一、背景及现状

网络的快速发展和大数据时代的到来让基于位置的社交网络有巨大的发展空间和潜力，并且快速发展起来。基于位置的社交网络(LBSN)已经融入到我们生活的方方面面，定位系统相关功能也有其身影。主要的功能就是位置推荐、位置签到、位置评论和位置共享。在 LBSN 中，用户可以被推荐可能感兴趣的地点，这样的功能可以用于零售行业，用户行为分析等研究领域。

其中最普遍用到的一类算法为协同过滤算法，此类算法被广泛应用与轨迹聚类和位置预测中，同时也包含大量的兴趣推荐算法。然而这类算法存在着以下几个问题：

1) 推荐不准确。

LBSN 中兴趣点总数异常庞大，用户访问的兴趣点只占极少一部分，这导致用户 兴趣点签到矩阵极为稀疏，使得挖掘用户的偏好具有难度，且由于各算法在考虑社交和地理位置等因素的影响时不够深入全面，进一步造成推荐的不准确。

2) 推荐缺乏个性化、推荐效率低。

协同过滤算法的基础是相似度计算，算法缺乏个性化的表示方法，导致难以获取用户的个性化偏好。由于算法缺乏简单有效的召回机制，需要针对所有兴趣点计算推荐得分，因此造成大量的时间消耗，导致推荐效率低。

为了解决以上问题，探索用户好友及 2-hop 好友的社交影响，缓解数据的稀疏问题；本文利用核密度估计方法对用户的个人签到数据进行建模，用以获取用户个性化的地理位置偏好。

二、问题重述和解析

1、对于前 10 名用户（具有更多 checkins）构建：

--- 一篮子推荐：场地；

--- 用户将根据他们的朋友访问的可能场馆列表

--- 他们将在哪里

2、对于前 10 名“社交”用户（更多朋友）：

--- 将路径绘制用户 checkins 图

--- 列出您的朋友以及他们在“品味”方面的近距离

针对以上目的，对需要解决的问题进行解释如下，主要目标即为：

1. 对于前 10 名用户构建基于好友和地点的一系列推荐地点。
2. 对于前 10 名社交用户构建好友相似度。
3. 绘制用户一定时间内路径
4. 预测用户出现位置

前两个问题将会重点分析并讨论。

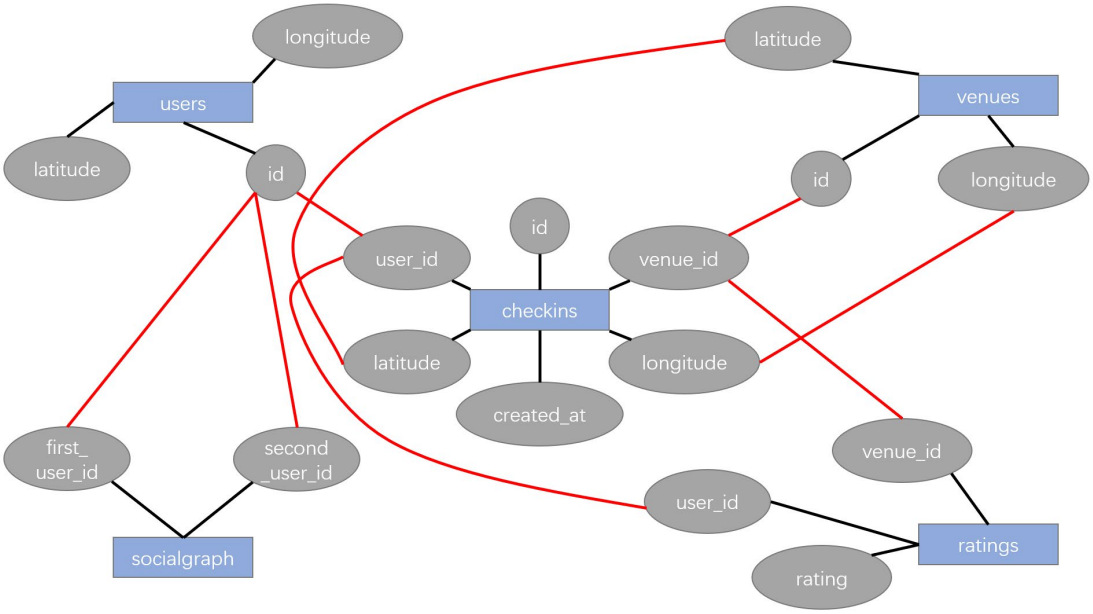
三、文献研究

对于地点特征和社交网络，文献[1]在协同过滤算法的基础上，综合考虑社交因素和地理位置因素，提出了融合地理 社交因素的个性化位置推荐(Personalized Geo-Social Location Recommendation，iGSLR) 算法; 文献[2]利用矩阵分解(Matrix Factorization，MF) 模型，融合用户的社交影响力对模型进行增强; 文献[3]将好友类型划分为社交好友、位置好友和邻近好友，系统地考虑了不同好友的社交影响; 文献[4－5]通过对社交因素、地理位置因素和兴趣点类别因素之间相关性的研究，提升推荐效果。

与音乐、电影等项目推荐不同，地理位置因素是兴趣点推荐的一个独特因素，大量的研究利用兴趣点在空间上的聚集现象来提高推荐的准确性[6－9]。主流的地理位置建模方法有以下三类: 幂律分布、高斯分布和核密度估计方法。其中:文献[6－7]认为用户签到的分布符合幂律分布，通过幂律分布获取用户签到的距离偏好; Liu 等[8]将用户签到分布抽象为高斯分布，结合泊松因子模型进行协同过滤推荐; 与以上方法不同，文献[1, 9]利用核密度估计方法从二维角度对用户签到记录进行建模，避免了对所有用户设置公共距离函数的局限性，因此可以获取个性化的地理位置偏好。因此本文也拟采用核密度的计算方式对地理位置偏好进行个性化概率估计。

四、数据预处理部分

首先，我们对 5 个数据集分别进行了处理（查看缺失值、去重等）。然后，再有针对性地对 5 个数据集中的关联内容进行处理。下图为 5 个数据集之间的联系。



4.1 读取数据

使用传统的 pandas 来读取数据集，根据记录的数量大小，设置 chunksize 参数来控制每次迭代数据的大小，以加快读取速度。

4.2 checkins 数据集预处理

① 我们发现在该数据集中相同 venue_id 对应的 latitude 和 longitude 有细微的差别，考虑可能是用户打卡的地点问题，所以统一使用 venue 表中的地理位置信息作为每个地点唯一的经纬度。

② 将 user_id、venue_id 和 created_at 三个字段均相同的记录视为重复并删除。

4.3 socialgraph 数据集预处理

① 删除表格尾部有一些杂乱的字段。

② 因为 id 都是整数，直接将 float 转换为 int，以节省内存空间。

③ 将 first_user_id 和 second_user_id 两个字段均相同的记录视为重复并删除。

4.4 ratings 数据集预处理

① 观察到，同一个用户对同一个地点有多个甚至是多个相同的评分。因为缺乏时间数据，所以无法对数据是否重复下肯定的结论。因此，在我们的模型中规定：ratings 中的每一条记录视为用户对该地点的一次评分，即使 user_id、venue_id 和 rating 均相等，也不视为重复。

4.5 users 数据集预处理

① 对 id、latitude 和 longitude 三个字段均相同的记录视为重复并删除

4.6 venue 数据集预处理

① 我们发现有部分经纬度均为 0 的地点。于是我们尝试使用 google 地图对经纬度为 0 的地点进行定位，发现是在大西洋上。于是我们认为经纬度为 0 的数据代表着该地点的经纬度数据缺失，将其从数据集中删除。

4.7 考虑是否存在“关注”行为

“关注”行为，顾名思义，就是用户 A 与用户 B 有联系，但用户 B 与用户 A 之间不存在联系。

我们使用 networkx 库搭建了用户关系图，并计算图中每一个节点的入度与出度的差值，最后按照从大到小的顺序排序，结果如下所述：

① 仅有 2 个节点入度大于出度，其差值为 1，即这两个节点属于“被关注”节点

② 仅有 1 个节点的出度大于入度，差值为 2，即该节点属于“关注”节点

③ 其他所有节点的入度均等于出度

因此，对于该数据集而言，我们认为可以在之后的建模中忽略“关注”行为的存在。

4.8 处理 ratings 数据和 checkins 数据不匹配的问题

我们使用 dask 库重新读入 ratings 和 checkins 数据，让 dask 加速我们之后聚合运算。我们首先在 ratings 数据集中聚合了用户对一个地点的评分，得到每个用户对每个地点的评分均值和评分次数。之后，我们在 checkins 表中聚合了每个用户访问每个地点的打卡次数。通过外连接合并两个聚合表，我们发现一些用户对一些地点的 checkins 次数少于 ratings 次数，甚至有一些用户只有 ratings 而没有 checkins，但不存在 checkins 次数大于 ratings 次数的情况。

因为无法验证是否存在 ratings 重复的情况，并且出现了部分用户对于部分地点只有 ratings 但没有 checkins 的情况。因此，在接下来的模型中，我们认为 ratings 中的数据是完整的，并以 ratings 的次数作为用户对某一个地点的访问次数。原本数据集中同一个人访问的不同的地点数最多只有 50 多个，通过 ratings 的“修正”，数据集中最多的一个人访问了 1000 多个不同的地点。

4.9、使用 reduce_mem_usage 函数通过调整数据类型，帮助我们减少数据在内存中占用的空间

通过判断每列数据中的最大值和最小值，来将“int64”修改为“int32”、“int16”或“int8”，将“float64”修改为“float32”或“float16”，以减少数据在内存中的占用空间。

六、基于用户的协同过滤

根据社会同质理论和影响理论，具有共同的兴趣爱好和相似行为的用户社交形象更为相似。在基于用户的推荐中，其更倾向于接受口味与自己相似的用户的推荐。基于用户的协同过滤算法就是参考了这一思路。该算法改进基于好友的协同过滤算法，不同于基于好友的协同过滤模式，算法分别计算用户与其二度相关用户的打卡经历相似度和好友相似度，并结合用户历史签到记录和评分，计算用户访问兴趣点的概率。

符号解释

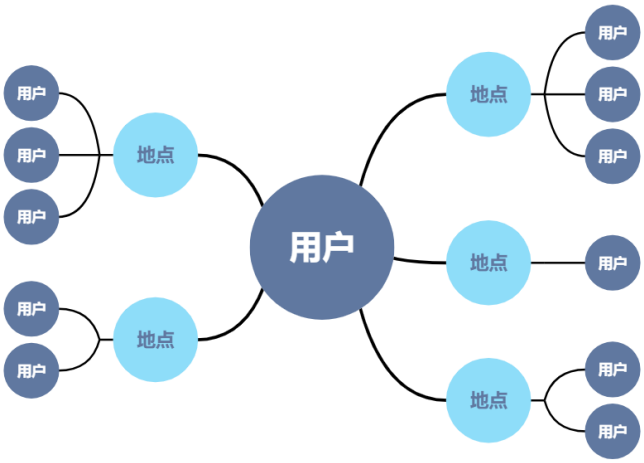
符号	描述
u_i	第 i 号用户
F_i	用户 u_i 的好友集合
L_i	用户 u_i 的打卡地点集合
$SI_{i,j}$	用户 u_i 和 u_j间的相似度$SI_{i,j}$
$p_{j,k}$	用户u_j打卡地点l_k的频率
$r_{j,k}$	用户u_j对地点l_k的评分
$c_{i,k}$	用户u_i对地点l_k的推荐评级

定义 1 二度相关用户：

首先，由一个用户的 check in 经历，可以得到该用户打卡的地点的集合；

而在这个地点集合中，历史曾经访问过其中任意一个地点的用户，可定义为初始用户的二度相关用户。

由图片直观表示即为：



图片中环绕四周的小圆圈所代表的用户即为中心用户的二度相关用户。

公式 1：用户u_i,u_j间的相似度$SI_{i,j}$：

用户相似度 $SI_{i,j}$，是基于用户 u_i 和 u_j 的共同好友比例、共同打卡地点比例进行计算的，共同好友与共同打卡点的比例越大，好友相似度越高。

式子中 λ 决定了共同好友比例、共同打卡地点比例的权重。当 $\lambda=1$ 时，用户相似度完全由共同好友比例决定；当 $\lambda=0$ 时，用户相似度完全由共同打卡地点比例决定。

$$SI_{i,j} = \lambda \frac{F_i \cap F_j}{F_i \cup F_j} + (1-\lambda)\frac{L_i \cap L_j}{L_i \cup L_j}$$

公式中：

F_i是用户u_i的朋友集，F_j是用户 u_j 的朋友集；

L_i是用户u_i的打卡地点集合，L_j是用户u_j的打卡地点集合；

$SI_{i,j}$反映用户u_i,u_j之间的相似度，相似度最低为 0，最高为 1。

公式 2：用户u_i对地点l_k的推荐评级$c_{i,k}$：

$$c_{i,k} = \frac{\sum_{u_j \in F_i} SI_{i,j} \times \sqrt[3]{p_{j,k}} \times r_{j,k}}{\sum_{u_j \in F_i} SI_{i,j}}$$

u_j是用户u_i的二度相关用户；$r_{j,k}$是用户u_j对地点l_k的评分（已经经过归一化处理），$p_{j,k}$是用户u_j打卡地点l_k的频率。为了将频率与评分同时英勇进推荐系统中，令 r 定义为：$\sqrt[3]{p_{j,k}} \times r_{j,k}$ 即 $\sqrt[3]{频率} \times 评分$

公式 3：修正后的推荐评级$c_{i,k}$：

$$c_{i,k} = \frac{1 + \sum_{u_j \in F_i} SI_{i,j} \times \sqrt[3]{p_{j,k}} \times r_{j,k}}{1 + \sum_{u_j \in F_i} SI_{i,j}}$$

为防止分母为 0 导致$c_{i,k}$取值无意义，在分子分母同时加 1 予以修正

最终修正$c_{i,k}$可反应用户u_i对地点l_k的推荐指数，最低为 1。推荐评级$c_{i,k}$同样要进行归一化处理，以便于联合余下两个算法得出系统的总推荐评分。

七、基于随机游走的好友相似度计算

基于文献[10-12]，我们决定尝试使用随机游走类型的算法计算好友相似度，因为在文献里，作者经过比较后得出使用该方法获得的好友相似度比传统的基于好友的协同过滤的方法更稳定也更可靠。

在下文中，首先介绍了马尔可夫链的相关原理，并以此引出使用的网络模型，之后介绍我们对复现的算法的具体实现、优化，最后将介绍我们对算法的理解与尝试的改进。

7.1 马尔可夫性质、马尔可夫过程和马尔可夫链

马尔可夫性质：对于任何给定的时间，当前时间点往后的未来状态的条件分布仅取决于当前状态，而完全不取决于过去状态（无记忆性）

马尔可夫过程：具有马尔可夫性质的随机过程称为马尔可夫过程，其公式可以表示如下。

$$P(future|present,past)=P(future|present)$$

马尔可夫链：马尔可夫链是具有离散时间和离散状态空间的马尔可夫过程，是遵循马尔可夫性质的一个离散状态序列。

在数学上，可以表示一个马尔可夫链为

$$X=(X_n)_{n\in\mathbb{N}}=X_0,X_1,X_2,...$$
$$其中,X_n\in E \quad \forall n\in\mathbb{N}$$

那么，马尔可夫性质意味着有

$$\mathbb{P}(X_{n+1}=s_{n+1}|X_n=s_n,X_{n-1}=s_{n-1},X_{n-2}=s_{n-2},...)=\mathbb{P}(X_{n+1}=s_{n+1}|X_n=s_n)$$

7.2 社交网络：一个马尔可夫链

为什么说社交网络是一个马尔可夫链呢？社交网络中，每个人都被视为一个节点，朋友、亲属、同事等多个关系将两个节点连接在一起。假设一个幸运的福袋从八竿子打不着的人（你和他没有直接的关系）经过层层朋友、同学等关系连接传递到了你这里，然后你只能将它继续传递给你的朋友、亲属或同事等和你有

直接关系的人。无论这个福袋从哪来，到你这，你就只能传递给特定的人（你的朋友、亲属等），符合马尔可夫过程，即未来的状态（未来谁将持有福袋）只与现在的状态（你持有福袋有关），并且是具有离散状态空间（不会传给 0.1 个人或 0.5 个人，只能传给 1 个人）。

因此，社交网络中的活动可以看作一个马尔可夫链。从一个节点到另一个节点的转移概率取决于两个节点之间的亲密程度。越亲密，转移的概率自然也就越大。

定义社交网络的邻接矩阵U如下，如果i和j两人有直接联系，则$u_{ij}=u_{ji}=1$，反之$u_{ij}=u_{ji}=0$.

$$U=\begin{pmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{n1} & \cdots & u_{nn} \end{pmatrix}$$

在不考虑用户亲密度差异的情况下（即认为用户与所有有直接关系的人的亲密度均一致），从用户u_a转移到用户u_b的概率可以表示为

$$p(b|a)=\frac{u_{ab}}{\sum_{i=1}^{n} u_{ai}}$$

因此，可以得到用户u_a的转移矩阵为

$$p=[p(u_1|u_a),p(u_2|u_a),...]$$

7.3 拉普拉斯矩阵L和其伪逆矩阵（也称摩尔-彭若思广义逆）$L+$

以社交网络邻接矩阵为例，该矩阵的拉普拉斯矩阵的定义如下

$$L(G)=D(G)-A(G)$$

$$L(G)=\begin{pmatrix} degree(u_1) & -u_{12} & \cdots & -u_{1n} \\ -u_{21} & degree(u_2) & \cdots & -u_{2n} \\ \vdots & \ddots & \vdots & \vdots \\ -u_{n1} & -u_{n2} & \cdots & degree(u_n) \end{pmatrix}$$

根据文献[3]，$L(G)$的伪逆矩阵$L+$被定义为

$$L+=(L-ee^T/n)^{-1}+ee^T/n$$

因为大型稀疏矩阵的逆是极难求解的，并且只需要求解出具有最多 checkins 数据的前 10 位用户与好友的相似度以及具有最多朋友的前 10 为用户与好友之间的相似度。因此，根据文献[3]提供的迭代方法每次求取$L+$的一行 l_i^+ 以减少运算量

以下为计算l_i^+的方法

1.定义 $I=\begin{pmatrix}1&0&\cdots&0\\0&1&\cdots&0\\\vdots&\ddots&\vdots&\vdots\\0&0&\cdots&1\end{pmatrix}$, $e_i=I$的第i列向量, $e=[1\ \ 1\ \ \cdots\ \ 1]^T$

2.计算向量e_i在L的列空间上的投影

$$y_i=proj_L(e_i)=(I-ee^T/n)e_i$$

3.找到方程$Ll=y_i$的解l_i^{*+}

4.将结果l_i^{*+}投影到L的行空间上

$$l_i^+=proj_L(l_i^{*+})=(I-ee^T/n)l_i^{*+}$$

在运算中，ee^T/n会产生一个全部元素均为$1/n$的$n\times n$稠密矩阵。当 n 比较大时（在我们的计算中n的单位均为万级以上，最高为十万级别），会导致内存溢出，无法正常进行运算。因此有必要对其进行优化以更节约内存、更快速地进行计算。

7.3.1 对第二步（计算向量e_i在L的列空间上的投影）进行再化简

$$(I-ee^T/n)e_i=\begin{pmatrix}\frac{n-1}{n}&-\frac{1}{n}&\cdots&-\frac{1}{n}\\-\frac{1}{n}&\frac{n-1}{n}&\cdots&-\frac{1}{n}\\\vdots&\ddots&\vdots&\\-\frac{1}{n}&-\frac{1}{n}&\cdots&\frac{n-1}{n}\end{pmatrix}_{n\times n}\cdot e_i=[-\frac{1}{n},\cdots,-\frac{1}{n},\underset{i}{\frac{n-1}{n}},-\frac{1}{n},\cdots,-\frac{1}{n}]^T=I\text{的第}i\text{列每个元素}-\frac{1}{n}$$

通过手工计算，可以将第二步化简为I的第i列向量所有元素减去$1/n$，从而避免了生成一个巨大的稠密矩阵。

7.3.2 根据文献[11]对第三步（找到方程$Ll=y_i$的解l_i^{*+}）进行再化简

使用普通的求解稠密矩阵方程的方法求解大规模的稀疏矩阵方程效率很低，因此，结合拉普拉斯矩阵半正定的性质性，使用*cvxopt*库提供的*cvxopt.cholmod.linsolve*方法对L进行科列斯基分解并求解方程。

7.3.3 对第四步（将结果l_i^{*+}投影到L的行空间上）进行再化简

$$(I-ee^T/n)l_i^{*+}=\begin{pmatrix}\frac{n-1}{n} & -\frac{1}{n} & \cdots & -\frac{1}{n}\\ -\frac{1}{n} & \frac{n-1}{n} & \cdots & -\frac{1}{n}\\ \vdots & \ddots & \vdots & \\ -\frac{1}{n} & -\frac{1}{n} & \cdots & \frac{n-1}{n}\end{pmatrix}_{n\times n}\cdot l_i^{*+}=l_{i1}^{*+}\begin{pmatrix}\frac{n-1}{n}\\ -\frac{1}{n}\\ \vdots\\ -\frac{1}{n}\end{pmatrix}+l_{i2}^{*+}\begin{pmatrix}-\frac{1}{n}\\ \frac{n-1}{n}\\ \vdots\\ -\frac{1}{n}\end{pmatrix}+\cdots+l_{in}^{*+}\begin{pmatrix}-\frac{1}{n}\\ \vdots\\ -\frac{1}{n}\\ \frac{n-1}{n}\end{pmatrix}$$

$$=\begin{pmatrix}l_{i1}^{*+}\cdot\frac{n-1}{n}-(\sum_{j=1}^{n}l_{ij}^{*+}-l_{i1}^{*+})\cdot\frac{1}{n}\\ l_{i2}^{*+}\cdot\frac{n-1}{n}-(\sum_{j=1}^{n}l_{ij}^{*+}-l_{i2}^{*+})\cdot\frac{1}{n}\\ \vdots\\ l_{in}^{*+}\cdot\frac{n-1}{n}-(\sum_{j=1}^{n}l_{ij}^{*+}-l_{in}^{*+})\cdot\frac{1}{n}\end{pmatrix}=\begin{pmatrix}l_{i1}^{*+}-\frac{1}{n}\sum_{j=1}^{n}l_{ij}^{*+}\\ l_{i2}^{*+}-\frac{1}{n}\sum_{j=1}^{n}l_{ij}^{*+}\\ \vdots\\ l_{in}^{*+}-\frac{1}{n}\sum_{j=1}^{n}l_{ij}^{*+}\end{pmatrix}=\begin{pmatrix}l_{i1}^{*+}-avg(l_i^{*+})\\ l_{i2}^{*+}-avg(l_i^{*+})\\ \vdots\\ l_{in}^{*+}-avg(l_i^{*+})\end{pmatrix}$$

与第二步化简类似，通过人工计算，可以将其化简为l_i^{*+}中的每个元素减去l_i^{*+}的均值，同样避免了一个巨大稠密矩阵的生成。

通过人工手动先手动化简再编写程序执行的方式，可以大大提高程序的执行效率、减少内存的占用、加速计算过程。

7.4 $L+$的意义是什么？

原文[11]是这样描述的：$L+$ provides a similarity measure ($sim(i,j)=l_{ij}^{+}$) since it is the matrix containing the inner products of the node vectors in the Euclidean space where the nodes are exactly separated by the ECTD. (ECTD means Euclidean Commute Time Distance)

同时，$L+$还可以用来计算两个节点之间的平均通勤时间（Average Commute Time）。平均通勤时间定义为：从一个节点出发经过另一个节点并重新返回起始

节点所消耗的平均时间，根据l_i^+计算评价通勤时间的公式定义如下：

$$ACT(i,j)=V_G(l_{ii}^+ + l_{jj}^+ - 2l_{ij}^+)$$

其中，V_G为所有节点的度的总和

可以发现，当其他条件不变的时候，随着l_{ij}^+的增加，平均通勤时间$ACT(i,j)$也就越小。平均通勤时间越小，在一定程度上说明两个节点之间的短路径也就越多，说明这对节点之间的联系也就越紧密。

7.5 无权拉普拉斯矩阵和加权拉普拉斯矩阵

根据上文的阐述，两个节点之间联系的强弱会影响转移矩阵的概率取值，自然也会影响两个节点的平均通勤时间，进而对也会对$L+$有影响。那我们应该如何去考虑两个节点之间联系的强弱呢？

7.5.1 无权拉普拉斯矩阵

首先，根据文献[2-3]，我们选择较为普遍的无权拉普拉斯矩阵来计算$L+$（即在邻接矩阵U中，如果两个节点之间有连接，则$u_{ij}=u_{ji}=1$，反之$u_{ij}=u_{ji}=0$）并且将地点作为节点加入到图中，两个节点是否相连由是否满足某一强度条件来判断。下表为该方法下的邻接矩阵中元素取值条件说明。

用户-用户边的权重	用户-地点边的权重
是朋友，则$=1$	访问次数 > 均值 & 评分均值 > 均值，则$=1$
不是朋友，则$=0$	反之，则$=0$

7.5.2 加权拉普拉斯矩阵

考虑到之前的方法无法很好地把信息应用到模型中（只有 0 和 1），我们在之后的模型中尝试引入加权的拉普拉斯矩阵。但遗憾的是，在计算$L+$的第三步（找到方程$Ll=y_i$的解l_i^{*+}）过程中，在前 10 个 checkins 数量最多的用户中仅有第一个用户的数据能够成功找到解，其他的用户数据均无法观察到解。虽然失败了，但鉴于我目前对该算法真正的原理的掌握还不是很透彻，还是有继续往下

研究、实践的潜力。下表为该方法下的邻接矩阵中元素取值条件说明。

用户-用户边的权重	用户-地点边的权重
是朋友，则取值为基于无权的邻接矩阵的 Jaccard 相似度	权重为 $评分均值 \times \sqrt[3]{访问次数}$
不是朋友，则＝0	

注：1、在计算用户-用户边的权重时，因为用户的好友过多，在算力允许的条件下，只计算了 checkins 数量最多的前 10 个各自与其所有朋友之间的 Jaccard 相似度，他们的朋友与他们的朋友的朋友之间的相似度由各自计算得到的他们与朋友的相似度的中位数替代。

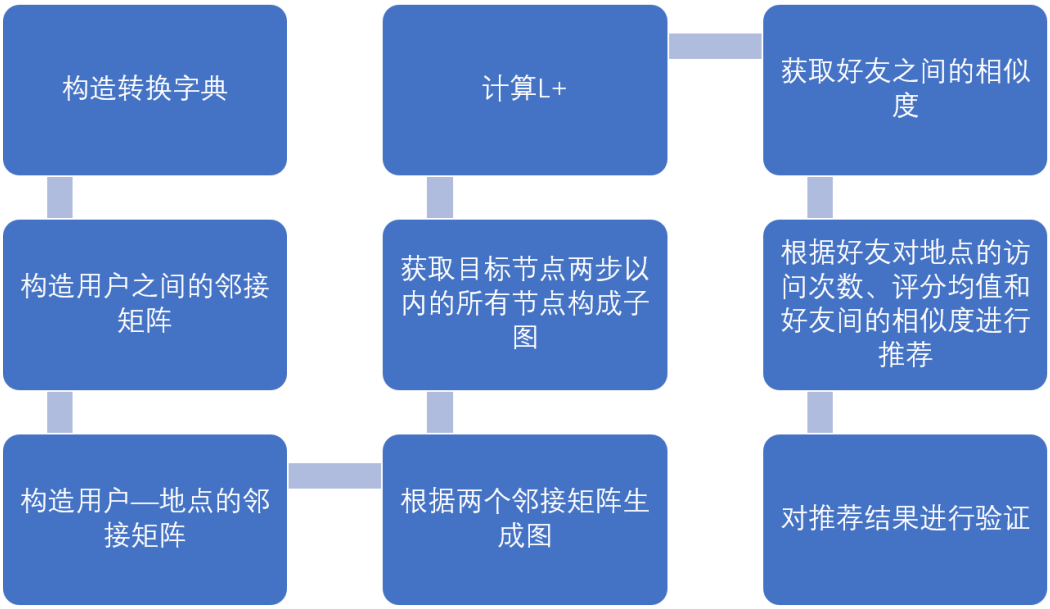
2、评分均值先进行了归一化后才用于权重计算，用户-用户边的权重和用户-地点边的权重在计算完成后均进行了归一化处理。

7.5.3 改进的无权拉普拉斯矩阵（暂未实现）

在加权拉普拉斯矩阵运算出错的情况下，我又想到了可以尝试先对无权的拉普拉斯矩阵进行改良。改良针对用户-用户边的权重：因为 4.5.1 中对所有好友不加鉴别地赋予了 1，只对用户-地点进行了筛选，而未对好友进行筛选。参照加权拉普拉斯矩阵中用户-用户边的权重计算公式，我们计划对 Jaccard 相似度设置一个阈值（可以是均值、中位数等），大于阈值的两个用户之间的权重为 1，反之则为 0。

7.6 代码流程图

由于 Python 代码过多，并且直接放到文档中进行展示的效果也不佳。因此，下面将以流程图的形式对整个计算 $L+$ 的过程进行简要介绍和阐述。如有需要，可以直接阅读 jupyter notebook 代码源文件。



关于流程图的说明：

1、构造转换字典：建立 user_id、venue_id 与图中节点号的一一对应关系

2、根据两个邻接矩阵生成图：两个邻接矩阵生成一张图

3、获取目标节点两步以内的所有结点构成子图：因为数据集过于庞大，并且除了有较近联系的节点外，其他节点的相似度计算是几乎没有意义的。因此，考虑到算力问题，我们只选择了目标节点（checkins 数量最多的 10 个和朋友数量最多的 10 个）两步以内的节点构成的子图计算 $L+$

4、获取好友之间的相似度：节点 i 和 j 的相似度由 l_{ij}^{+} 给出

5、根据好友对地点的访问次数、评分均值和好友间的相似度进行推荐：将地点 m 推荐给用户 i 的概率由以下公式给出

$$p_{i.m}=\frac{\sum_{j=1}^{Friends} sim(u_i,u_j)\cdot rate_{j,m}\cdot\sqrt[3]{freq_{j,m}}}{\sum_{j=1}^{Friends} sim(u_i,u_j)}$$

$$\text{其中，}\ sim(u_i,u_j)=l_{ij}^{+}$$

6、对推荐结果进行验证：首先，随机遮蔽用户 30%的地点的 checkins（暂不考虑时间先后问题），使用用户剩下的地点和好友关系对其进行推荐，并观察推荐的准确率。

7.7、推荐结果说明

目前仅对基于无权拉普拉斯矩阵计算得到的$L+$进行了验证，以传统的基于好友的协同过滤算法作为基准进行比较，但结果并不是很理想。在前 1%、5%和10%的推荐中，两个算法均几乎没有推荐“成功”的情况出现。总体而言，在该数据集中，$L+$和基于好友的协同过滤算法打成平手，并没有哪一个表现特别突出或特别逊色。我们认为存在以下几个因素导致推荐的准确率较低：

1、数据涵盖的信息较少：一个人只有他的 checkins 信息、评分和好友信息，难以拿到个体属性的准确信息，故而较难进行精确的推荐。

2、推荐算法还有待改进：目前使用的推荐算法运算感觉还属于比较简单的层次，存在较大改进和提升的空间

八、核密度估计

一方面，同一用户访问的每对位置之间的距离会对用户签入行为有地理影响；另一方面，核密度估计可以用于任意分布，而不需要假定距离分布的形式是已知的。因此，为提高推荐的个性化程度，我们利用个人签到数据，进行核密度估计，建立个性化地理位置模型。模型的建立分为三个步骤：距离样本数据的获取、距离分布的估计和个性化地理位置影响力的计算。

首先，对于用户u_i，其签到地点集合为L_i={l_1, l_2，…，l_k}，可以计算得到L_i中每一对地点之间的距离$d'_{m,n}$，从而获取距离样本数据的集合 D。同样地，对于一个新的签到地点l_j，根据式（1）计算每一对地点的距离$d_{j,k}$。

$$d_{j,k} = distance(l_j, l_k);\ \forall l_k \in L_i \qquad (1)$$

根据式（1）计算得到每一对兴趣点距离为 $d_{j,k}$ 之后，由式（2）计算核密度估计$\hat{f}(d_{j,k})$：

$$\hat{f}(d) = \frac{1}{|D|h}\sum_{d' \in D} K\left(\frac{d-d'}{h}\right) \qquad (2)$$

其中：K(•)为核函数，h 是平滑参数，也被称为带宽或窗口。核函数选择最常用的高斯核函数$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$。当选择高斯核核函数时，通过最小化分布的平均积分误差，平滑参数的最佳选择为$h = (\frac{4\hat{\sigma}^5}{3n})^{1/5} \approx 1.06\hat{\sigma}n^{-1/5}$，$\hat{\sigma}$为 D 的标准差。

最后，用户u_i访问一个新兴趣点的概率为由每一对$\hat{f}(d_{j,k})$共同决定：

$$p_g(i,j) = \frac{1}{n}\sum_{1}^{n} \hat{f}(d_{j,k})$$

代码部分

（1）获取目标用户基本数据

```
import pandas as pd
import numpy as np
#读入目标用户的数据
USERS_DATA = pd.read_csv("/Users/xiesihua/Desktop/targeted Data/targetUsers.csv")
VENUES_DATA = pd.read_csv("/Users/xiesihua/Desktop/targeted Data/targetVenues.csv")
RATINGS_DATA = pd.read_csv("/Users/xiesihua/Desktop/targeted Data/targetRatings.csv")
#定义变量
users_max = USERS_DATA.iloc[-1].values[0]
users = [] #ratings_data 中，users_id
venues = [] #ratings_data 中 venues_id
raw_counts = 0
std_sigma = [] #每个用户 D 的标准差
abs_z_sigma = [] #｜D｜
h_sigma = [] #平滑参数
d_ = [] #每个用户都有一组 d'，全部 d'的集合
venues____ = []  # 所有的用户去过的地点
for index, row in RATINGS_DATA.iterrows():
    user_id, venue_id = row['user_id'], row['venue_id']
    users.append(user_id)
    venues.append(venue_id)
```

（2）获取用户签到地点的距离样本数据集合，计算其标准差及｜D｜，得到平滑参数

```
#从第一个人到最后一个人
for i in range(1, 1 + int(users_max)):
```

```
        coordinate = [] #经纬度，二维坐标
        venues__ = []   # 用户经常去的地点
        count = users.count(i) #每个人出现的次数，包括重复地点
        if count == 0:
            continue
        for j in range(raw_counts, raw_counts + count):
            venue_id = RATINGS_DATA.loc[j].values[1]
            v = VENUES_DATA[VENUES_DATA['id'] == venue_id]
            venues__.append(venue_id)
            coordinate.append([v['latitude'].values[0], v['longitude'].values[0]])
        ture_coordinate = [coordinate[0]]
        venues_ = [venues__[0]]   # 该用户签到的地点
        raw_counts += count
        for j in range(1, len(coordinate)):#抓取地点坐标
            if coordinate[j][0] == coordinate[j - 1][0] and coordinate[j][1] ==
coordinate[j - 1][1]:
                continue
            else:
                ture_coordinate.append(coordinate[j])
                venues_.append(venues__[j])
        venues____.append(venues_)
        del coordinate, venues__
        z = np.zeros((len(ture_coordinate), len(ture_coordinate))) #先做一个全 0 矩阵
        for j in range(len(ture_coordinate)):#
            for k in range(len(ture_coordinate)):
                if j == k: #相同地方
                    continue
                else:
                    z[j, k] = z[k, j] = ((ture_coordinate[j][0] - ture_coordinate[k][0]) **
2 + (
```

```
                                    ture_coordinate[j][1] - ture_coordinate[j][1]) ** 2 ) **
0.5
        zz = []
        for j in range(z.shape[0]):
            for k in range(z.shape[1]):
                if z[j,k] != 0:
                    zz.append(z[j,k])
        std = np.std(zz)
        std_sigma.append(std)#标准差的集合
        abs_z = np.linalg.norm(zz, ord=2)
        abs_z_sigma.append(abs_z)
        h = (4 / 3 * std ** 5 / float(len(ture_coordinate))) ** 0.2 #平滑参数
        h_sigma.append(h) #平滑参数的集合
        dd = [] #一个人 d'的集合
        for j in range(int(z.shape[0])):
            for k in range(j + 1, z.shape[1]):
                dd.append(z[j, k])
        d_.append(dd)
```

（3）定义核函数

```
def he(x):
    return 1 / (2 * np.pi) ** 0.5 * np.exp(-1 * x ** 2 / 2)
```

（4）定义$\hat{f}(d)$

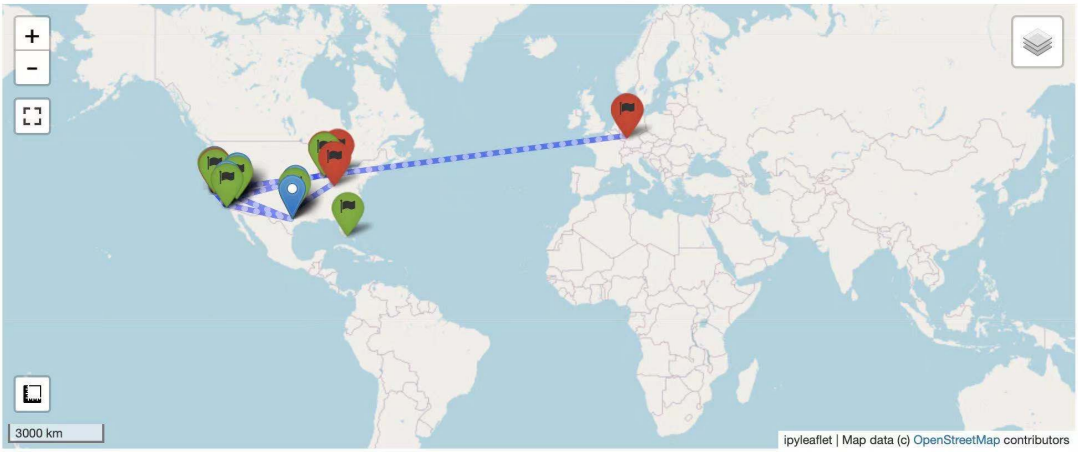
```
def f(d, abs_z_sigma, h_sigma, d_, user):
    const = 1 / (abs_z_sigma[user - 1] * h_sigma[user - 1])
    sig = 0
    for m in range(len(d_[user - 1])):
        sig += const * he((d - d_[user - 1][m]) / h_sigma[user - 1])
    return sig
```

（5）传入新地点经纬度，用户 id，得到该用户访问新兴趣点的概率

```
def getans(latii, longii, venues____, VENUES_DATA, user, m1, m2, m3):  # m1，m2，m3 分别是丨D丨集合，平滑参数集合，d'm,n集合
    venue = venues____[user-1]
    sig_sigma = []
    for u in range(len(latii)):
        dist = []
        sig = 0
        lati = latii[u]
        longi = longii[u]
        for q in venue:
            vv = VENUES_DATA[VENUES_DATA['id'] == q]
            lat = vv['latitude'].values[0]
            lon = vv['longitude'].values[0]
            distance = ((lat - lati) ** 2 + (longi - lon) ** 2) ** 0.5
            dist.append(distance)
        for m in dist:
            res_ans = f(m, m1, m2, m3, user)
            sig += res_ans
            sig_sigma.append(sig / len(dist))
    return sig_sigma
```

九、绘图

由于具有最多朋友的前 10 名用户都没有任何一条 checkins，于是我们便把限制放宽了。但在具有最多朋友的前 100 名用户中，也才仅有 11 个用户具有为数不多的 checkins 记录。于是，我们便将这 11 个人的 checkins 使用 ipyleaft 库绘制成图，如下图所示。



图例说明：

1、红色的代表起点，也就是最远的时间点的 checkins；绿色的代表终点，也就是最近的时间点的 checkins；蓝色代表中间节点

2、鼠标点击节点可以看到节点的 id、被访问的时间等信息

3、蓝白色的线的流动的方向表示移动的先后顺序（checkins 的先后顺序）