



中国科学技术大学
University of Science and Technology of China

社会计算第二次实验报告

黄业琦 SA21011007

伊 洋 SA21011041

黄 震 SA21011131

曹振伟 SA21011154

石济帆 SA21011111

谭邵杰 SA21011013

2022 年 4 月 17 日

目录

1 介绍	3
1.1 组员介绍	3
1.2 实验任务描述	3
1.3 实验任务分析	3
2 数据预处理	5
2.1 数据清洗和格式整理	5
2.1.1 数据解压和初步校验	5
2.1.2 文件内容整理和信息提取	7
2.2 选题和思考	12
2.3 数据库建立和加工	13
2.3.1 数据库选择	13
2.3.2 PostgreSQL 以及 Pgadmin 环境搭建	14
2.3.3 数据表格构建	16
2.3.4 数据库可视化界面	25
2.3.5 数据出库	26
3 实验方案	34
3.1 实验总体思路	34
3.1.1 符号定义	34
3.1.2 $f(u_i, g_k)$	34
3.1.3 $h(u_i, g_k)$	34
3.1.4 网络训练	35
3.1.5 网络验证	36
3.2 图神经网络的具体实现	36
3.2.1 创建消息传递网络	36
3.2.2 使用 PyG 的 mini-batch	38
3.3 训练过程	39
3.3.1 样本不均衡的问题及应对方法	39
4 结果分析	40
4.1 实验环境	40
4.2 损失函数选择	40
4.2.1 MSE	40
4.2.2 加权交叉熵	41

4.2.3	Focal Loss	41
4.3	实验结果	42
4.3.1	初步实验结果	42
4.3.2	地域聚合	43
4.3.3	数据精简	45
4.3.4	结果综合	47
4.3.5	存在问题	47
5	参考文献	49
6	实验代码	49

1 介绍

1.1 组员介绍

组长:

1. 黄业琦: SA21011007, huangyeqi@mail.ustc.edu.cn

组员:

1. 伊洋: SA21011041, yiyang21@mail.ustc.edu.cn
2. 黄震: SA21011131, zhenhuang@mail.ustc.edu.cn
3. 曹振伟: SA21011154, caozhenwei@mail.ustc.edu.cn
4. 石济帆: SA21011111, shijifan8@mail.ustc.edu.cn
5. 谭邵杰: SA21011013, shaojiemike@mail.ustc.edu.cn

1.2 实验任务描述

组成 4-6 人小组，围绕指定数据集进行自定方案分析实验，记录实验过程并撰写实验报告。

数据集来自著名在线活动组织网站 Meetup，其机制为：

- 整个 Meetup 社区由若干社团组成，每个社团有若干名用户，用户可以随时加入或退出。
- 活动以社团为主体，由若干名社团成员发起组织（部分活动组织者缺失）。仅有社团成员会收到邀请。社团成员可以选择是否参加活动（Yes/No/Maybe），但不是所有人都会回应。
- 部分活动会注明限制人数（Headcount），但不一定会起到约束作用。

1.3 实验任务分析

实验要求有如下几个要点：

- 解决一个基于社会网络的预测性问题。
- 预测性问题要求对数据根据时间戳进行拆分，利用历史数据对未来情况进行预测。问题本身需要有明确的可验证性及对应的量化指标。

经过小组讨论后决定实现一个动态网络，通过训练动态网络中用户在主题/标签偏好上的演变，来预测用户是否会加入某个新的社团。

2 数据预处理

2.1 数据清洗和格式整理

2.1.1 数据解压和初步校验

给定的数据格式是 XML 格式，首先我们先使用一些工具去校验数据是否是合法的 XML 格式，即第一级的表单中有且只有一个单一元素。

首先我们对于这里需要进行处理的文件对象进行路径提取和整理。

```
1 DATA_PATH=/extra/SocialComputing/All_Unpack
2 RESULT_PATH=/extra/SocialComputing/codes/data
3
4 if [[ ! -e $RESULT_PATH ]]; then
5     mkdir -p $RESULT_PATH
6 fi
7
8 # Past Events
9 FILE_REG1="PastEvent*.xml"
10 RESULT_FILE1="$RESULT_PATH/event.txt"
11 if [[ -e $RESULT_FILE1 ]]; then
12     rm $RESULT_FILE1
13 fi
14 touch $RESULT_FILE1
15 for item in ${DATA_PATH}/${FILE_REG1}; do
16     echo $item >> $RESULT_FILE1
17 done
18
19 # Member
20 FILE_REG2="Memeber*.xml"
21 RESULT_FILE2="$RESULT_PATH/member.txt"
22 if [[ -e $RESULT_FILE2 ]]; then
23     rm $RESULT_FILE2
24 fi
25 touch $RESULT_FILE2
26 for item in ${DATA_PATH}/${FILE_REG2}; do
```

```

27     echo $item >> $RESULT_FILE2
28 done
29
30 # Group
31 FILE_REG3="Group*.xml"
32 RESULT_FILE3="$RESULT_PATH/group.txt"
33 if [[ -e $RESULT_FILE3 ]]; then
34     rm $RESULT_FILE3
35 fi
36 touch $RESULT_FILE3
37 for item in ${DATA_PATH}/${FILE_REG3}; do
38     echo $item >> $RESULT_FILE3
39 done
40
41 # RSVPs
42 FILE_REG4="RSVPs*.xml"
43 RESULT_FILE4="$RESULT_PATH/rsvps.txt"
44 if [[ -e $RESULT_FILE4 ]]; then
45     rm $RESULT_FILE4
46 fi
47 touch $RESULT_FILE4
48 for item in ${DATA_PATH}/${FILE_REG4}; do
49     echo $item >> $RESULT_FILE4
50 done

```

之后我们逐个文件进行校验：

```

1 # Check all files are Leagal XML file and all of them can be dealt with pprint+xmltodict package
2 def InfoChecker(list_filename: str):
3     print("Checking " + list_filename)
4     info_list = []
5     file_list = []
6     with open(list_filename, "r") as f:
7         for line in f.readlines():
8             line = line.strip()

```

```

9         file_list.append(line)
10        file_index = 1
11        file_number = len(file_list)
12        for filename in file_list:
13            print(f"Dealing with {file_index}/{file_number} ...", end="\r")
14            file_index += 1
15            with open(filename, 'r', encoding='utf-8') as f:
16                my_xml = f.read()
17                my_dict = xmltodict.parse(my_xml)
18                # pprint.pprint(my_dict, indent = 2)
19                if len(my_dict) != 1:
20                    print("error")
21
22
23 # InfoChecker("../data/group.txt")
24 # InfoChecker("../data/member.txt")
25 # InfoChecker("../data/event.txt")
26 # InfoChecker("../data/rsvp.txt")

```

从而进行了初步的文件内容分析和理解。

2.1.2 文件内容整理和信息提取

由于 Python 的 xml 提取库会将文件转换成 OrderedDict 类型的数据，为了方便之后的查询和处理，我们提供了“递归查询模式”和“非递归查询模式”，方便对数据更好的加工和处理。

```

1 def SearchRecursive(ordered_dict, key):
2     for k, v in ordered_dict.items():
3         if isinstance(v, OrderedDict):
4             rec = SearchRecursive(v, key)
5             if rec is not None:
6                 return rec
7         if k == key:
8             return v

```

```

9     return None
10
11 def SearchUnrecursive(ordered_dict, key):
12     for k, v in ordered_dict.items():
13         if k == key:
14             return v
15     return None

```

对于每一种类型的文件（下面使用 Group 举例），我们分析我们需要的数据信息，从而完成我们需要的信息进行处理。

```

1 def ReadInfoFromGroup(filename: str):
2
3     with open(filename, 'r', encoding='utf-8') as f:
4
5         my_xml = f.read()
6
7         origin_dict = xmltodict.parse(my_xml)[ "item" ]
8
9         new_dict = {}
10
11     # who info
12
13     try:
14
15         new_dict[ "who" ] = SearchUnrecursive(origin_dict, "who")
16
17         if new_dict[ "who" ] is None:
18
19             raise TypeError
20
21     except:
22
23         raise ValueError(f"Value who is not found in {filename}")
24
25
26     # rating info
27
28     try:
29
30         new_dict[ "rating" ] = SearchUnrecursive(origin_dict, "rating")
31
32         if new_dict[ "rating" ] is None:
33
34             raise TypeError
35
36     except:
37
38         raise ValueError(f"Value rating is not found in {filename}")
39
40
41     # topics info
42
43     try:
44
45         new_dict[ "topics" ] = dict(SearchUnrecursive(origin_dict, "topics"))[ "topics_item" ]

```

```

25     if not isinstance(new_dict["topics"], list):
26         new_dict["topics"] = [dict(SearchUnrecursive(origin_dict, "topics"))["topics_item"]]
27
28     if new_dict["topics"] is None:
29         raise TypeError
30
31     except:
32         new_dict["topics"] = []
33
34     # raise ValueError(f"Value topics is not found in {filename}")
35
36     # category info
37
38     try:
39         new_dict["category"] = SearchUnrecursive(origin_dict, "category")
40
41         if new_dict["category"] is None:
42             raise TypeError
43
44     except:
45         new_dict["category"] = "unknown"
46
47         # raise ValueError(f"Value category is not found in {filename}")
48
49     # link info
50
51     try:
52         new_dict["link"] = SearchUnrecursive(origin_dict, "link")
53
54         if new_dict["link"] is None:
55             raise TypeError
56
57     except:
58         raise ValueError(f"Value link is not found in {filename}")
59
60     return new_dict

```

由于在数据处理初期我们发现大量的信息和我们预期的格式不一致，为了方便我们调查具体的原因，我们这里把每一个查询使用 try...except 进行处理。这样每一次数据卡住的时候，我们可以应对具体的情况具体定制化处理各种类型的 exception，例如用户没有写该表单数据。

在处理数据的时候，我们同时发现了 topic 数据类型应当有更好的处理方式，他和用户、组之间都是多对多的关系，多个用户或者组都可能同时关注同一个 topic。为此我们将 topic

数据单独处理，分拆出了：“用户——兴趣话题”、“小组——兴趣话题”、“兴趣话题 ID——兴趣话题名称”三个独立的数据表。方便我们进行兴趣类型的处理和划分。

```
1 def PrintcsvTopics():
2     topics_dict = TopicsExtractor()
3
4     with open("../data/topics.csv", 'w') as csvfile:
5         csvfile.write("id, name\n")
6         for index in range(len(topics_dict)):
7             key = list(topics_dict)[index]
8             val = topics_dict[key]
9             csvfile.write(f"{key}, {val}\n")
10
11 PrintcsvTopics()
```

最终为了方便我们将数据进行下一阶段的导入和数据处理，我们把数据格式转换成 CSV 格式，方便之后的操作。

```
1 def PrintcsvGroups():
2     group_info_file_list = "../data/group.txt"
3     csvfilename = "../data/group.csv"
4
5     print("Printing out Groups")
6     file_list = []
7     with open(group_info_file_list, "r") as f:
8         for line in f.readlines():
9             line = line.strip()
10            file_list.append(line)
11    file_index = 1
12    file_number = len(file_list)
13
14    csv_heading = [
15        "id",
16        "name",
17        "urlname",
18        "rating",
```

```

19     "categoryname",
20     "join_mode",
21     "lat",
22     "lon",
23     "country",
24     "city",
25     "created",
26     "members",
27     "organizer"
28 ]
29
30 with open("../data/group.csv", "w") as csvfile:
31     csvfile.write(", ".join(csv_heading) + "\n")
32     for filename in file_list:
33         print(f"Dealing with {file_index}/{file_number} ...", end="\r")
34         file_index += 1
35         info_dict = ReadInfoFromGroup(filename)
36
37         current_info = []
38         for item in csv_heading:
39             current_info.append(FormatInfoString(info_dict[item]))
40         csvfile.write("\\" + "\\", \\".join(current_info) + "\\" + "\n")
41
42     csv_heading = [
43         "groupid",
44         "topicid"
45     ]
46
47     print("Listing out Group-Topic Relations")
48     file_index = 1
49     with open("../data/grouptopics.csv", "w") as csvfile:
50         csvfile.write(", ".join(csv_heading) + "\n")
51         for filename in file_list:
52             print(f"Dealing with {file_index}/{file_number} ...", end="\r")
53             file_index += 1

```

```
54     info_dict = ReadInfoFromGroup(filename)
55
56     group_id = info_dict["id"]
57
58     if len(info_dict["topics"]) == 0:
59         continue
60
61     for item in info_dict["topics"]:
62         topic_id = item["id"]
63
64         csvfile.write(f"\{group_id}\, {topic_id}\n")
```

这个模板非常方便，我们可以自己对我们需要的“数据元素”进行定制化的提取。同时我们写了一个进度条，方便我们查看数据提取和整理的数据进度。经过上述处理，数据变成了规整的表格形式，为之后的实验奠定了较好的实验基础。

2.2 选题和思考

对数据清洗和整理之后，我们对选题进行了一些思考。并且自己在 MeetUp 网站上注册帐号，试图参加了一些线上的活动。

首先我们发现，很多 MeetUp 的活动虽然由一个社团发起，但是参与用户往往是不受限制的。即这里的社团不具有排外性。用户和社团之间的关系更加稀疏。

其次我们整理了关于“兴趣话题”的统计。用户在不同的兴趣话题上投入的活动数量，以及不同的话题的兴趣人数，进行了初步统计。我们发现这个数据其实是非常具有流动性的。我们使用图表整理了我们的用户兴趣，颜色越红表示该兴趣越热门，纵向长度越长，感兴趣的人数就越多；颜色越蓝，表示该兴趣越冷门（也有可能越邪门，比如 Sleep with cats and parrots）。

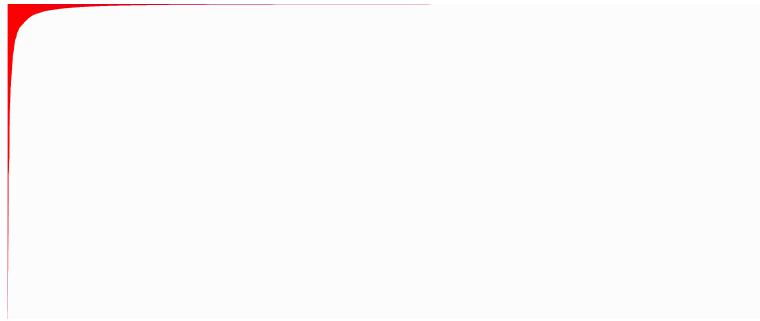


图 1: 用户兴趣统计图



图 2: 用兴趣统计缩放

可以看出用户的兴趣其实非常非常两极分化，最受欢迎的话题”Dining Out”（约饭？）有 12211 个用户进行了关注。但是冷门话题例如 Women Working from Home 只有一个人关注。但是我们认为“关注人数较少的话题的关注者之间更容易产生较强的联系，从而更容易将互相的兴趣进行交流”，同时也认为“线下互动性质更强的活动更容易产生兴趣话题的交流，从而将新人带到新话题里”。所以我们制定了实现一个动态网络，通过训练动态网络中用户在主题/标签偏好上的演变，来预测用户是否会加入某个新的社团。

2.3 数据库建立和加工

2.3.1 数据库选择

清洗后的数据按 csv 文件存储，考虑到数据量相对较大，反复读写文件不仅编写代码繁琐，而且效率较低，因此将所有数据存储到数据库中。在数据库的选择上，考虑到原始数据中包含地理信息，为了方便后续可能存在的对地理信息的处理，数据库选择使用 PostgreSQL (PostgreSQL 有一个地理数据库的扩展 PostGIS，对地理信息处理较为方便)

2.3.2 PostgreSQL 以及 Pgadmin 环境搭建

2.3.2.1 安装与配置

在默认情况下 PostgreSQL 安装完成后，自带命令行工具 SQL shell (psql)

使用 psql 命令可以直接以当前用户进入 psql 交互界面。

2.3.2.2 修改密码

登录进命令行，使用 ‘password’修改密码

```
acsacom@snode6:~$ psql
psql (12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
Type "help" for help.

acsacom=# \password
Enter new password:
Enter it again:
acsacom=# |
```

2.3.2.3 允许远程连接

Postgresql 的配置文件保存在 /etc/postgresql/12/main/postgresql.conf 路径中（以实际路径为准），修改此文件中的 listen_addresses。修改前内容如下：

```
58 # - Connection Settings -
59
60 #listen_addresses = 'localhost' # what IP address(es) to listen on;
```

修改后的內容（修改的目的是允许数据库远程连接）

```
58 # - Connection Settings -
59
60 listen_addresses = '*' # what IP address(es) to listen on;
```

之后修改同一目录下pg_hba.conf文件，虽然上面配置允许任意地址连接PostgreSQL，但是还需要在pg_hba.conf中配置服务端允许的认证方式。末尾端添加内容如下：

```
104 host    all          all      0.0.0.0/0      md5
```

默认 pg 只允许本机通过密码认证登录，修改为上面内容后即可以对任意 IP 访问进行密码验证。之后使用sudo service postgresql restart重启 PostgreSQL 服务即可生效。

2.3.2.4 配置 pgadmin

目前 pgadmin 通用的版本为 pgadmin4, sudo apt-get update之后系统可安装包只有 pgadmin3，具体原因参考 (<https://stackoverflow.com/questions/58239607/pgadmin-package-pgadmin4-has-no-installation-candidate>)

```
root@snodes6:/etc/postgresql/12/main# apt search pgadmin
Sorting... Done
Full Text Search... Done
pgadmin3/focal 1.22.2-6build1 amd64
  graphical administration tool for PostgreSQL

pgadmin3-data/focal,focal 1.22.2-6build1 all
  graphical administration tool for PostgreSQL - documentation

phppgadmin/focal,focal 7.12.1+dfsg-1 all
  web-based administration tool for PostgreSQL

postgresql-12-pldebugger/focal 1:1.0-10-g2a298eb-2 amd64
  PostgreSQL pl/pgsql Debugger API
```

解决方案如下：

- 安装对应库的公钥

```
1 curl <https://www.pgadmin.org/static/packages\_pgadmin\_org.pub> | sudo apt-key add
2
```

- 创建库的配置文件

```
1 sudo sh -c 'echo "deb <https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) pgadmin4 main" > /etc/apt/sources.list.d/pgadmin4.list && apt update'
2
```

- 安装 pgadmin4 (desktop 和 web 模式, 只安装 web 模式则对应 pgadmin4-web, 只安装 desktop 模式则对应 pgadmin-desktop 模式)

```
1 sudo apt install pgadmin4
2
```

- 初始化配置, 使用/usr/pgadmin4/bin/setup-web.sh脚本



```
root@snodes6:/etc/postgresql/12/main# /usr/pgadmin4/bin/setup-web.sh
Setting up pgAdmin 4 in web mode on a Debian based platform...
Creating configuration database...
NOTE: Configuring authentication for SERVER mode.

Enter the email address and password to use for the initial pgAdmin user account:

Email address: [REDACTED]
Password:
Retype password:
pgAdmin 4 - Application Initialisation
=====
Creating storage and log directories...
We can now configure the Apache Web server for you. This involves enabling the wsgi module and configuring the pgAdmin 4 application to mount at /pgadmin4. Do you wish to continue (y/n)? y
The Apache web server is running and must be restarted for the pgAdmin 4 installation to complete. Continue (y/n)? y
Apache successfully restarted. You can now start using pgAdmin 4 in web mode at http://127.0.0.1/pgadmin4
```

之后可以通过‘[ip]/pgadmin4’访问 pgadmin, 用户名为配置时输入的邮箱账号, 密码为设置的密码。通过 register server 进行连接服务器中的数据库(在同一台服务器上运行的 pgadmin 和 postgresql, 连接时为本地连接, 地址为 127.0.0.1)。

2.3.3 数据表格构建

2.3.3.1 Topics

创建表格

```
1 create table topics(
2 id bigint not null primary key,
3 name text
4 );
```

导入数据库（带 csv 头部）

```
1 copy topics from '/extra/SocialComputing/codes/data/topics.csv' csv header
2
```

2.3.3.2 Events

创建表格

```
1 create table events(
2     id text not null primary key,
3     name text,
4     event_url text,
5     lat float,
6     lon float,
7     city text,
8     groupid bigint,
9     created timestamp with time zone,
10    visibility text,
11    time timestamp with time zone
12 );
```

对各种信息的处理比较复杂，使用脚本将数据导入数据库。其中 pd 代表 pandas 库。

```
1 def init_events(self):
2     filepath = FILE_CONFIG['events']
3     data = pd.read_csv(filepath, skipinitialspace=True,
4                         escapechar='\\\\\\', quotechar='"')
5     data['id'].astype('str')
6     data['joined'] = data['joined'].astype('int')
7     data['visited'].astype('int')
8     data['groupid'].astype('int')
```

```

9     cur = self.conn.cursor()
10
11    for row in track(data.iterrows(), description="inserting event...", total=len(data)):
12        c = row[1]
13
14        lat = 0.0 if c.lat == 'virtual' else float(c.lat)
15        lon = 0.0 if c.lon == 'virtual' else float(c.lon)
16
17        sql = ("insert into events(id, name, event_url, lon, lat, city, groupid, created,
18           visibility, time)"
19
20           ' values (\\'{}\\', \\'{}\\', \\'{}\\', {}, {}, \\'{}\\', {}, \\'{}\\', \\'{}\\'
21           ', \\'{}\\')'
22
23           ).format(c.id, c.test_name, c.event_url, lon, lat, c.city, c.groupid,
24
25           datetime.strftime(datetime.fromtimestamp(
26
27               c.created/1000), '%Y-%m-%d_%H:%M:%S'), c.visibility,
28
29           datetime.strftime(datetime.fromtimestamp(
30
31               c.time/1000), '%Y-%m-%d_%H:%M:%S'))"
32
33        cur.execute(sql)
34
35        self.conn.commit()

```

2.3.3.3 Members

创建表格

```

1 create table members(
2 id bigint primary key not null,
3 name text,
4 pos point,
5 lat float,
6 lon float,
7 link text
8 country text,
9 city text,
10 hometown text,
11 joined timestamp with time zone,
12 lang text,
13 visited timestamp with time zone

```

```
14 );
```

导入代码

```
1 def init_member(self):
2     # Init members table from csv file. Attention: replace '' with '!'
3     member_file = FILE_CONFIG['members']
4     data = pd.read_csv(member_file, skipinitialspace=True,
5                         escapechar='\\\\\\', quotechar='''')
6     data['id'].astype('int')
7     data['lat'].astype('float')
8     data['lon'].astype('float')
9     data['joined'] = data['joined'].astype('int')
10    data['visited'].astype('int')
11    cur = self.conn.cursor()
12    for row in track(data.iterrows(), description="inserting data", total=len(data)):
13        c = row[1]
14        sql = ("insert into members(id, name, link, lon, lat, country, city, hometown, joined,
15                                     lang, visited)"
16              ' values ({}, \\"{}\\", \\"{}\\", {}, {}, \\"{}\\", \\"{}\\", \\"{}\\",
17              \\"{}\\", \\"{}\\", \\"{}\\")'
18              ).format(c.id, c.test_name, c.link, c.lon, c.lat, c.country, c.city, c.hometown
19              ,
20                  datetime.strftime(datetime.fromtimestamp(
21                      c.joined/1000), '%Y-%m-%d_%H:%M:%S'), c.lang,
22                  datetime.strftime(datetime.fromtimestamp(
23                      c.visited/1000), '%Y-%m-%d_%H:%M:%S'))
24      cur.execute(sql)
25      self.conn.commit()
```

2.3.3.4 Group

创建表格

```

1 create table groups(
2     id bigint not null primary key,
3     name text not null,
4     urlname text,
5     rating float,
6     join_mode text,
7     lat float,
8     lon float,
9     country text,
10    city text,
11    created timestamp with time zone,
12    members int,
13    organizer bigint not null
14 );

```

导入数据

```

1 def init_groups(self):
2     filepath = FILE_CONFIG['groups']
3     data = pd.read_csv(filepath, skipinitialspace=True,
4                         escapechar='\\\\\\', quotechar='''')
5     data['id'].astype('int')
6     data['rating'].astype('float')
7     data['lat'].astype('float')
8     data['lon'].astype('float')
9     data['created'].astype('int')
10    data['members'].astype('int')
11    data['organizer'].astype('int')
12    cur = self.conn.cursor()
13    for row in track(data.iterrows(), description="inserting groups...", total=len(data)):
14        c = row[1]
15        sql = ("insert into groups(id, name, urlname, rating, join_mode, lat, lon, country,
16                               city, created, members, organizer)"
17               "values({}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}")
18               .format(*c))

```

```

17         c.id, c.test_name, c.urlname, c.rating, c.join_mode, c.lat, c.lon, str(
18             c.country).lower(), c.city,
19             datetime.strftime(datetime.fromtimestamp(
20                 c.created/1000), '%Y-%m-%d %H:%M:%S'),
21             c.members, c.organizer)
22     cur.execute(sql)
23     self.conn.commit()

```

2.3.3.5 Group

创建表格

```

1 create table groups(
2     id bigint not null primary key,
3     name text not null,
4     urlname text,
5     rating float,
6     join_mode text,
7     lat float,
8     lon float,
9     country text,
10    city text,
11    created timestamp with time zone,
12    members int,
13    organizer bigint not null
14 );

```

导入数据

```

1 def init_groups(self):
2     filepath = FILE_CONFIG['groups']
3     data = pd.read_csv(filepath, skipinitialspace=True,
4                         escapechar='\\\\\\', quotechar='''')
5     data['id'].astype('int')

```

```

6     data['rating'].astype('float')
7
8     data['lat'].astype('float')
9
10    data['lon'].astype('float')
11
12    data['created'].astype('int')
13
14    data['members'].astype('int')
15
16    data['organizer'].astype('int')
17
18    cur = self.conn.cursor()
19
20    for row in track(data.iterrows(), description="inserting groups...", total=len(data)):
21
22        c = row[1]
23
24        sql = ("insert into groups(id, name, urlname, rating, join_mode, lat, lon, country,
25               city, created, members, organizer)"
26
27               "values({}, '\\\\{}\\\\', '\\\\{}\\\\', {}, '\\\\{}\\\\', {}, {}, {}, '\\\\{}\\\\', '\\\\{}\\\\',
28               '\\\\{}\\\\', {}, {})").format(
29
30               c.id, c.test_name, c.urlname, c.rating, c.join_mode, c.lat, c.lon, str(
31
32                   c.country).lower(), c.city,
33
34                   datetime.strftime(datetime.fromtimestamp(
35
36                       c.created/1000), '%Y-%m-%d_%H:%M:%S'),
37
38                   c.members, c.organizer)
39
40        cur.execute(sql)
41
42        self.conn.commit()

```

2.3.3.6 EventHost

创建表格

```

1 create table eventhosts(
2     eventid text not null,
3     memberid bigint not null,
4     primary key(eventid, memberid)
5 );

```

导入到数据库

```

1 copy eventhosts from '/extra/SocialComputing/codes/data/eventhosts.csv' csv header

```

2.3.3.7 GroupTopics

创建表格

```
1 create table grouptopics(
2     groupid bigint not null,
3     topicsid bigint not null,
4     primary key(groupid, topicsid)
5 );
```

导入到数据库

```
1 copy grouptopics from '/extra/SocialComputing/codes/data/grouptopics.csv' csv header
```

2.3.3.8 MemberTopics

创建表格

```
1 create table membertopics(
2     memberid bigint not null,
3     topicsid bigint not null,
4     primary key(memberid, topicsid)
5 );
```

导入到数据库

```
1 copy membertopics from '/extra/SocialComputing/codes/data/membertopics.csv' csv header
```

2.3.3.9 RSVP

创建表格

```
1 create table rsvps(
2     eventid text not null,
3     memberid text not null,
```

```
4 mtime timestamp with time zone,
5 primary key(eventid, memberid)
6 );
```

导入到数据库

```
1 def init_rsvp(self):
2     filepath = FILE_CONFIG['rsvp']
3     data = pd.read_csv(filepath, skipinitialspace=True,
4                         escapechar='\\\\\\', quotechar='''')
5     data['eventid'].astype('str')
6     data['mtime'].astype('int')
7     cur = self.conn.cursor()
8     for row in track(data.iterrows(), description="inserting rsvp...", total=len(data)):
9         c = row[1]
10        sql = ("insert into rsvps(eventid, memberid, mtime) values(\\'{}\\', {}, \\'{}\\')").format(
11            c.eventid, c.memberid,
12            datetime.strftime(datetime.fromtimestamp(c.mtime/1000), '%Y-%m-%d %H:%M:%S'))
13        cur.execute(sql)
14    self.conn.commit()
```

2.3.3.10 MemberGroups

创建表格

```
1 create table membergroups(
2     memberid bigint not null,
3     groupid bigint not null,
4     join_time timestamp with time zone,
5     primary key(memberid, groupid)
6 );
```

导入到数据库

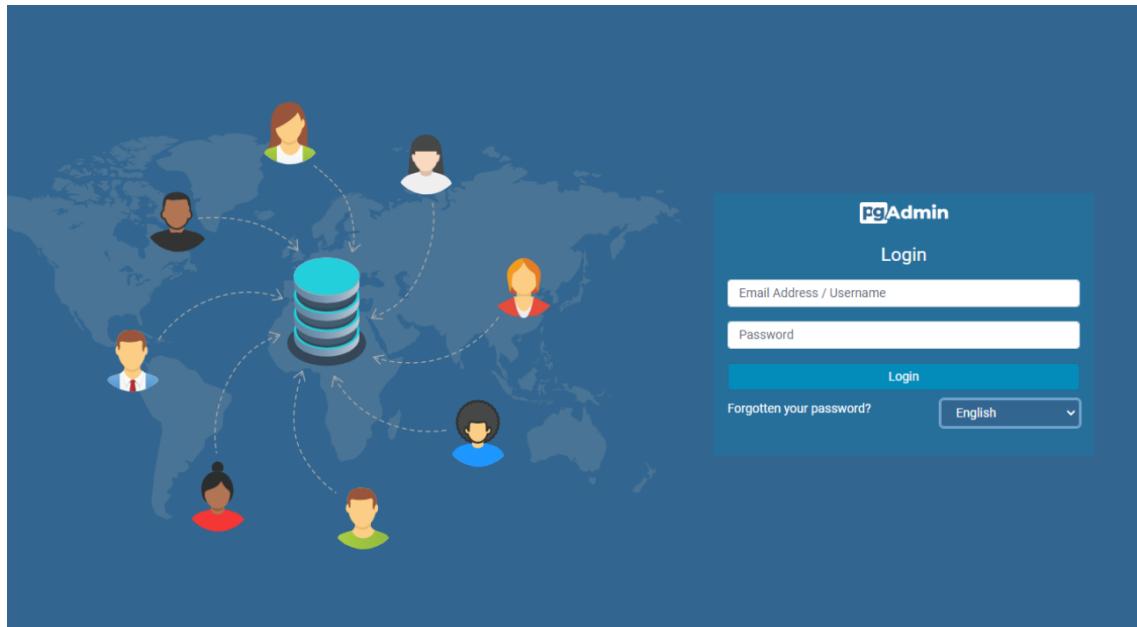
```

1 def init_member_groups(self):
2     filepath = FILE_CONFIG['member_groups']
3     cursor = self.conn.cursor()
4     with open(filepath, 'r+') as f:
5         data = json.load(f)
6         for memberid, values in track(data.items(), description='insert into membergroups'):
7             for index, groupid in enumerate(values[0]):
8                 sql = ("insert into membergroups(memberid, groupid, join_time) values ({}, {}, "
9                        "\\"{}\\\"").format(int(memberid), int(values[0][index]), datetime.strftime(datetime.
10                           fromtimestamp(
11                               int(values[1][index])/1000), '%Y-%m-%d %H:%M:%S'))
12             cursor.execute(sql)
13             # print(f"sql: {sql}")
14             self.conn.commit()

```

2.3.4 数据库可视化界面

配置完 pgadmin 后，可以通过浏览器进行可视化访问数据库



构建完成之后的数据表如下

The screenshot shows a database interface with a sidebar titled "Tables (9)". Below the title, there is a list of nine tables, each represented by a blue folder icon:

- > eventhosts
- > events
- > groups
- > grouptopics
- > membergroups
- > members
- > membertopics
- > rsvps
- > topics

此时可以通过 SQL 语句在此界面上对数据进行处理，示例如下

The screenshot shows a database query editor with the following details:

Query Editor Query History

```
1 select * from members limit 10
```

Data Output Explain Messages

	id [PK] bigint	name text	pos point	link text	country text	city text	hometown text	joined timestamp with time zone	lang text	visited timestamp with time zone
1	5700377	Gerry	[null]	http://www.meetup.com/members/5700377	us	Salem	unkown	2007-11-23 23:58:42+00	en_US	2013-03-06 00:46:46+00
2	5700656	billy g	[null]	http://www.meetup.com/members/5700656	us	East Weymouth	weymouth	2007-11-24 01:06:40+00	en_US	2012-09-21 11:52:48+00
3	57006922	Jed Morris	[null]	http://www.meetup.com/members/57006922	us	Bedford	unkown	2012-08-02 01:53:54+00	en_US	2013-05-11 01:02:05+00
4	57006942	Nate FM	[null]	http://www.meetup.com/members/57006942	us	Cambridge	unkown	2012-08-02 01:54:04+00	en_US	2013-05-24 03:48:54+00
5	5700781	Sally	[null]	http://www.meetup.com/members/5700781	us	Milton	Milton	2007-11-24 01:36:12+00	en_US	2013-05-11 12:40:04+00
6	57008742	Thomas	[null]	http://www.meetup.com/members/57008742	us	Boston	unkown	2012-08-02 02:10:28+00	en_US	2012-08-26 21:36:23+00
7	57009282	Catherine_/_	[null]	http://www.meetup.com/members/57009282	us	Natick	Natick, MA	2012-08-02 02:15:16+00	en_US	2013-05-16 03:12:55+00
8	57010472	Mareike	[null]	http://www.meetup.com/members/57010472	us	North Attleboro	unkown	2012-08-02 02:24:55+00	en_US	2013-05-19 22:11:29+00
9	5701253	Debabrata	[null]	http://www.meetup.com/members/5701253	us	Boston	unkown	2007-11-24 02:47:24+00	en_US	2013-04-18 19:12:50+00
10	5701535	Susana	[null]	http://www.meetup.com/members/5701535	us	Mansfield Center	Mansfield Center	2007-11-24 03:24:19+00	en_US	2013-04-07 18:29:14+00

图 3: 用兴趣统计缩放

2.3.5 数据出库

在这一步中，主要功能为从数据库中提取有用的信息，然后以约定好的接口和数据格式提供给训练网络。

2.3.5.1 通用的查询接口

为了方便对单表中数据的查询，实现了一个通用的查询接口，支持各种筛选条件、排序以及限制结果条目数，以列表的形式返回查询结果。接口数据格式如下

```
1 def query(self, tablename: str, attrs=[], conditions=[]) -> list:
2     """query from meetup data, returning results as a list.
3     Remember to add \\' when query string
4
5     Args:
6         tablename (str): table name
7         attrs (list): attrbuites list
8         conditions (list): a list of conditions
9     """
10
```

对此通用接口使用 unittest 编写了单元测试，内容如下

```
1 import unittest
2 import pprint
3 from database import DataBase
4
5 class TestDataBase(unittest.TestCase):
6     def test_query(self):
7         db = DataBase()
8         test_case = [
9             {'tablename': 'groups', 'attrs': ['id', 'name'], 'conditions': [
10                 'rating > 4.3', 'members > 100', 'limit 10']},
11             {'tablename': 'groups', 'attrs': [], 'conditions': [
12                 'urlname = \\'science-88\\'', 'limit 20']},
13             {'tablename': 'groups', 'attrs': [], 'conditions': [
14                 'join_mode = \\'open\\'', 'lat > 42', 'order by id', 'limit 10']}
15         ]
16         for index, case in enumerate(test_case):
17             print(f'\n=====  Test case {index}  start! =====\n')
18             result = db.query(case['tablename'],
19                               case['attrs'], case['conditions'])
```

```

20         pprint.pprint(result)
21     print(f'\n===== Test case {index}  fininsh! =====\n')
22
23 if __name__ == '__main__':
24     unittest.main()

```

2.3.5.2 输出数据格式

输出数据主要包含三种信息，结果均按照二维列表存储。

1. Member 和 Member 之间的边的关系，有好友关系则存在对应的双向边。
2. Group 的 label，即 Group 和 Member 之间的关系，以 0/1 表示 Group 和 Member 有/无关系。
3. Group 的 feature，即 Group 和 Topic 之间的关系，以 0/1 表示 Group 和 Topic 有/无关系。

2.3.5.3 对 Member 数量的限定

现有机器显存大小不支持对所有数据的训练，因此需要对 member 数量进行限定。数据出库时提供了 membernum 参数，用以支持固定 member 数量的小规模训练，此时 group 和 topic 对应的依然为全量数据。

2.3.5.4 对 Topics 的筛选定

清洗完的数据一共有约 18000 个 topic，导致每个 feature 特别稀疏，观察 topic name 发现，很多 topic 之间存在包含关系或者非常接近，因此需要对 topic 做进一步的精简以提高结果准确程度。精简 topic 时，如果两个 topic name 之间存在相同关键词，则认为两个 topic 可以归为一类，从其中选择一个代表即可，在筛选 topic 时，尽量选择 name 中单词数较少的作为代表（容易理解，词越少，越代表这个 topic 涵盖范围比较大）。筛选算法如下：

- 1 输入：topics，以列表的形式存储，每个元素为 [id, name] 的数据对
- 2 输出：topic_info，以 id 为键，表示从一组等价 topic 之间选出的代表 topic 的 id，value 为两个 list，第一个 list 为这一组 topic 的 id，第二个 list 存储这一组 topic 的 name 对应的 word set，id 和 name 之间为一一对应关系。
topic_mapping，key 和 value 均为 topic id，表示 topic 和其所在等价组中代表 topic 的映射关系
- 3 对 topics 的一份拷贝，将其中 name 的所有字母转化为小写格式，以空格分词，构造单词集合，并对 topics 按照单词集合的大小进行升序排列

```

4 初始化topic_info，每个topic的代表都是自身，value对应的id和name列表中仅包含自身对应的值，元素按照对应
5 单词集合大小进行升序排列；
6
7 index = 0, delete = False
8
9 while True:
10
11     if index > len(topic_info): break;
12     for prev_index in [0 ... index-1]:
13         if index对应的单词集合和prev_index对应的集合有交集：
14             将index对应的id和name加入topic_info[prev_index]中的两个列表中
15             # 设置对应的id映射关系
16             topic_mapping[topic_info[index][1][0]] = topic_info[prev_index][1][0]
17             delete = True
18             break
19         if delete:
20             del topic_info[index]
21             delete = False
22         else:
23             index += 1

```

经过筛选之后，库中 18000 个 topic 可以被精简为 4000 左右，观察精简之后的映射关系，结果符合常理，topic_info 部分内容如下。

```

    > 0: {'sales'}
    > 1: [16, 18700, 19256, 21822, 26291, 65207, 84712, 1216452, 20554, ...]
    > 2: ['Sales', 'Sales Training', 'Sales Professionals', 'Sales Strategies', 'Direct Sal...
        len(): 3

    > 0001: [{'baseball'}, [80, 1033, 4334, 23831, 98062, 98063, 6298, 23832, 52540, ...], [...]
    > special variables
    > function variables
    > 0: {'baseball'}
    > 1: [80, 1033, 4334, 23831, 98062, 98063, 6298, 23832, 52540, 25356]
    > 2: ['Baseball', 'Fantasy Baseball', 'Baseball Primer', 'Baseball Networking', 'Baseba...
        len(): 3

    > 0002: [{'nascar'}, [84, 79130, 17515, 79129, 80006], ['NASCAR', 'Nascar Fans', 'Auto Ra...
    > special variables
    > function variables
    > 0: {'nascar'}
    > 1: [84, 79130, 17515, 79129, 80006]
    > 2: ['NASCAR', 'Nascar Fans', 'Auto Racing: NASCAR', 'Watch Nascar Races Together', 'N...
        len(): 3

```

2.3.5.5 对城市的筛选

考虑到实际情况，member 参与距离较远的 group 的意愿会比较低，这会导致在 label 矩阵上，member 对应的列的值大部分元素为 0，因此直观上的想法是对 group，对 topic，对 member 在城市的维度上进行进一步的筛选。

在城市筛选时，按照每个城市所拥有的 group 数目进行降序排序，然后从其中按顺序取出若干个城市的 group。在 member 筛选时，过滤掉参与 group 过少的情况。在对 topic 进行筛选时，只考虑这些 group 和这些 member 涉及到的 topic，过滤掉 group 和 member 均不感兴趣的 topic。

数据出库时提供了两个参数可以设定，min_num_groups 代表每个成员至少应该参与的 group 数目，citynum 表示按 group 数目降序排列时，从中选择的城市的数量。

使用以下的 SQL 语句从数据库中获得对应的信息

```

1 sql = ("select groups.city, "
2         "array_agg(distinct groups.id) groupids, "

```

```
3      "array_agg(distinct members.id) memberids, "
4      "array_agg(distinct topicsid) topicids, "
5      "array_agg(distinct topics.name) topicnames, "
6      "count(distinct groups.id) groupnum, "
7      "count(distinct members.id) membernum "
8      "from groups left join members on groups.city=members.city "
9      "left join membertopics on members.id=membertopics.memberid "
10     "left join topics on membertopics.topicsid=topics.id "
11     "where topicsid is not null and "
12     "topics.name is not null "
13     "and members.id in "
14     "  (select memberid from "
15     "    (select membergroups.memberid, count(distinct groupid) groupnum "
16     "     from membergroups group by memberid "
17     "     order by groupnum desc "
18     "   ) "
19     "   as membergroupnum where groupnum >= {} "
20     " ) "
21     "group by groups.city order by groupnum desc limit {}".format(min_num_groups,
citynum)
```

查询得到的（部分）结果如下

```

✓ 0: ('Boston', [2348, 12763, 29048, 32412, 33097, 55264, 56189, 57664, 59186, ...], [58579, 77378, 104717, 1059
  > special variables
  > function variables
    0: 'Boston'
  > 1: [2348, 12763, 29048, 32412, 33097, 55264, 56189, 57664, 59186, 61202, 71144, 86694, 87187, 111392, ...]
  > 2: [58579, 77378, 104717, 105921, 110544, 127163, 181839, 182104, 206220, 209991, 253319, 286346, 401024, 414
  > 3: [16, 57, 58, 60, 62, 73, 75, 76, 78, 79, 80, 83, 85, 86, ...]
  > 4: ['12 Step Recovery', '20-30 somethings', '20 Million Loud', '20s', "20's, 30's & 40's", "20's & 30's Social"
    5: 184
    6: 1770
    len(): 7
  ✓ 1: ('Cambridge', [21458, 23495, 63770, 81154, 87071, 119696, 145862, 165993, 169595, ...], [237353, 309719, 46
    > special variables
    > function variables
      0: 'Cambridge'
    > 1: [21458, 23495, 63770, 81154, 87071, 119696, 145862, 165993, 169595, 183855, 216208, 252130, 278345, 302546
    > 2: [237353, 309719, 461418, 530199, 568269, 647527, 839148, 906018, 907022, 930721, 1098472, 1107563, 1701364
    > 3: [16, 57, 58, 60, 72, 75, 79, 80, 83, 85, 86, 87, 88, 89, ...]
    > 4: ['20-30 somethings', '20s', "20's, 30's & 40's", "20's & 30's Social", "20's and 30's", '20 somethings',
      5: 84
      6: 787
      len(): 7
  ✓ 2: ('Manchester', [160942, 447070, 527156, 594346, 643770, 1101339, 1292928, 1316784, 1322739, ...], [105242,
    > special variables
    > function variables
      0: 'Manchester'
    > 1: [160942, 447070, 527156, 594346, 643770, 1101339, 1292928, 1316784, 1322739, 1332912, 1413769, 1479885, 14
    > 2: [105242, 1192026, 2399923, 2404346, 2527759, 2734556, 3033585, 3189013, 3384225, 3397763, 3641714, 4194127
    > 3: [57, 83, 89, 90, 91, 92, 95, 99, 120, 123, 124, 125, 127, 130, ...]
    > 4: ['30s and 40s', "70's Music", '80-10-10', '9/11 Questions', '9/11 Truth', 'Abraham Hicks', 'A Course In Mi
      5: 21
      6: 86
      len(): 7

```

2.3.5.6 训练数据的划分

Group id 和该 group 建立的时间相关，因此在查询时按照 group id 升序排列，按一定比例取前面一部分作为训练集，后一部分作为测试集。时间上来说，测试集的时间靠后。

2.3.5.7 主要接口

提供了一个主要接口用于提供训练和测试数据，各参数和对应说明见注释。

```

1 def exampleDataFrom(self, membernum=-1, percent=0.8, simple_topics=False, use_top_city=True,
2   citynum=5, min_group_num=3):
3   """get train data and prediction data

```

```
3
4     Args:
5         membernum (int, optional): member num, invalid when use_top_city=True. Defaults to
6         -1, which means select all members.
7
8         percent (float, optional): train data percent. Defaults to 0.2.
9
10        simple_topics (bool, optional): whether to use reduced topics. Defaults to False.
11
12        use_top_city (bool, optional): whether get data from top member num cities. Defaults
13        to True.
14
15        citynum (int, optional): city number when use_top_city is true. Defaults to 5.
16
17        mini_group_num (int, optional): minimum group number a member supposed to participated
18        . Defaults to 3.
19
20
21    Returns:
22        tuple : [train_dataset, edges], [prediction_dataset, edges]
23
24    """
25
```

3 实验方案

3.1 实验总体思路

我们将网络中的预测社团参与问题视作分类问题。通过对每个用户 u_i 和每个社团 g_k 建立相似性函数 $f(u_i, g_k)$ 和阈值 $h(u_i, g_k)$ 来预测独立的参与意愿 $s_{i,k}$.

如果 $f(u_i, g_k) \geq h(u_i, g_k)$, 则 $s_{i,k} = 1$. 如果 $f(u_i, g_k) < h(u_i, g_k)$, 则 $s_{i,k} = 0$

3.1.1 符号定义

Table 3: Mathematical Notations.

SYMBOL	DESCRIPTION
$U = \{u_i\}$	the set of users
$G = \{g_k\}$	the set of groups
p_i	preference vector for u_i
a_k	attributes vector for g_k
w_{ij}	social connection strength from u_i to u_j
$f_{i,k}$	intention for u_i to attend g_k
$h_{i,k}$	threshold for u_i to attend g_k
$s_{i,k}$	attendance for u_i to g_k

3.1.2 $f(u_i, g_k)$

某用户对某社团的初始偏好由两者偏好特征向量的相似性决定

$$f(u_i, e_k) = \text{Cosine}(a^T, p^T) \quad (1)$$

3.1.3 $h(u_i, g_k)$

由于网络中的社交关系, 周围人的决定会影响自己的决定阈值 (比如, 周围的人都参加某个事情, 可能自己对参与此事就没有这么反感)。考虑到现实生活中, 一个人周围的人对其自身的影响具有累加效应 (即从众现象), 我们设计了如下的阈值函数:

$$h(u_i, g_k) = \prod_{j \in N_i} [1 - \mathcal{I}(f_{j,k} - h_{j,k}) \cdot w_{ji}] \quad (2)$$

其中, 我们使用 Sigmoid 函数将预测的结果映射到 $[0, 1]$ ([不加入, 加入]) 之间:

$$\mathcal{I}(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

3.1.4 网络训练

在训练的过程中，我们的输入数据包括社团集合 $\mathbf{G} = \{g_k\}$ ，用户集合 $\mathbf{U} = \{u_i\}$ 以及用户在过去所加入社团信息的集合 $\mathbf{S} = \{s_{i,k}^0\}$ 。同时对于每一个社团 g_k ，我们也有对应的话题信息 \mathbf{a}_k 。在这一阶段中，我们的目标是学习每个用户的偏好特征向量 p_i ，以及在社交网络中用户之间的联系强弱，即网络的边的权重 $\{w_{ij}\}$ 。

Algorithm 1 Iterative Solution for Training Stage

Input: target user group $\mathbf{U} = \{u_i\}$, group set $\mathbf{G} = \{g_k\}$ and attendance records $\{s_{i,k}^0\}$

Store: group attributes \mathbf{a}_k for each $g_k \in \mathbf{G}$

Output: users' profile $\langle \mathbf{p}_i, h_{i,0} \rangle$ and social strength w_{ij}

```

1: Iteration = True;
2: while Iteration do
3:   Iteration = False;
4:   for  $u_i \in \mathbf{U}, g_k \in \mathbf{G}$  do
5:     update  $\langle \mathbf{p}_i, h_{i,0} \rangle$  and  $\{w_{ij}\}$  until convergence;
6:     update  $f_{i,k}, h_{i,k}$  based on Equation 2;
7:     update  $s_{i,k}$  as  $\mathcal{I}(f_{i,k} - h_{i,k})$ ;
8:     if  $s_{i,k}$  changed then
9:       Iteration = True;
10:    end if
11:   end for
12: end while

```

在这一过程中，我们希望最小化的损失函数为：

$$\arg \min_{\mathbf{p}, h_0, w} \sum_{u_i \in U} \sum_{g_k \in G} [s_{i,k}^0 - \mathcal{I}(f_{i,k} - h_{i,k})]^2 \quad (4)$$

其中 $s_{i,k}^0$ 表示用户实际的参加情况。

为了达成这一目标，我们假设用户某一轮（一段时间）中选择仅仅被上一轮中与之相关朋友的决策所影响，设计了如下的迭代函数：

$$F^t(U, G) = \sum_{u_i \in U} \sum_{g_k \in G} [s_{i,k}^0 - \mathcal{I}(f_{i,k}^t - h_{i,k}^t)]^2 \quad (5)$$

其中

$$h_{i,k}^t = \prod_{j \in N_i} [1 - \mathcal{I}(f_{j,k}^{t-1} - h_{j,k}^{t-1}) \cdot w_{ji}^t] \quad (6)$$

3.1.5 网络验证

从训练的过程中获取了用户的社交关系以及兴趣偏好后，我们使用神经网络对用户将来是否加入某个社团进行预测。在这里，我们选择基于公式 2，通过不断的迭代收敛获取稳定的预测结果。具体算法如 Algorithm 2 所示。

Algorithm 2 Iterative Solution for Test Stage

Input: target user \mathbf{U} and group g_k with attributes \mathbf{a}_k ;
Store: users' profiles $\langle \mathbf{p}_i, h_{i,0} \rangle$ and connection weights w_{ij} ;
Output: $s_{i,k}$ for each $u_i \in \mathbf{U}$

- 1: **for** $u_i \in \mathbf{U}$ **do**
- 2: calculate $f_{i,k}$ for each u_i ;
- 3: **end for**
- 4: Iteration = True;
- 5: **while** Iteration **do**
- 6: Iteration = False;
- 7: **for** $u_i \in \mathbf{U}$ **do**
- 8: update $h_{i,k}$ based on Equation 2;
- 9: update $s_{i,k}$ based on $f_{i,k}$ and $h_{i,k}$;
- 10: **if** $s_{i,k}$ changed **then**
- 11: Iteration = True;
- 12: **end if**
- 13: **end for**
- 14: **end while**
- 15: **return** $\mathbf{S} = \{s_{i,k}\}$;

3.2 图神经网络的具体实现

使用 pytorch 的图神经网络库 PyG (PyTorch Geometric) 实现

3.2.1 创建消息传递网络

将卷积运算符推广到不规则域通常表示为邻域聚合或消息传递方案。

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right) \quad (7)$$

其中 $\mathbf{x}_i^{(k-1)} \in \mathbb{R}^F$ 表示第 $(k-1)$ 层第 i 个节点的特征， $\mathbf{e}_{j,i} \in \mathbb{R}^D$ 表示从节点 j 到节点 i 的边的特征， \square 表示具有置换不变性的可微函数，如 sum. mean 或 max 等。我们使用 PyG 提供的一种 MESSAGE PASSING 的模板来实现对应的网络。

```

1  class DSI(MessagePassing):
2
3      def __init__(self, NodeNum, TopicNum, EdgeNum, device, BatchSize):
4          super().__init__(aggr="mul") # "mul" aggregation (Step 5).
5
6          self.nodePreffferVector = nn.Parameter(
7              torch.rand(NodeNum, TopicNum, requires_grad=True)
8          )
9
10         self.cos = nn.CosineSimilarity(dim=1, eps=1e-6)
11
12         self.Wij = nn.Parameter(torch.rand(EdgeNum, 1, requires_grad=True)) # W_{ij}
13
14         self.Sig = torch.nn.Sigmoid()
15
16         self.device = device
17
18         self.NodeNum = NodeNum
19
20         self.TopicNum = TopicNum
21
22         self.BatchSize = BatchSize
23
24
25     def forward(self, trainGroup_batch, edge_index, threshold_H):
26
27         # edge_index has shape [2, E]
28
29         self.BatchSize = trainGroup_batch.size()[0]
30
31         edge_index = BatchExpand(edge_index, self.BatchSize)
32
33         edge_index, _ = remove_self_loops(edge_index)
34
35
36         # print(trainGroup_batch)
37
38         tmpCosInput = trainGroup_batch[0].expand(self.NodeNum, self.TopicNum)
39
40         for i in range(self.BatchSize - 1):
41
42             tmpCosInput = torch.cat(
43
44                 (
45
46                     tmpCosInput,
47
48                     trainGroup_batch[i + 1].expand(self.NodeNum, self.TopicNum),
49
50                 ),
51
52                 0,
53
54             )
55
56
57         expandNodePreffferVector = torch.cat(
58
59             [self.nodePreffferVector] * self.BatchSize, 0
60
61         )
62
63         f = self.cos(tmpCosInput, expandNodePreffferVector) # [nodeNum * 1]

```

```

36     fMinusH = f - torch.cat([threshold_H] * self.BatchSize, 1)
37
38     # Step 4-5: Start propagating messages.
39
40     return self.propagate(edge_index, x=fMinusH.transpose, f=f)
41
42     def message(self, x_j, f):
43
44         # x_j has shape [E, out_channels]
45
46         L = self.Sig(x_j)
47
48         tmpCat = torch.cat((L, torch.cat([self.Wij] * self.BatchSize)), 1)
49
50         LW = torch.prod(tmpCat, 1)
51
52         x_j = (1 - LW).view(1, -1).t()
53
54     return x_j
55
56
57     def update(self, aggr_out, f):
58
59         # aggr_out has shape [N, out_channels]
60
61         ht = aggr_out.t()
62
63         return [self.Sig(f - ht), ht.detach()]

```

3.2.2 使用 PyG 的 mini-batch

为了充分利用 GPU 资源加速 GNN 的训练，我们采用了 pyg 的 mini-batch。其核心思想是将多个图的邻接矩阵拼接成一个大的邻接矩阵 A，使 A 可以看做是近似的对角矩阵，然后采用稀疏图的存储方式进行存储和计算：

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_n \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_n \end{bmatrix}. \quad (8)$$

在这一过程中，GNN 的相关算子并不需要被修改，同时由于稀疏图的特性，内存开销也得以减小，提升了计算效率。

```

1 for epoch in track(
2     range(N_EPOCHS), total=N_EPOCHS, description="epoch"
3 ): # loop over the dataset multiple times
4     testNum = 0
5     correctNum = 0

```

```

6     for id_batch, (trainGroup_batch, label_batch) in enumerate(dataloader):
7         trainGroup_batch = trainGroup_batch.to(device)
8         label_batch = label_batch.reshape(1, nodeNum * label_batch.size()[0])
9         label_batch = label_batch.to(device)
10        optimizer.zero_grad()
11        [predict, tmpthreshold_H] = net(trainGroup_batch, edge_index, threshold_H)
12        threshold_H = meanBatchOut(tmpthreshold_H)
13        loss = criterion(predict, label_batch)
14        loss.backward(retain_graph=True)
15        optimizer.step() # Does the update

```

3.3 训练过程

3.3.1 样本不均衡的问题及应对方法

1. 平衡原始数据
 - (a) 地域聚合: 只保留同一个城市的用户和社团信息。
 - (b) 数据精简: 剔除参加 Group 数小于 3 的用户 (因相关信息过少, 难以建立精准画像)。
2. 采用 Focal Loss

Focal Loss 定义为调变因子 (modulating factor) 乘以原来的交叉熵损失:

$$\text{FL}(p_t) = -(1 - p_t)^\gamma \log(p_t) \quad (9)$$

其中调变因子随着置信度的增加而衰减为 0, 从而实现在训练期间自动降低简单样本的贡献, 并快速将模型集中在困难样本上。

4 结果分析

4.1 实验环境

系统版本: Ubuntu 20.04.4 LTS

GPU: GeForce GTX 1080 Ti, 12GB 显存

python: 3.8.10

torch: 1.11.0+cu113, torchaudio: 0.11.0+cu113, torchvision: 0.12.0+cu113

4.2 损失函数选择

4.2.1 MSE

实验首先考虑使用公式4的 MSE 作为损失函数进行训练，得到训练过程的 LOSS 变化如图4所示：

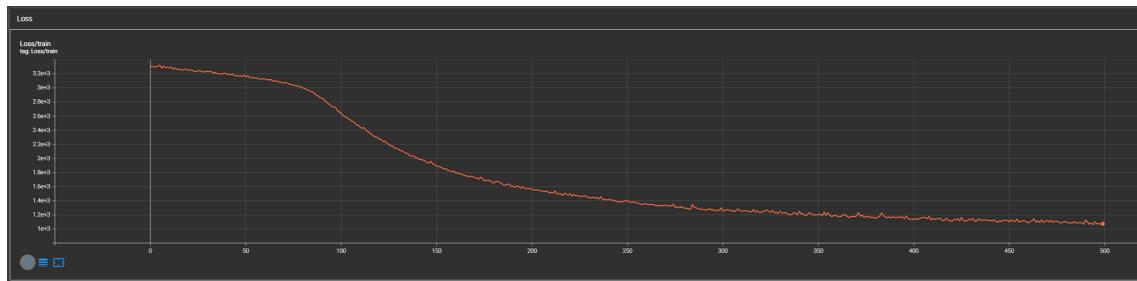


图 4: 使用 MSE 作为损失函数训练过程 LOSS 变化图

通过 LOSS 变化曲线可以看出，模型在训练中进行了不错的收敛，但在使用模型对测试集进行预测的过程中，发现了样本不均匀导致模型倾向于预测用户不加入社团的情况（将在下一部分详细阐述），如图5所示：

```

repetitionSocialNetwork on □ main [!] took 1m59s
> python .\src\main.py predict
nodeNum,edgeNum,topicNum,groupNum predictGroupNum 1000 46624 4422 626 157
skip train
cuda:0
32000
64000
96000
128000
157000
test accuracy: 0.941236
positiveLabelNum: 53 negativeLabelNum: 156947 ratio: 0.000338

```

图 5: 使用 MSE 作为损失函数在测试集上验证结果图

4.2.2 加权交叉熵

为了缓解样本不均衡问题对于预测结果的影响，小组考虑使用加权交叉熵作为损失函数，但是模型训练结果不够理想，损失函数不断波动，不进行收敛，结果如图6所示。不进行收敛的原因为：加权交叉熵需要合理设置初始的权重，0, 1label 对应的权重开始选择需要和样本成的反比，但人为设定初始权重的方式不够灵活且难以收敛。

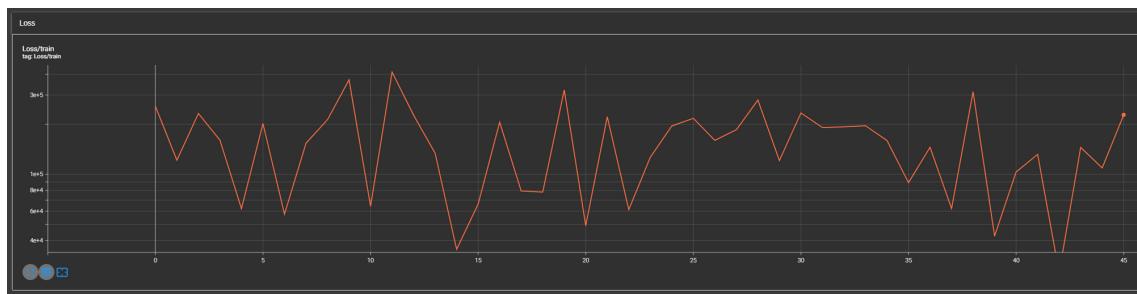


图 6: 使用加权交叉熵作为损失函数训练过程 LOSS 变化图

4.2.3 Focal Loss

由 3.3.1 节公式9Focal Loss 定义为调变因子乘以原来的交叉熵，由于调变因子的加入，调变因子会逐渐下降趋近于 0，这样就可以综合加权交叉熵不收敛的特性，将模型快速集中在困难样本上，突出困难样本对于模型训练的贡献程度，达到收敛的效果。

下图7也很好地展示了 Focal Loss 损失函数通过调变因子对加权交叉熵进行调整达到的训练过程 LOSS 变化。

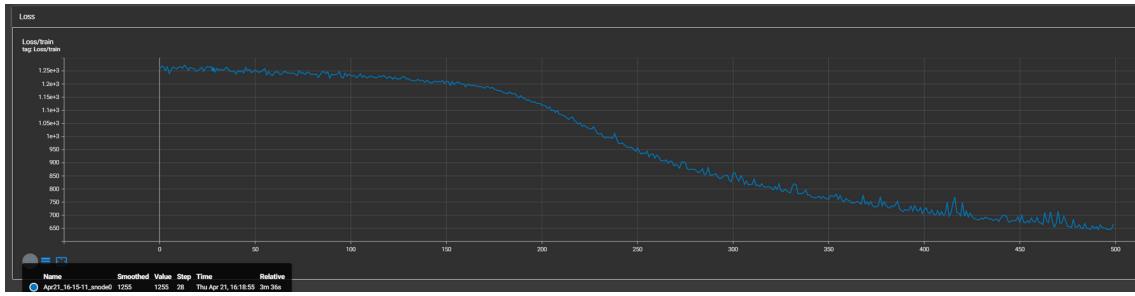


图 7: 使用 Focal Loss 作为损失函数训练过程 LOSS 变化图

4.3 实验结果

4.3.1 初步实验结果

通过使用 MSE 作为损失函数, 对使用预处理的数据对网络进行训练, 训练过程的 PR 曲线如图8所示。

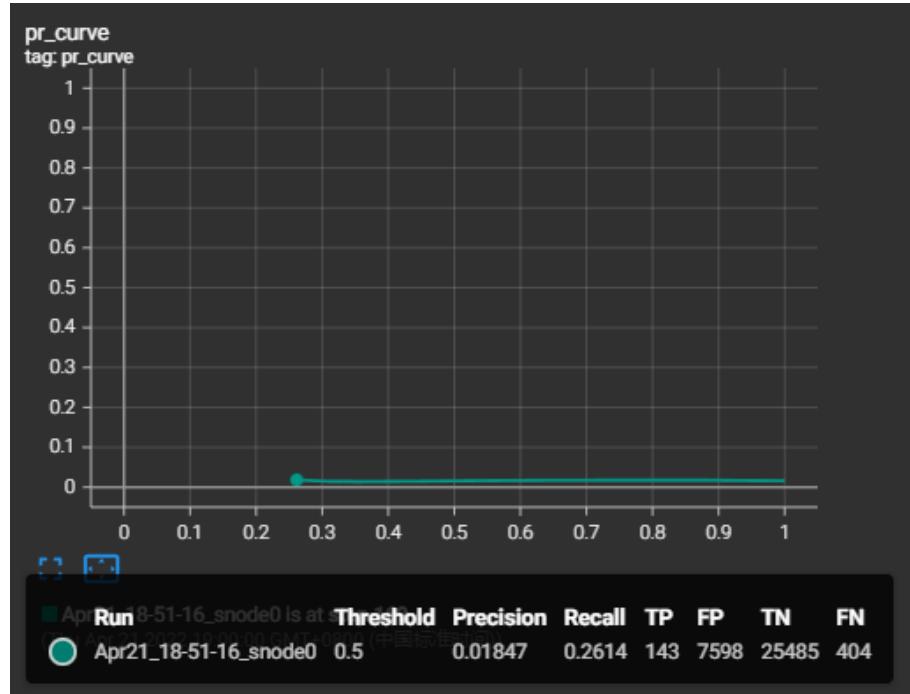


图 8: 使用 MSE 作为损失函数的训练过程 PR 曲线图

同时可以得到对应的混淆矩阵 TP, TN, FP 和 FN 的结果如表1所示:

真实情况	预测结果	
	正例	反例
正例	143(TP)	404(FN)
反例	7598(FP)	25485(TN)

表 1: 使用 MSE 作为损失函数的混淆矩阵

可以计算得到准确率:

$$\text{accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) = (143 + 25485) / (143 + 25485 + 7598 + 404) = 0.76$$

训练模型在划分的测试集上的预测结果如图5所示。可以看到，测试结果的准确率高达 0.94，甚至高于模型训练过程中的准确率，模型发生了过拟合。同时，positive label 的数量仅为 53，negative label 的数量高达 15 万，样本比例严重不均衡，考虑为造成测试集准确率过高的原因。

4.3.2 地域聚合

对初步实验结果分析，PR 曲线过于不理想且在测试集上的准确率过高，可以看到由于样本正反例极不平衡，训练出的模型尽可能地预测用户不加入 group 可以达到更高地准确率。小组对造成这种现象的原因进行分析后，发现大多数的人只会加入几个 group 中，但是模型采用了 800 个 group，这样就会造成模型更倾向于预测用户不参加 group 来获得更低的惩罚代价。

针对发现的原因，小组认为可以先按照地区筛选出一批 group 和用户，这样 group 和用户就会集中起来，不会出现向几百个 group 中判断用户是否加入的情况。在上述思考的基础上，在数据预处理部分，增加了对于同一城市数据 members, groups, topics 的筛选，实现 group 和用户的集中。

使用重新预处理后的数据，重新训练网络模型，得到的 PR 曲线和混淆矩阵如图9和表2所示。

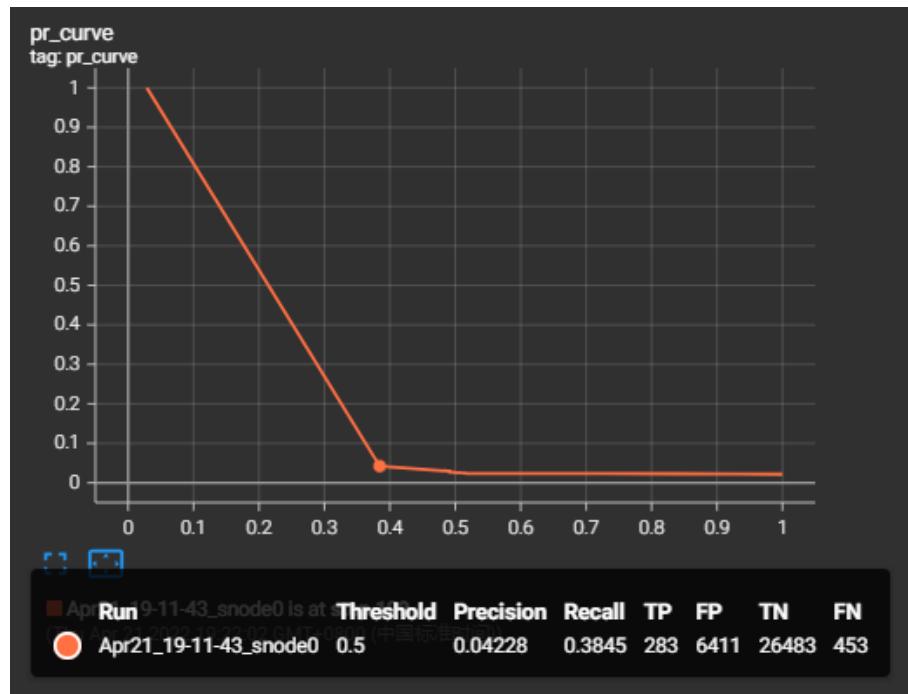


图 9: 地域聚合后的训练过程 PR 曲线图

真实情况	预测结果	
	正例	反例
正例	283(TP)	453(FN)
反例	6411(FP)	26483(TN)

表 2: 地域聚合后的混淆矩阵

可以计算得到准确率:

$$\text{accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) = (283 + 26483) / (283 + 26483 + 6411 + 453) = 0.80$$

训练模型在划分的测试集上的预测结果如图10所示。与上一节的测试集上的准确率相比，准确率相对下降，但是换来了模型更为强大的泛化能力，能够对用户 group 进行较为合理预测，而不是仅仅为了降低惩罚预测不加入 group 中。

```
epoch: 491, loss: 2255.582795
epoch: 492, loss: 2216.870850
epoch: 493, loss: 2139.109863
epoch: 494, loss: 2217.358887
epoch: 495, loss: 2215.688721
epoch: 496, loss: 2193.150391
epoch: 497, loss: 2429.840820
epoch: 498, loss: 2284.553467
epoch: 499, loss: 2316.420410
epoch ████████████████████████████████████████████████████████████████████████████████ 100% 0:00:00
predict
cuda:0
84904
169808
254712
339616
392681
test accuracy: 0.855957
positiveLabelNum: 89 negativeLabelNum: 392592 ratio: 0.000227
```

图 10: 地域聚合后在测试集上验证结果图

4.3.3 数据精简

小组继续对模型预测结果进行分析，发现很多 group 的参数人数过少，这些 group 中的用户提供不了对于是否加入该 group 的相关信息，并严重影响了模型的泛化能力。

对于上述问题，小组决定剔除参加 group 数小于 3 的用户，进一步平衡数据样本，提高模型的泛化能力。

使用重新预处理后的数据，重新训练网络模型，得到的 PR 曲线和混淆矩阵如图11和表3所示。

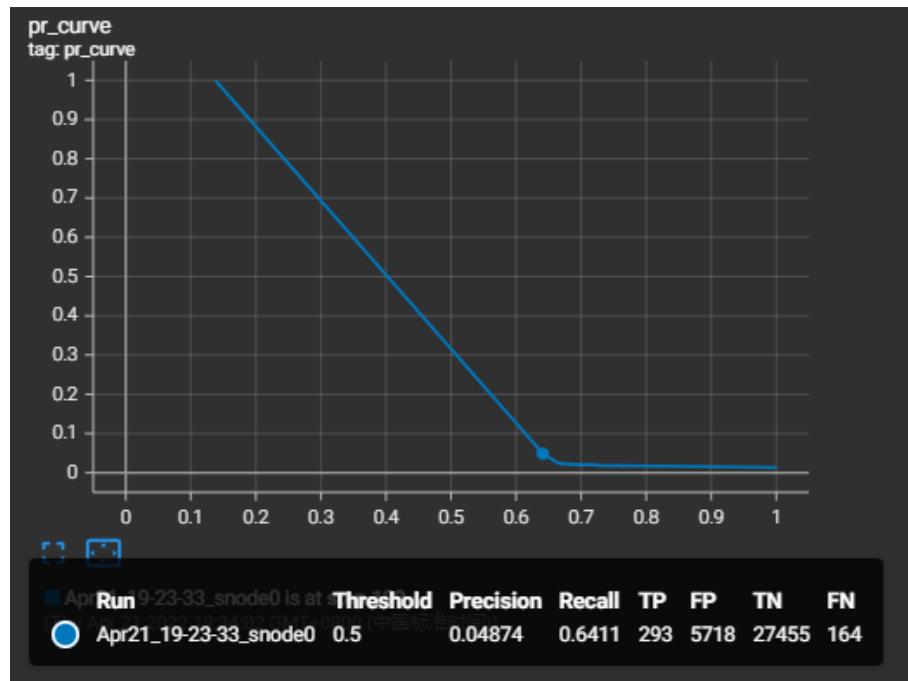


图 11: 数据精简后的训练过程 PR 曲线图

真实情况	预测结果	
	正例	反例
正例	293(TP)	164(FN)
反例	5718(FP)	27455(TN)

表 3: 数据精简后的混淆矩阵

可以计算得到准确率：

$$\text{accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) = (293 + 27455) / (293 + 27455 + 5718 + 164) = 0.83$$

训练模型在划分的测试集上的预测结果如图12所示。可以看出，模型在初始实验结果的基础上，经过地域聚合和数据精简后，虽然牺牲了一定的精度，但是换来了模型的泛化能力。同时，模型的预测的准确率为 0.79，较训练的准确率 0.83 也较为合理。

```

epoch: 197, loss: 3263.950195
epoch: 198, loss: 3369.235840
epoch: 199, loss: 3364.612305
epoch ━━━━━━━━━━━━━━━━ 100% 0:00:00
predict
cuda:0
56640
65490
test accuracy: 0.798656
positiveLabelNum: 40 negativeLabelNum: 65450 ratio: 0.000611

```

图 12: 地域聚合后在测试集上验证结果图

4.3.4 结果综合

将使用 MSE 不进行优化的初步实验结果，依次经过地域聚合和数据精简后的实验结果放在一起进行分析，综合结果如表4所示。从表格中可以看到，经过 MSE，地域聚合和数据精简

	训练集							测试集
	TP	FP	TN	FN	Precision	Recall	Accuracy	
MSE	143	7598	25485	404	0.02	0.26	0.76	0.94
地域聚合	283	6411	26483	453	0.04	0.38	0.8	0.86
数据精简	293	5718	27455	164	0.05	0.64	0.83	0.80

表 4: MSE, 地域聚合, 数据精简综合实验结果汇总表

等对数据进行预处理的操作，缓解了正样本和负样本比例不均衡的问题，模型训练过程中的 Precision, Recall 和 Accuray 不断提升。同时，测试集的 Accuracy 不断下降，而这正是我们小组预期的结果。因为开始的模型过于特化，几乎都将用户预测不加入 group 中来逃避惩罚，经过优化后，虽然模型的准确率下降，但是模型获得了更强的泛化能力，更能为一般情况（而非正负样本比例不均衡）进行预测。

4.3.5 存在问题

由于选题的因素，实验存在如下问题：

1. 样本不均衡：用户参与 group 的数量相对于用户参与事件很少，绝大部分用户都只会参加 1 个 group。
2. 样本数量少：group 数相对事件数很少，只有 800 个。
3. 预处理后数据集减少：在经过地域聚合和数据精简预处理后，不符合模型需求的数据被剔除，用于训练和测试的数据集减少。

小组成员已经在上述问题的基础上进行了思考，用于优化因选题因素导致的样本不均衡问题，实验结果也可以表明我们的优化成果，但是实验结果的效果依然存在较大的改进空间，也是小组后续努力的方向。

5 参考文献

Tong Xu, Hao Zhong, Hengshu Zhu, Hui Xiong, Enhong Chen, Guannan Liu, Exploring the Impact of Dynamic Mutual Influence on Social Event Participation, In Proceedings of 2015 SIAM International Conference on Data Mining (SDM'15), Vancouver, Canada, 2015, 262-270.

6 实验代码

github: <https://github.com/Kirrito-k423/repetitionSocialNetwork>