl','rb') as f:
    behavior_data = pickle.load(f)
```

III - 数据建模与可视化

1.时间上下文相关的UserCF算法

step1: 找到目标用户u的K个最相似用户

In [16]:

```
# 定义用户之间相似度计算方式
import math

def user_similarity_social(u, v):

    # 用户u、用户v的好友集合
    data_u = set(social_data.query('u == @u')['v'])
    data_v = set(social_data.query('u == @v')['v'])

    return len(data_u & data_v) / math.sqrt(len(data_u) * len(data_v))
```

In [17]:

```
# 用户u的K个相似用户
def get_uk(u, K=10):

    # 用户u的打卡地点集合
    u_p = set(behavior_data.query('u == @u')['p'])
    # 跟用户u在同一地点打过卡的用户集合
    p_v = set(behavior_data.query('p in @u_p')['u'])

    # 计算相似度
    w_df = pd.DataFrame(columns=['w'])
    for v in p_v:
        w_df.loc[v] = user_similarity_social(u, v)
    # 排序取前K个 排除用户u
    return w_df['w'].sort_values(ascending=False)[1:K+1]
```

In [18]:

```
%%time
# 算法演示
uk = get_uk(2, K=10)
```

```
CPU times: total: 1min 4s
Wall time: 2min 20s
```

```
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\3190754988.py:14: FutureWarni
ng: The behavior of `series[i:j]` with an integer-dtype index is deprecated. In a
future version, this will be treated as *label-based* indexing, consistent with e.
g. `series[i]` lookups. To retain the old behavior, use `series.iloc[i:j]`. To get
the future behavior, use `series.loc[i:j]`.
  return w_df['w'].sort_values(ascending=False)[1:K+1]
```

In [19]:

```
uk
```

Out[19]:

```
1779    0.215577
2039    0.203101
1174    0.197672
742     0.175281
590     0.171830
2292    0.170600
111     0.169438
1060    0.164228
1517    0.162230
1279    0.159814
Name: w, dtype: float64
```

step2：找到这K个最相似用户喜欢的物品（打卡地点），构成推荐列表

In [23]:

```
from tqdm import tqdm

# 为用户u推荐n个地点
def recommend_u(u, n=10, K=10):
    '''
    part1 获取初步推荐列表
    '''
    # K个相似用户的打卡地点
    uk = get_uk(u, K)
    v_p = set(behavior_data.query('u in @uk.index')['p'])
    # 排除用户u打过卡的地点
    p_set = v_p - set(behavior_data.query('u == @u')['p'])

    '''
    part2 对推荐列表排序，得到最终推荐列表
    '''
    # 将t0设置为2010年11月1日
    t0 = datetime.strptime('2010-11-01','%Y-%m-%d')
    # 用户打卡地点和时间数据(Series)
    # index(u p) value(time)
    vi_time = behavior_data.set_index(['u','p'])['time']

    # 基于用户相似度和打卡时间计算用户u对地点的兴趣
    df = pd.DataFrame(columns=['interest'])
    alpha = 0.01

    # 遍历打卡地点
    for i in tqdm(p_set):
        interest = 0
        # 遍历相似用户
        for v in uk.index:
            w = uk.loc[v]
            try:
                # 用户(v)在打卡地点(i)的所有打卡时间(time)
                t_list = pd.Series(vi_time[(v,i)])
                # 遍历打卡时间
                for t in t_list:
                    #
                    interest += w / (1 + alpha * ((t0-t).days))
            except Exception as e:
                pass
        df.loc[i] = interest

    return df['interest'].sort_values(ascending=False)[:n]
```

In [22]:

```
!pip install tqdm
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting tqdm
  Downloading tqdm-4.65.0-py3-none-any.whl (77 kB)
     ---------------------------------------- 77.1/77.1 kB 857.1 kB/s eta 0:00:00
Requirement already satisfied: colorama in c:\users\rgzn2023\appdata\roaming\pytho
n\python38\site-packages (from tqdm) (0.4.6)
Installing collected packages: tqdm
Successfully installed tqdm-4.65.0

  WARNING: The script tqdm.exe is installed in 'C:\Users\rgzn2023\AppData\Roaming
\Python\Python38\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warnin
g, use --no-warn-script-location.
```

In [24]:

```
%%time
# 算法演示
print(recommend_u(2,10,10))
```

```
ceWarning: indexing past lexsort depth may impact performance.
  t_list = pd.Series(vi_time[(v,i)])
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\2272709095.py:35: Performan
ceWarning: indexing past lexsort depth may impact performance.
  t_list = pd.Series(vi_time[(v,i)])
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\2272709095.py:35: Performan
ceWarning: indexing past lexsort depth may impact performance.
  t_list = pd.Series(vi_time[(v,i)])
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\2272709095.py:35: Performan
ceWarning: indexing past lexsort depth may impact performance.
  t_list = pd.Series(vi_time[(v,i)])
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\2272709095.py:35: Performan
ceWarning: indexing past lexsort depth may impact performance.
  t_list = pd.Series(vi_time[(v,i)])
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\2272709095.py:35: Performan
ceWarning: indexing past lexsort depth may impact performance.
```

In [25]:

```
# pip intall tqdm # 安装进度条

from tqdm import tqdm
import time

# for i in tqdm(range(1000)):
#     # 休眠1秒
#     time.sleep(1)

pbar = tqdm(list(range(10)))

for i in pbar:
    time.sleep(1)
    pbar.set_description('处理进度')
```

```
处理进度: 100%|█████████████████████████████████████████████████████████████████████████| 1
0/10 [00:10<00:00,  1.01s/it]t/s]
```

2.基于图的方法

1）set法实现：使用小数据集演示

step1：将用户打卡数据表示为二分图

In [26]:

```
# 新建用户打卡的小数据集
data = pd.DataFrame(columns=['u','p'])
data['u'] = [0,0,0,1,1,1,2,2,3,3,3,4,4]
data['p'] = [1,2,4,1,2,3,3,5,1,4,5,2,6]
data
```

...

In [27]:

```
# 为用户u和地点p的id添加标识
data['u'] = data['u'].apply(lambda x: 'u' + str(x))
data['p'] = data['p'].apply(lambda x: 'p' + str(x))
data
```

Out[27]:

	u	p
0	u0	p1
1	u0	p2
2	u0	p4
3	u1	p1
4	u1	p2
5	u1	p3
6	u2	p3
7	u2	p5
8	u3	p1
9	u3	p4
10	u3	p5
11	u4	p2
12	u4	p6

In [30]:

```
# 将用户打卡数据表示为二分图
def extra_G(data):

    G = {}
    # 存储每个用户节点和对应的打卡地点
    for u, group in data.groupby('u'):  # 分组字段u->index 其它->每组的数据
        G[u] = list(group['p'])
    # 存储每个地点节点和对应的用户
    for p, group in data.groupby('p'):
        G[p] = list(group['u'])

    return G
```

In [28]:

```
G = {}
for u, group in data.groupby('u'):
    G[u] = list(group['p'])

for p, group in data.groupby('p'):
    G[p] = list(group['u'])

G
```

...

In [31]:

```
# 调用函数生成二分图
G = extra_G(data)
G
```

Out[31]:

```
{'u0': ['p1', 'p2', 'p4'],
 'u1': ['p1', 'p2', 'p3'],
 'u2': ['p3', 'p5'],
 'u3': ['p1', 'p4', 'p5'],
 'u4': ['p2', 'p6'],
 'p1': ['u0', 'u1', 'u3'],
 'p2': ['u0', 'u1', 'u4'],
 'p3': ['u1', 'u2'],
 'p4': ['u0', 'u3'],
 'p5': ['u2', 'u3'],
 'p6': ['u4']}
```

step2: 实现PersonalRank算法，计算用户对每个节点的访问概率

In [32]:

```
# 实现PersonalRank算法
def personal_rank(G, root, alpha=0.6, n_iter=10):

    # 初始化所有节点的访问概率为0
    rank = { x: 0 for x in G.keys() }
    # 初始化vu节点的访问概率为1
    rank[root] = 1

    # 迭代n次计算用户对每个节点的访问概率
    for k in range(n_iter):
        # 新建字典保存结算结果
        tmp = {x: 0 for x in G.keys()}
        # 计算每一个节点的被访问概率
        for i in G.keys():
            for j in G[i]:
                tmp[j] += alpha * rank[i] / len(G[i])
        tmp[root] += (1 - alpha)
        # 更新节点访问概率
        rank = tmp

    return rank
```

In [33]:

```
# 算法演示
personal_rank(G, 'u0', alpha=0.6, n_iter=10)
```

Out[33]:

```
{'u0': 0.47926683040000007,
 'u1': 0.050640364800000004,
 'u2': 0.008401271999999998,
 'u3': 0.060770752000000004,
 'u4': 0.0281882624,
 'p1': 0.11729185279999998,
 'p2': 0.11359181439999998,
 'p3': 0.0121056256,
 'p4': 0.10744109439999999,
 'p5': 0.0141285184,
 'p6': 0.008173612799999999}
```

step3:给用户做推荐

In [41]:

```
# 为c个用户各推荐2个地点
def recommend(c=[1,3], k=2):

    # 新建字典保存结果
    recommend = {}
    for i in c:
        # 合成用户id
        u = 'u' + str(i)
        # 计算节点访问概率
        rank = personal_rank(G, u, alpha=0.6, n_iter=10)
        # 形成初步推荐列表
        res = {}
        for key, value in rank.items():
            # 保留未打开的地点节点
            if (key[0] == 'p') and (key not in G[u]):
                res[key] = value
        # 排序并选取前2个地点保存到推荐列表 x(key,value) -> x[1]
        recommend[u] = sorted(res.items(), key=lambda x: x[1], reverse=True)[:k]

    return recommend
```

In [42]:

```
# 算法演示
recommend([3,2])
```

Out[42]:

```
{'u3': [('p2', 0.0195607808), ('p3', 0.0176268832)],
 'u2': [('p1', 0.024533798399999994), ('p2', 0.014214556799999998)]}
```

2) 算法实现：案例演示

In [43]:

```
import pickle
from tqdm import tqdm

# 导入用户打卡数据
with open('behavior_data.pkl', 'rb') as f:
    behavior_data = pickle.load(f)
behavior_data
```

Out[43]:

	u	time	x	y	p
0	0	2010-10-19	30.235909	-97.795140	22847
1	0	2010-10-18	30.269103	-97.749395	420315
2	0	2010-10-17	30.255731	-97.763386	316637
3	0	2010-10-17	30.263418	-97.757597	16516
4	0	2010-10-16	30.274292	-97.740523	5535878
...	...	...	...	...	...
6442887	196578	2010-06-11	51.742988	-0.488065	906885
6442888	196578	2010-06-11	51.746492	-0.490780	965121
6442889	196578	2010-06-11	51.741916	-0.496729	1174322
6442890	196585	2010-10-08	50.105516	8.571525	471724
6442891	196585	2010-10-07	50.027812	8.785098	4555073

6442291 rows × 5 columns

In [44]:

```
# 删除用户和打卡地点重复数据
behavior_data.drop_duplicates(['u','p'], inplace=True)
behavior_data.shape
```

Out[44]:

(3981334, 5)

In [45]:

```
# 为用户u和地点p的id添加标识
behavior_data['u'] = behavior_data['u'].apply(lambda x: 'u' + str(x))
behavior_data['p'] = behavior_data['p'].apply(lambda x: 'p' + str(x))
behavior_data.head()
```

Out[45]:

	u	time	x	y	p
0	u0	2010-10-19	30.235909	-97.795140	p22847
1	u0	2010-10-18	30.269103	-97.749395	p420315
2	u0	2010-10-17	30.255731	-97.763386	p316637
3	u0	2010-10-17	30.263418	-97.757597	p16516
4	u0	2010-10-16	30.274292	-97.740523	p5535878

In [46]:

```
# 将用户打卡数据表示为二分图
def extra_G(data):

    G = {}
    # 存储每个用户节点和对应的打卡地点
    tbar_u = tqdm(data.groupby('u'))
    for u, group in tbar_u:
        G[u] = list(group['p'])
        tbar_u.set_description('处理用户节点')
    # 存储每个地点节点和对应的用户
    tbar_p = tqdm(data.groupby('p'))
    for p, group in tbar_p:
        G[p] = list(group['u'])
        tbar_p.set_description('处理地点节点')

    return G
```

In [5]:

```
# 实现PersonalRank算法
def personal_rank(G, root, alpha=0.6, n_iter=10):

    # 初始化所有节点的访问概率为0
    rank = {x: 0 for x in G.keys()}
    # 初始化vu节点的访问概率为1
    rank[root] = 1

    # 迭代n次计算用户对每个节点的访问概率
    for k in range(n_iter):
        # 新建字典保存结算结果
        tmp = {x: 0 for x in G.keys()}
        # 计算每一个节点的被访问概率
        for i in G.keys():
            for j in G[i]:
                tmp[j] += alpha * rank[i] / len(G[i])
        tmp[root] += (1 - alpha)
        # 更新节点访问概率
        rank = tmp

    return rank
```

In [47]:

```

# 为id小于100的用户各推荐50个地点
def recommend():

    # 新建字典保存结果
    recommend = {}
    tbar = tqdm(range(100))
    for i in tbar:
        # 合成用户id
        u = 'u' + str(i)
        # 确认该id在二分图中
        if u in G.keys():
            # 计算节点访问概率
            rank = personal_rank(G, u, alpha=0.6, n_iter=10)
            # 形成初步推荐列表
            res = {}
            for key, value in rank.items():
                # 保留未打卡的地点节点
                if (key[0] =='p') and (key not in G[u]):
                    res[key] = value
            # 排序并选取前50个地点保存到推荐列表
            recommend[u] = sorted(res.items(), key=lambda x: x[1], reverse=True)[:50]
            tbar.set_description('筛选用户打卡地点')

    return recommend
```

In [49]:

```
# 拆分训练集与测试集
def data_split(data):

    # 测试集
    test_index = []
    tbar = tqdm(range(1000))
    for i in tbar:
        u = 'u' + str(i)
        # 判断用户打卡地点数是否大于5
        if len(data[data['u'] == u]) > 5:
            # 提取用户的前5个打卡记录
            test_index.extend(data[data['u'] == u].index[:5])
        tbar.set_description('测试集处理')
    test_data = data.loc[test_index]

    # 训练集
    train_index = set(data.index) - set(test_index)
    train_data = data.loc[train_index]

    return train_data, test_data
```

In [*]:

```
%%time
# 算法演示

# 训练数据
train_data = data_split(behavior_data)[0]
# 将训练集表示为二分图
G = extra_G(train_data)
# 推荐列表
recommend_place = recommend()
```

```
测试集处理: 100%|████████████████████████████████
██████████████████████████████████| 1000/1000 [0
7:52<00:00,  2.12it/s]it/s]
C:\Users\rgzn2023\AppData\Local\Temp\ipykernel_25040\1103769920.py:18: FutureWarni
ng: Passing a set as an indexer is deprecated and will raise in a future version.
Use a list instead.
  train_data = data.loc[train_index]
处理用户节点: 100%|███████████████████████████████
████████████████████████████| 107092/107092 [04:31<00:
00, 394.35it/s]?it/s]
处理地点节点:   9%|█████                                    |
114703/1280361 [05:10<1:17:36, 250.30it/s]?it/s]
```

In [9]:

```
# 输出结果
with open('recommend_place_graph.pkl','wb') as f:
    pickle.dump(recommend_place, f)
```

In [10]:

```
# 读取结果数据
with open('recommend_place_graph.pkl','rb') as f:
    res = pickle.load(f)

res
```

```
  ('p4727569', 8.319187635007402e-05),
  ('p476254', 8.282132932100817e-05),
  ('p195136', 8.276395219358745e-05),
  ('p1305095', 8.26254245690645e-05),
  ('p1560791', 8.26254245690645e-05),
  ('p1561262', 8.26254245690645e-05),
  ('p1876711', 8.26254245690645e-05),
  ('p2362880', 8.26254245690645e-05),
  ('p3389722', 8.26254245690645e-05),
  ('p4403203', 8.26254245690645e-05),
  ('p5537459', 8.26254245690645e-05),
  ('p349642', 8.140270683877037e-05),
  ('p14475', 8.133584009306749e-05),
  ('p39440', 7.929368889318013e-05),
  ('p20484', 7.738555431224855e-05),
  ('p655046', 7.395623662994595e-05),
  ('p385555', 7.349229080991209e-05),
  ('p23372', 7.342929485638989e-05),
  ('p22831', 7.115840362312808e-05),
  ('p10259', 7.057935731196482e-05),
```

3.基于矩阵分解的方法

1) 传统矩阵分解

In [1]:

```
import pickle
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt

# 导入用户打卡数据
with open('behavior_data.pkl', 'rb') as f:
    behavior_data = pickle.load(f)
```

In [2]:

```
# 获取评分数据
behavior_data.drop_duplicates(['u','p'], inplace=True)
behavior_data['s'] = 1
ratings = behavior_data[['u','p','s']].copy()
ratings
```

Out[2]:

	u	p	s
0	0	22847	1
1	0	420315	1
2	0	316637	1
3	0	16516	1
4	0	5535878	1
...	...	...	...
6442886	196578	965051	1
6442887	196578	906885	1
6442888	196578	965121	1
6442890	196585	471724	1
6442891	196585	4555073	1

3981334 rows × 3 columns

In [3]:

```
# 切分训练集与测试集
def split_rating_data(data, n):

    # 测试集
    test_index = []
    for u in range(n):
        if len(data[data['u'] == u]) > 5:
            test_index.extend(data[data['u']==u].index[:5])
    test_data = data.loc[test_index]

    # 训练集
    train_index = set(data.index) - set(test_index)
    train_data = data.loc[train_index]

    return train_data, test_data
```

In [4]:

```
small_ratings = ratings.query('u < 20').copy()
# small_ratings
train_data, test_data = split_rating_data(small_ratings, n=20)
train_data
```

```
C:\Users\86135\AppData\Local\Temp\ipykernel_27424\160065438.py:13: FutureWarning:
Passing a set as an indexer is deprecated and will raise in a future version. Use
a list instead.
  train_data = data.loc[train_index]
```

Out[4]:

	u	p	s
5	0	15372	1
6	0	21714	1
8	0	153505	1
10	0	23261	1
11	0	16907	1
...	...	...	...
4142	19	692618	1
4143	19	160220	1
4144	19	583165	1
4145	19	31324	1
4146	19	34714	1

3002 rows × 3 columns

In [5]:

```
# 初始化矩阵P、Q
def initPQ(R, K):

    # 用户数量n和地点数量m
    n = len(set(R['u']))
    m = len(set(R['p']))
    # 初始化矩阵P、Q
    P = pd.DataFrame(np.random.rand(n, K), index=list(set(R['u'])), columns=range(K))
    Q = pd.DataFrame(np.random.rand(K, m), index=range(K), columns=list(set(R['p'])))

    return P, Q
```

In [6]:

```
P,Q = initPQ(train_data, K=2)
print(P.shape, Q.shape)
```

```
(18, 2) (2, 2917)
```

In [7]:

```
Q[15372]
```

Out[7]:

```
0    0.320541
1    0.936608
Name: 15372, dtype: float64
```

In [8]:

```
# 计算均方根误差RMSE
def root_mean_squared_error(R, P, Q):

    # 计算真实评分和预测评分的平方损失
    squared_error = [pow(rui - np.dot(P.loc[u], Q[i]), 2) for u, i, rui in R.values]
    # 计算RMSE
    rmse = np.sqrt(np.mean(squared_error))

    return rmse
```

In [9]:

```
R = train_data
squared_error = [pow(rui - np.dot(P.loc[u], Q[i]), 2) for u, i, rui in R.values]
np.sqrt(np.mean(squared_error))

rmse = root_mean_squared_error(train_data, P, Q)
rmse
```

Out[9]:

```
0.6964454556514479
```

In [10]:

```
# 获取矩阵P、Q的估计值，以及每次迭代的预测误差RMSE
def matrix_factorization(R, test_data, K=2, max_iter=1000, tol=0.001, alpha=0.0002, beta=0.02):

    # 初始化矩阵P、Q
    P, Q = initPQ(R, K)
    # 创建列表保存训练集和测试集的RMSE
    train_rmse_list = []
    test_rmse_list = []

    # 在迭代中计算矩阵P、Q的估计值，并计算训练集和测试集的RMSE
    for step in tqdm(range(max_iter)):

        # 优化估计矩阵P、Q
        for u, i, rui in R.values:
            # 真实评分与预测评分的差值
            eui = rui - np.dot(P.loc[u], Q[i])
            # 根据随机梯度下降优化估计矩阵P、Q
            for k in range(K):
                P.loc[u,k] += alpha * (eui * Q.loc[k,i] - beta * P.loc[u,k])
                Q.loc[k,i] += alpha * (eui * P.loc[u,k] - beta * Q.loc[k,i])

        # 训练误差RMSE
        train_rmse = root_mean_squared_error(R, P, Q)
        train_rmse_list.append(train_rmse)
        # 测试误差RMSE
        train_p = set(R['p'])
        test_new = test_data.query('p in @train_p')
        test_rmse_list.append(root_mean_squared_error(test_new, P, Q))

        # 判断停止迭代条件
        if train_rmse < tol:
            break

    return P, Q, train_rmse_list, test_rmse_list
```

In [ ]:

```
# 测试一下矩阵分解算法
# P, Q, train_rmse_list, test_rmse_list = matrix_factorization(train_data, test_data)
```

In [11]:

```
# 基于传统矩阵分解为测试集用户推荐TOP 100地点
def recommend():

    # 获取估计矩阵P、Q以及训练集和测试集的RMSE
    P, Q, train_rmse_list, test_rmse_list = matrix_factorization(train_data, test_data)

    # 为测试集用户做推荐
    recommend = {}
    for u in set(test_data['u']):
        # 用户u未打过卡的地点
        p_set = set(train_data.query('u != @u')['p'])
        # 对这些地点进行评分预测
        predict = {i: np.dot(P.loc[u], Q[i]) for i in p_set}
        # 推进TOP100地点
        recommend[u] = sorted(predict.items(), key=lambda x: x[1], reverse=True)[:100]

    return recommend, train_rmse_list, test_rmse_list
```

In [12]:

```
# 提取部分评分数据
small_ratings = ratings.query('u < 20').copy()
# 对地点编号进行数据转换
small_ratings['p'] = pd.qcut(small_ratings['p'], 200, labels=range(200))
# 删除重复打卡记录
small_ratings.drop_duplicates(['u', 'p'], inplace=True)
small_ratings
```

Out[12]:

	u	p	s
0	0	24	1
1	0	109	1
2	0	96	1
3	0	16	1
4	0	199	1
...	...	...	...
4141	19	113	1
4143	19	74	1
4144	19	128	1
4145	19	31	1
4146	19	34	1

1049 rows × 3 columns

In [13]:

```
%%time
# 算法演示

# 切分训练集和测试集
train_data, test_data = split_rating_data(small_ratings, n=20)
recommend_place, train_rmse_list, test_rmse_list = recommend()
# 输出结果
with open('recommend_place_mf.pkl','wb') as f:
    pickle.dump((recommend_place, train_rmse_list, test_rmse_list), f)
```

```
C:\Users\86135\AppData\Local\Temp\ipykernel_27424\160065438.py:13: FutureWarning:
Passing a set as an indexer is deprecated and will raise in a future version. Use
a list instead.
  train_data = data.loc[train_index]
100%|██████████████████████████████████████
████████████████████████████████████████|
1000/1000 [12:55<00:00,  1.29it/s]

CPU times: total: 6min 30s
Wall time: 12min 55s
```

In [14]:

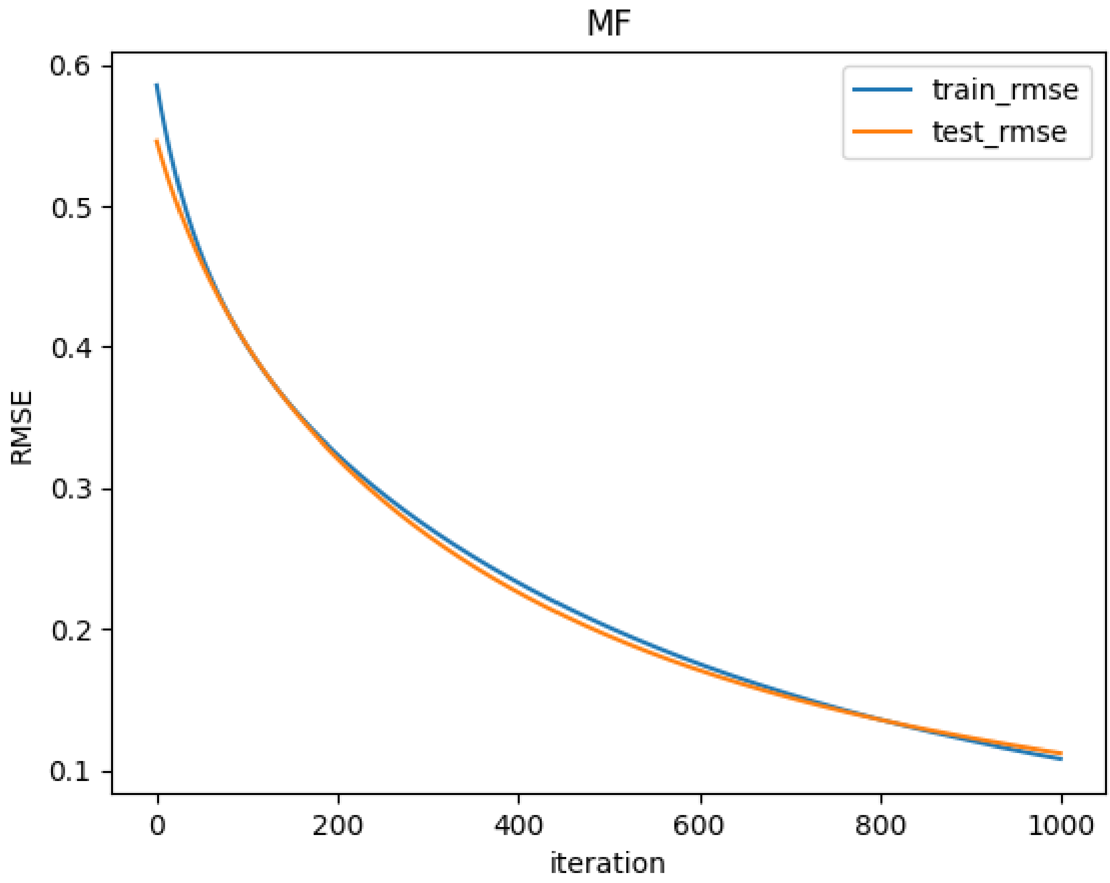
```
# 读取结果数据
with open('recommend_place_mf.pkl','rb') as f:
    res = pickle.load(f)
```

In [15]:

```
# 绘制训练集和测试集的RMSE折线图
def draw_line(train_rmse, test_rmse, title):
    x = range(len(train_rmse))
    plt.plot(x, train_rmse, label='train_rmse')
    plt.plot(x, test_rmse, label='test_rmse')
    plt.legend()
    plt.title(title)
    plt.xlabel('iteration')
    plt.ylabel('RMSE')
    plt.show()
```

In [16]:

```
# 使用结果数据绘图
draw_line(res[1], res[2], 'MF')
```

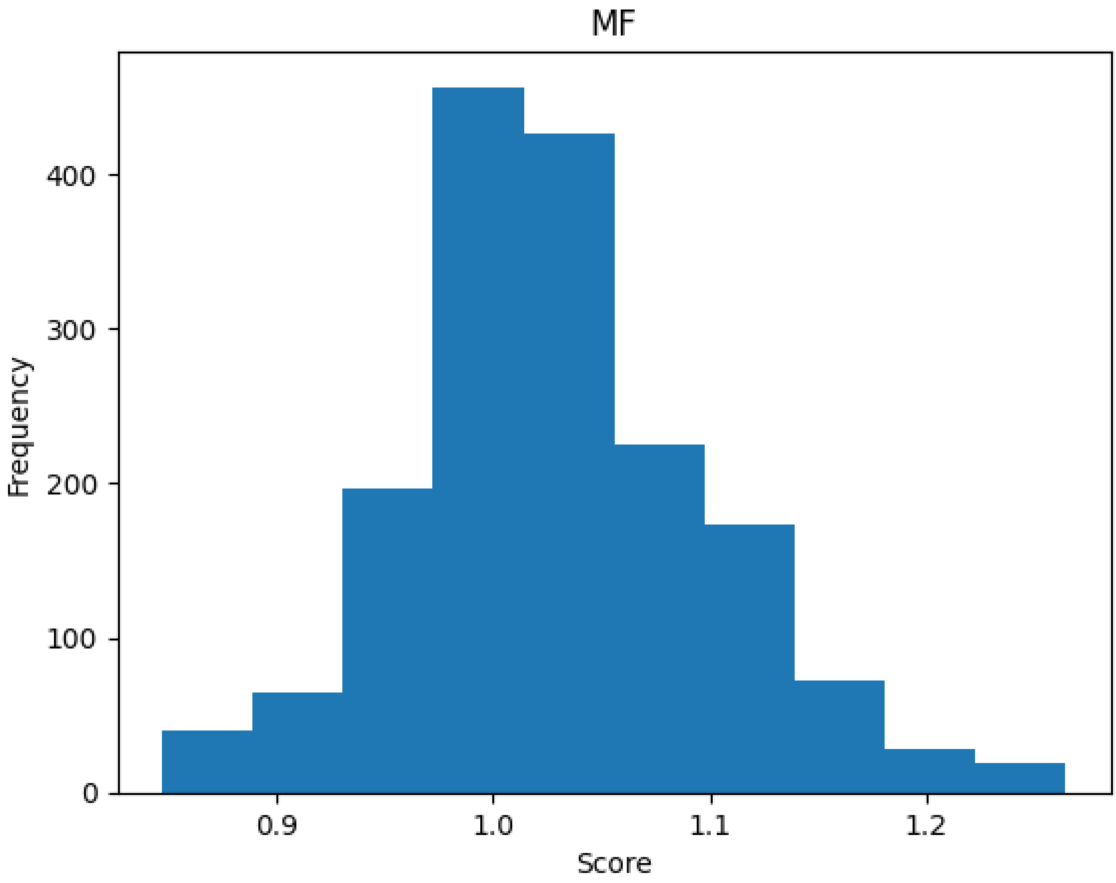


In [17]:

```
# 绘制推荐度评分直方图
def draw_hist(data, title):
    # 获取推荐度评分
    scores = []
    for value in data.values():
        scores.extend([x[1] for x in value])
    # 绘制直方图
    plt.hist(scores)
    plt.title(title)
    plt.xlabel('Score')
    plt.ylabel('Frequency')
    plt.show()
```

In [18]:

```
# 绘制推荐结果中推荐度评分的分布情况
draw_hist(res[0], 'MF')
```



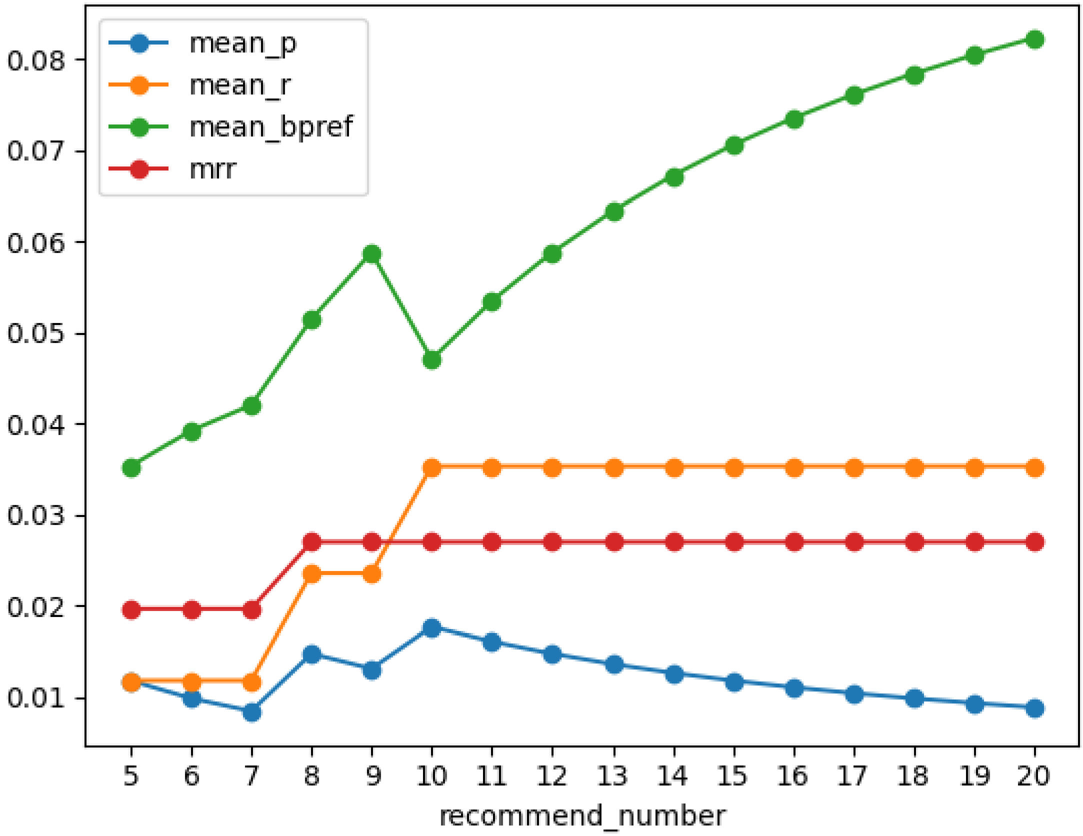
In [19]:

```
# 计算不同推荐地点数下的四个评价指标
def matrics(test_data, recommend_place, n):
    # 针对测试集中每一个用户的推荐结果：计算P、R、Bpref和倒数等级
    matrics = pd.DataFrame(columns=['Precision', 'Recall', 'Bpref',
                                    'Peciprocal_Rank'])
    for u in test_data['u'].unique():
        # 后续实际打卡地点
        p_test = test_data.groupby('u')['p'].unique()[u]
        # 地点推荐列表，推荐地点数n
        p_pred = [x[0] for x in recommend_place[u]][:n]
        # 初始化正确推荐地点数、二元偏好度量和倒数等级
        right = 0
        bpref = 0
        rr = 0
        # 计算正确推荐地点数、二元偏好度量和倒数等级
        for i, item in enumerate(p_pred):
            if item in p_test:
                # 正确推荐地点数加1
                right += 1
                # 计算二元偏好度量
                bpref += (1 - (i+1-right)/len(p_pred))
                # 如果是第一个正确推荐地点，计算倒数等级
                if right == 1:
                    rr = 1/(i+1)
        # 计算最终的二元偏好度量
        if right > 0:
            bpref /= right
        # 计算精确度和召回率
        p, r = right/len(p_pred), right/len(p_test)
        # 将计算结果放入结果表中
        matrics.loc[u] = p, r, bpref, rr

    # 针对测试集全部推荐结果：计算P、R、Bpref的均值以及MRR
    mean_p, mean_r, mean_bpref, mrr = matrics.mean()
    return mean_p, mean_r, mean_bpref, mrr
```

In [20]:

```
# 计算不同推进地点数下的四个指标
result = pd.DataFrame(columns=['mean_p', 'mean_r', 'mean_bpref', 'mrr'])
for n in range(5, 21):
    result.loc[n] = matrics(test_data, res[0], n)
# 绘制折线图
result.plot(xticks=range(5, 21), marker='o')
plt.xlabel('recommend_number')
plt.show()
```



2) 概率矩阵分解

In [47]:

```
import pickle
import numpy as np
import pandas as pd
from tqdm import tqdm

# 导入用户打卡数据
with open('behavior_data.pkl', 'rb') as f:
    behavior_data = pickle.load(f)
# 获取评分数据
behavior_data.drop_duplicates(['u','p'], inplace=True)
behavior_data['s'] = 1
ratings = behavior_data[['u','p','s']].copy()
ratings
```

Out[47]:

	u	p	s
0	0	22847	1
1	0	420315	1
2	0	316637	1
3	0	16516	1
4	0	5535878	1
...	...	...	...
6442886	196578	965051	1
6442887	196578	906885	1
6442888	196578	965121	1
6442890	196585	471724	1
6442891	196585	4555073	1

3981334 rows × 3 columns

In [49]:

```
# 切分训练集与测试集
def split_rating_data(data, n):

    # 测试集
    test_index = []
    for u in range(n):
        if len(data[data['u'] == u]) > 5:
            test_index.extend(data[data['u']==u].index[:5])
    test_data = data.loc[test_index]

    # 训练集
    train_index = set(data.index) - set(test_index)
    train_data = data.loc[train_index]

    return train_data, test_data
```

In [50]:

```
# 初始化矩阵P、Q
def initPQ(R, K):

    # 用户数量n和地点数量m
    n = len(set(R['u']))
    m = len(set(R['p']))
    # 初始化正态分布的矩阵P、Q
    P = pd.DataFrame(0.4 * np.random.randn(n, K), index=list(set(R['u'])), columns=range(K))
    Q = pd.DataFrame(0.4 * np.random.randn(K, m), index=range(K), columns=list(set(R['p'])))

    # 创建矩阵P、Q对应的零矩阵
    P_v = pd.DataFrame(np.zeros((n,K)), index=list(set(R['u'])), columns=range(K))
    Q_v = pd.DataFrame(np.zeros((K,m)), index=range(K), columns=list(set(R['p'])))

    return P, Q, P_v, Q_v
```

In [43]:

```
plt.hist(np.random.rand(1000),bins=50)
plt.show()
```

...

In [45]:

```
plt.hist(np.random.randn(1000) * 0.4,bins=50)
plt.show()
```

...

In [46]:

```
plt.hist(np.random.randn(1000),bins=50)
plt.show()
```

...

In [4]:

```
# 计算均方根误差RMSE
def root_mean_squared_error(R, P, Q):

    # 计算真实评分和预测评分的平方损失
    squared_error = [pow(rui - np.dot(P.loc[u], Q[i]), 2) for u, i, rui in R.values]
    # 计算RMSE
    rmse = np.sqrt(np.mean(squared_error))

    return rmse
```

In [51]:

```
# 获取矩阵P、Q的估计值，以及每次迭代的预测误差RMSE
def PMF(R, test_data, K=2, max_epoch=1000, tol=0.001, alpha=0.0002, beta=0.02, num_batches=11, ba

    # 初始化矩阵P、Q
    P, Q, P_v, Q_v = initPQ(R, K)
    # 创建列表保存训练集合测试集的RMSE
    train_rmse_list = []
    test_rmse_list = []

    # 在迭代中计算矩阵P、Q的估计值， 并计算训练集和测试集的RMSE
    for epoch in tqdm(range(max_epoch), desc='epoch'):

        # 打乱训练集样本顺序
        R = R.sample(frac=1, random_state=None)
        # 更新每一批中的数据
        for batch in range(num_batches):
            # 每一批中的训练数据
            R_batch = R.iloc[batch_size * batch : batch_size * (batch + 1) - 1]
            # 优化估计矩阵P、Q
            for u, i, rui in R_batch.values:
                # 真实评分与预测评分的差值
                eui = rui - np.dot(P.loc[u], Q[i])
                # 根据动量梯度下降法优化估计矩阵P、Q
                for k in range(K):
                    P_v.loc[u,k] = momentum * P_v.loc[u,k] + (-2 * eui * Q.loc[k,i] + 2 * beta *
                    P.loc[u,k] -= alpha * P_v.loc[u,k]
                    Q_v.loc[k,i] = momentum * Q_v.loc[k,i] + (-2 * eui * P.loc[u,k] + 2 * beta *
                    Q.loc[k,i] -= alpha * Q_v.loc[k,i]
        # 训练误差RMSE
        train_rmse = root_mean_squared_error(R, P, Q)
        train_rmse_list.append(train_rmse)

        # 测试误差RMSE
        train_p = set(R['p'])
        test_new = test_data.query('p in @train_p')
        test_rmse_list.append(root_mean_squared_error(test_new, P, Q))

        # 判断停止迭代条件
        if train_rmse < tol:
            break

    return P, Q, train_rmse_list, test_rmse_list
```

In [52]:

```
# 基于概率矩阵分解为测试集用户推荐TOP 100地点
def recommend():

    # 获取估计矩阵P、Q以及训练集和测试集的RMSE
    P, Q, train_rmse_list, test_rmse_list = PMF(train_data, test_data)

    # 为测试集用户做推荐
    recommend = {}
    for u in set(test_data['u']):
        # 用户u未打过卡的地点
        p_set = set(train_data.query('u != @u')['p'])
        # 对这些地点进行评分预测
        predict = {i: np.dot(P.loc[u], Q[i]) for i in p_set}
        # 推进TOP100地点
        recommend[u] = sorted(predict.items(), key=lambda x: x[1], reverse=True)[:100]

    return recommend, train_rmse_list, test_rmse_list
```

In [53]:

```
# 提取部分评分数据
small_ratings = ratings.query('u < 20').copy()
# 对地点编号进行数据转换
small_ratings['p'] = pd.qcut(small_ratings['p'], 200, labels=range(200))
# 删除重复打卡记录
small_ratings.drop_duplicates(['u','p'], inplace=True)
small_ratings
```

Out[53]:

	u	p	s
0	0	24	1
1	0	109	1
2	0	96	1
3	0	16	1
4	0	199	1
...	...	...	...
4141	19	113	1
4143	19	74	1
4144	19	128	1
4145	19	31	1
4146	19	34	1

1049 rows × 3 columns

In [57]:

```
%%time
# 算法演示
# 切分训练集合测试集
train_data, test_data = split_rating_data(small_ratings, n=20)
recommend_place, train_rmse_list, test_rmse_list = recommend()
```

```
C:\Users\86135\AppData\Local\Temp\ipykernel_30860\160065438.py:13: FutureWarning:
Passing a set as an indexer is deprecated and will raise in a future version. Use
a list instead.
  train_data = data.loc[train_index]
epoch: 100%|███████████████████████████████████
████████████████████████████████████| 1000/100
0 [26:32<00:00,  1.59s/it]

CPU times: total: 10min 39s
Wall time: 26min 33s
```

In [58]:

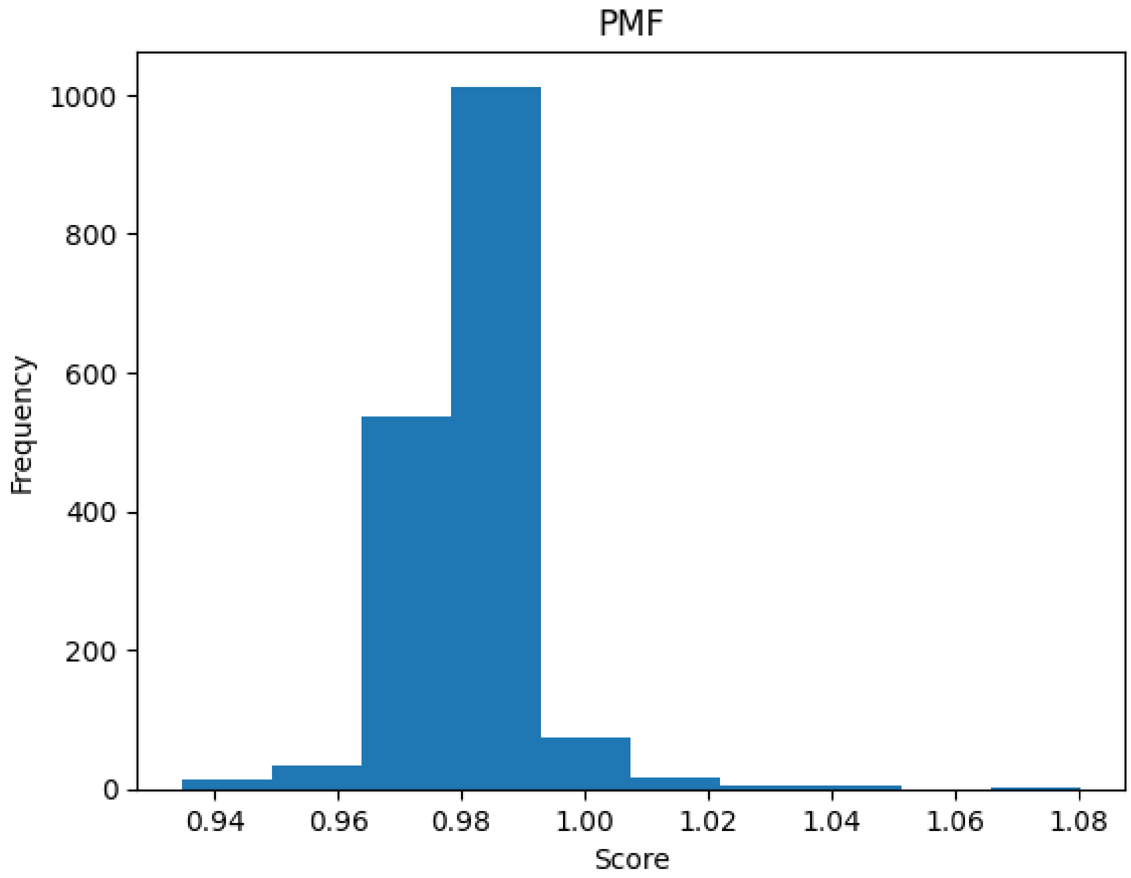
```
# 输出结果
with open('recommend_place_pmf.pkl','wb') as f:
    pickle.dump((recommend_place, train_rmse_list, test_rmse_list),f)
```

In [59]:

```
# 读取结果数据
with open('recommend_place_pmf.pkl','rb') as f:
    res = pickle.load(f)
```

In [60]:

```
# 绘制推荐结果中推荐度评分的分布情况
draw_hist(res[0], 'PMF')
```

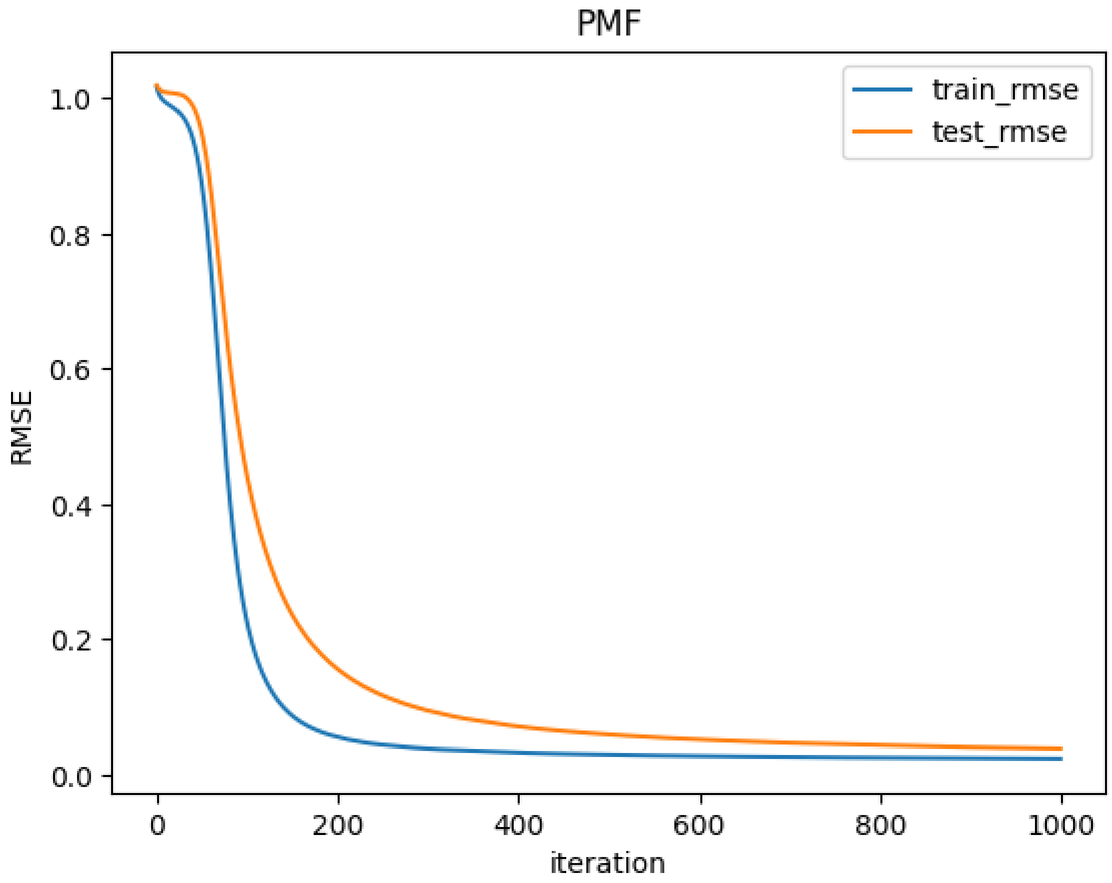


In [61]:

```
# 绘制训练集和测试集的RMSE折线图
def draw_line(train_rmse, test_rmse, title):
    x = range(len(train_rmse))
    plt.plot(x, train_rmse, label='train_rmse')
    plt.plot(x, test_rmse, label='test_rmse')
    plt.legend()
    plt.title(title)
    plt.xlabel('iteration')
    plt.ylabel('RMSE')
    plt.show()
```

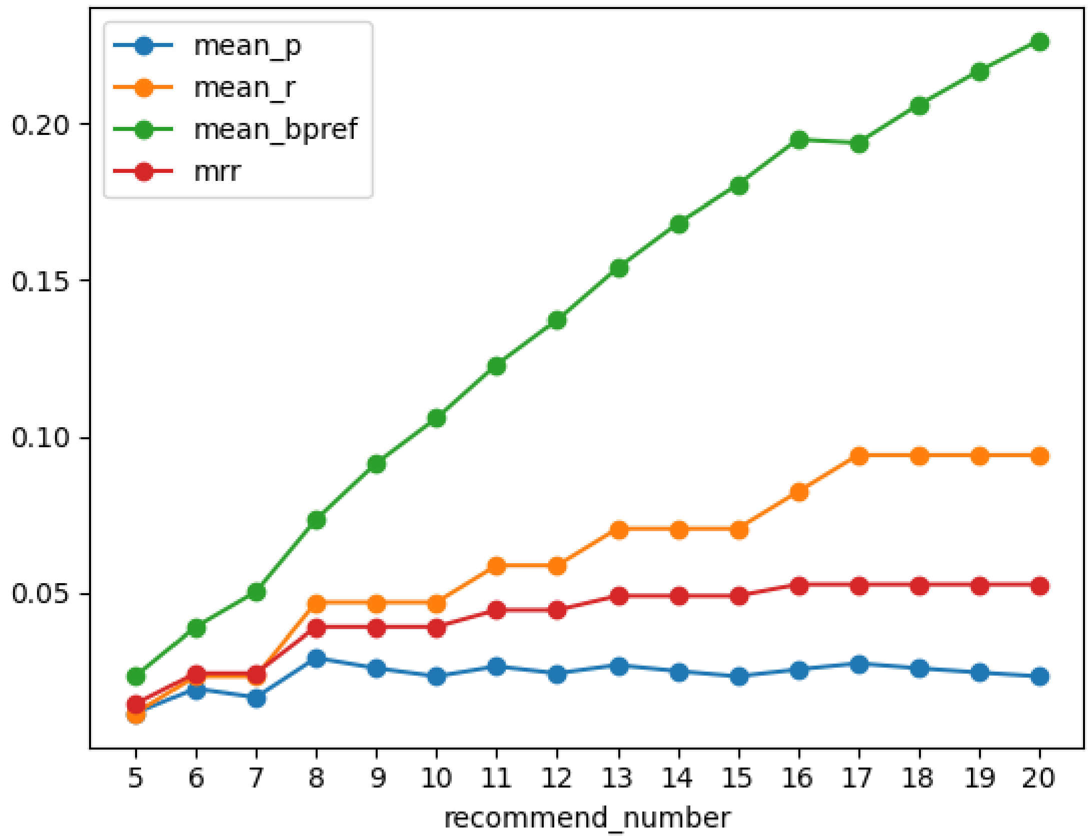
In [62]:

```
# 使用结果数据绘图
draw_line(res[1], res[2], 'PMF')
```



In [63]:

```
# 计算不同推进地点数下的四个指标
result = pd.DataFrame(columns=['mean_p', 'mean_r', 'mean_bpref', 'mrr'])
for n in range(5, 21):
    result.loc[n] = matrics(test_data, res[0], n)
# 绘制折线图
result.plot(xticks=range(5, 21), marker='o')
plt.xlabel('recommend_number')
plt.show()
```



In [ ]: