# 文献笔记

8.19–8.25

**1.GETNext: Trajectory Flow Map Enhanced Transformer for Next POI Recommendation**

(1) 研究内容：将全局转换模式、用户的一般偏好、时空上下文和时间感知类别嵌入结合到一个 transformer 模型中以更好的预测 next-POI

(2) 文章整体要点：

- next-POI 比传统的 POI 推荐更注重近期轨迹的时间影响，以预测用户的下一步行动
- 提出了一种与用户无关的全局轨迹流图和一种新颖的图形增强型 transformer 模型 (GETNext)，以更好地利用广泛的协作信号 (来自其他用户) 来进行更准确的下一个 POI 预测，同时缓解冷启动问题。
- GETNext：将全局转换模式、用户的一般偏好、时空上下文和时间感知类别嵌入结合到一个 transformer 模型中，以预测用户的未来移动。
- 原有的采用各种形式的递归神经网络 (RNN) 来编码时空信息的方法的局限：
  - 由于短轨迹提供的有限时空背景, 短轨迹上的模型性能与长轨迹相比显着下降
  - 冷启动问题：只签入少量 POI 的非活跃用户的推荐准确率通常较低。在现实生活中的推荐系统中存在冷启动问题较为常见
  - 现有模型无法弥合时间和 POI 类别。而 POI 类别通常呈现出很强的时间相关性 (与午夜相比，高峰时段火车站的签到频率显着高于午夜。)。
- 解决问题的思路：通过利用其他用户的集体信息，可以在一定程度上缓解由于固有的数据稀缺问题带来的限制。特别是，个人倾向于共享某些签到序列的片段，形成集体轨迹流。
- next-POI 推荐中利用这些轨迹流的问题及解决方案：
  - 如何聚合来自签到序列的信息以形成全局轨迹流模式的统一表示?
    - 构建了一个新颖的 (与用户无关的) 轨迹流图, 它在单个图形结构中总结了 POI 之间的轨迹以及每个 POI 的特征。
    - 图中的节点是具有包括地理位置、类别和签到次数在内的属性的 POI。如果在同一签入轨迹中连续访问它们，则有向边将 POI 连接到另一个 POI，边权重表示它们的共访问频率。

    - 该轨迹流程图捕获了 POI 之间的过渡影响。通过使用图卷积网络（GCN）将 POI 嵌入到潜在空间 (该潜在空间保留了 POI 之间的全局转换)，从而利用轨迹流图中的集体信息
    - 即 GCN 通过聚合其邻居的嵌入来更新每个节点的嵌入。从而使每个 POI 在轨迹流图中的嵌入都会受到其先例的影响，从而保留全局转换的信息。
    - 全局轨迹流图背后的一个基本假设是，推荐的 next-POI 可能不是最佳选择，但肯定会从随机猜测中得到改进。
  - 除轨迹流，如何保留类别信息、用户偏好等重要的时空上下文信息?
    - 利用嵌入层来捕捉用户的长期偏好，POI 类别嵌入和一个 time2vec 模型来描述时间嵌入 (即由用户最近的签到反映的短期偏好)。
    - 由于 POI 类别通常与时间有很强的相关性，可将类别和时间嵌入融合到一个融合模块中，生成具有时间感知的类别上下文嵌入。
  - 如何在 next-POI 推荐中利用以上所有信息，在通用运动模式和个性化需求之间取得平衡?
    - 提出了一个图形增强 transformer 框架 GETNext，该框架统一了通用的运动模式、用户的一般偏好、短期的时空内容变化以及时间感知的类别嵌入来预测用户的下一个 POI。
    - 在 GETNext 中，采用带有多个多层感知器 (MLP) 解码器的转换器编码器来集成在 POI 嵌入和其他个性化嵌入中编码的隐式全局流模式
- 主要贡献:
  - 提出了一个全局轨迹流图来模拟常见的 POI 访问顺序转换信息，并利用图卷积网络 (GCN) 进行 POI 嵌入
  - 提出了一个新的时间感知的类别上下文嵌入，以捕捉 POI 类别的不同时间模式
  - 提出了一个基于变换器的框架，将全局过渡模式、用户的一般口味 (长期偏好)、用户的短期轨迹和时空背景整合在一起，用于推荐 next-POI

(3) 问题基本描述:

- 给定大小为 M 的用户集合 U=$u_1, u_2, ..., u_M$ 和大小为 N 的 POI 集合 P=$p_1, p_2, ..., p_N$，其中 p=<latitude,longitude,category,frequency> 分别表示经度，纬度，类别以及访问频次
- 每次签到可以表示为 q=<u,p,t>$\in U \times P \times T$
- 对一个用户 u，其行动轨迹为 $S_u = (q_1, q_2, ..., q_m)$

(4) GETNext 模型架构: 图 1

(5) 轨迹图相关:

- 轨迹图的输出主要用于:
  - 使用图卷积网络 GCN 进行 POI Embedding

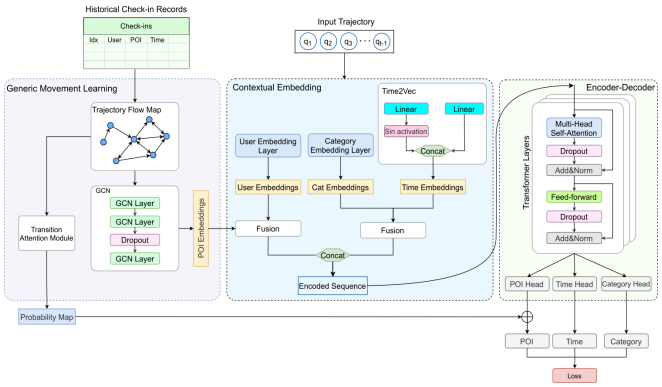



图 1:

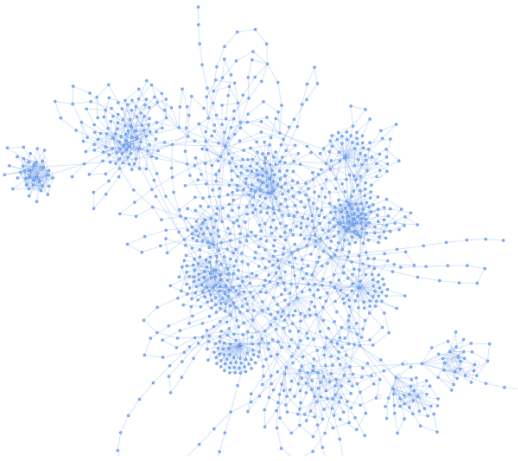


**Figure 4: Partial trajectory flow map of NYC (directions and edge weights are removed for better visualization)**

图 2:

  - 使用注意力模块 Attention 生成一个转移概率矩阵 (transsition attention map)
- Trajectory Flow Map: 图 2
- 有关该图的定义:
  - Trajectory Flow Map $\varrho = (V, E, l, w)$ 为有向带权图
  - nodes 集合 V=P，P 为 POI 集合
  - p=<latitude,longitude,category,frequency>$\in P$ 分别表示经度，纬度，类别以及访问频次
  - 若连续访问 $p_1, p_2$, 则添加边 $(p_1, p_2)$
  - 边 $(p_1, p_2)$ 上的权重为这条边出现的频次
- 可以看出:
  - 两个 POI 彼此距离越远，它们被连续签入的可能性就越小

  - 有几个密集区域

(6) POI Embeding:

- 使用图卷积网络 GCN
- 计算拉普拉斯矩阵并给出隐藏层更新方程:
  - $\widetilde{L} = (D + I_N)^{-1}(A + I_N)$
  - $H^{(l)} = \sigma(\widetilde{L}H^{(l-1)}W^{(l)} + b^{(l)})$
- 每次迭代中，GCN 层通过聚合节点的邻居信息和节点自己的嵌入来更新节点的嵌入
- 经过 $l^*$ 层的循环后，模块的输出可以表示为:
  - $e_p = \widetilde{L}H^{(l^*)}W^{(l^*+1)} + b^{(l^*+1)} \in R^{N\times\Omega}$
- 经过 GCN 可得到 POI 的向量表示

(7) Transition Attention Map: 从图 $\varrho$ 中学习到的 POI Embedding 只捕获了一般的行为模型，为进一步放大集体信号的影响，论文提出了转移概率矩阵 $\Phi$ 来明确从一个 POI 到另一个 POI 的转移概率

- $\Phi_1 = (X \times W_1) \times \alpha_1 \in R^{N\times 1}$
- $\Phi_2 = (X \times W_2) \times \alpha_2 \in R^{N\times 1}$
- $\Phi = (\Phi_1 \times 1^T + 1 \times \Phi_2^T) \bigodot (\widetilde{L} + J_N) \in R^{N\times N}$

(8) Contextual Embedding Module: 上下文嵌入模块，用于融合上下文信息，包括用户嵌入、POI 嵌入、POI 类别嵌入和时间编码

(9) POI-User Embeddings Fusion: 文中将 User embedding 和 POI embedding 进行连接以表示 check-in 活动

- $e_u = f_{embed}(u) \in R^{\Omega}$
- $e_{p,u} = \sigma(w_{p,u}[e_p, e_u] + b_{p,u}) \in R^{\Omega\times 2}$

(10) Time-Category Embeddings Fusion: 文中采用 Time2Vec, 其中将一天分为 48 个时段，每个时段 30 分钟，长度为看 k+1

- $$e_t[i] = \begin{cases} \omega_i t + \phi_i, & if \quad i = 0 \\ \sin(\omega_i t + \phi_i), & if \quad 1 <= i <= k \end{cases}$$
- 另一方面，由于数据的稀疏性以及噪声影响，本文将 Category Embedding 和 Time Embedding 进行拼接，探索 POI 类别的时间模式而非单个 POI
  - $e_c = f_{embed}(c) \in R^{\Psi}$
  - $e_{c,t} = \sigma(w_{c,t}[e_t; e_c] + b_{c,t}) \in R^{\Psi\times 2}$
- 从而将 check-in q=<p,u,t> 转换为向量 $e_q = [e_{p,u}; e_{c,t}]$ 作为 transformer 的输入

(11) Transformer Encoder: 主干网络使用 transformer，对一个输入序列 $S_u = (q_u^1, q_u^2, ..., q_u^k)$, 需要预测下一个活动 $q_u^{k+1}$，经过 check-in Embedding 后，对 $q_u^i$ 可以得到 $\chi^{[0]} \in R^{k \times d}$ 作为 transformer 的输入

- attention 操作：
  - $S = \chi^{[l]} W_q (\chi^{[l]} W_k)^T \in R^{k \times k}$
  - $S'_{i,j} = \frac{exp(S_{i,j})}{\sum_{j=1}^{d} exp(S_{i,j})}$
  - $head_1 = S' \chi^{[l]} W_v \in R^{k \times d/h}$
  - Multihead($\chi^{[l]}$)=$[head_1; ...; head_h] \times W_o \in R^{k \times d}$
- 之后是残差连接、LayerNorm，FFN：
  - $\chi_{attn}^{[l]} = LayerNorm(\chi^{[l]} + Multihead(\chi^{[l]}))$
  - $\chi_{FC}^{[l]} = ReLU(W_1 \chi_{attn}^{[l]} + b_1) W_2 + b_2 \in R^{k \times d}$
  - $\chi^{[l+1]} = LayerNorm(\chi_{attn}^{[l]} + \chi_{FC}^{[l]}) \in R^{k \times d}$

(12) MLP Decoders:

- 通过 Transformer Encoder 后得到输出 $\chi^{[l^*]}$, 后经多层感知机将输出分别映射到三个 MLP heads：
  - $\hat{Y}_{poi} = \chi^{[l^*]} W_{poi} + b_{poi}$
  - $\hat{Y}_{time} = \chi^{[l^*]} W_{time} + b_{time}$
  - $\hat{Y}_{cat} = \chi^{[l^*]} W_{cat} + b_{cat}$
- 对于 $\hat{Y}_{poi}$ 本文同时将上文得到的概率转移矩阵与其结合:
  - $\hat{y}_{poi} = \hat{Y}_{poi}^{(k)} + \Phi^{p_k} \in R^{1 \times N}$
- 本文认为两次 check-in 之间的时间间隔波动可能很大, 对应的 POI 类别也可能不同, 用户应在接下来的 1 小时和 5 小时受到不同的建议，因此在预测下一个 POI 的同时也预测下一次的 check-in 时间，类型，即文中使用三个 MLP heads 的原因

(13) Loss: 由于文中计算了三个预测结果, 因此最终损失为加权和, 由于时间经过标准化后 $\in [0, 1]$, 最后计算损失时乘了 10 倍

- $L_{final} = L_{poi} + 10 \times L_{times} + L_{cat}$

(14) 实验部分：

- 数据集：
  - FourSquare:NYC,TKY
  - Gowalla:CA
- 评价指标:
  - Acc@k=$\frac{1}{m} \sum_{i=1}^{m} 1(rank <= k)$
  - MRR=$\frac{1}{m} \sum_{i=1}^{m} \frac{1}{rank}$

**Table 2: Performance comparison in Acc@k and MRR on three datasets**

| | NYC | | | | | TKY | | | | | CA | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR |
| MF | 0.0368 | 0.0961 | 0.1522 | 0.2375 | 0.0672 | 0.0241 | 0.0701 | 0.1267 | 0.1845 | 0.4861 | 0.0110 | 0.0442 | 0.0723 | 0.1190 | 0.0342 |
| FPMC | 0.1003 | 0.2126 | 0.2970 | 0.3323 | 0.1701 | 0.0814 | 0.2045 | 0.2746 | 0.3450 | 0.1344 | 0.0383 | 0.0702 | 0.1159 | 0.1682 | 0.0911 |
| LSTM | 0.1305 | 0.2719 | 0.3283 | 0.3568 | 0.1857 | 0.1335 | 0.2728 | 0.3277 | 0.3598 | 0.1834 | 0.0665 | 0.1306 | 0.1784 | 0.2211 | 0.1201 |
| PRME | 0.1159 | 0.2236 | 0.3105 | 0.3643 | 0.1712 | 0.1052 | 0.2278 | 0.2944 | 0.3560 | 0.1786 | 0.0521 | 0.1034 | 0.1425 | 0.1954 | 0.1002 |
| ST-RNN | 0.1483 | 0.2923 | 0.3622 | 0.4502 | 0.2198 | 0.1409 | 0.3022 | 0.3577 | 0.4753 | 0.2212 | 0.0799 | 0.1423 | 0.1940 | 0.2477 | 0.1429 |
| STGN | 0.1716 | 0.3381 | 0.4122 | 0.5017 | 0.2598 | 0.1689 | 0.3391 | 0.3848 | 0.4514 | 0.2422 | 0.0810 | 0.1842 | 0.2579 | 0.3095 | 0.1675 |
| STGCN | 0.1799 | 0.3425 | 0.4279 | 0.5214 | 0.2788 | 0.1716 | 0.3453 | 0.3927 | 0.4763 | 0.2504 | 0.0961 | 0.2097 | 0.2613 | 0.3245 | 0.1712 |
| PLSPL | 0.1917 | 0.3678 | 0.4523 | 0.5370 | 0.2806 | 0.1889 | 0.3523 | 0.4150 | 0.4880 | 0.2542 | 0.1072 | 0.2278 | 0.2995 | 0.3401 | 0.1847 |
| STAN | 0.2231 | 0.4582 | 0.5734 | 0.6328 | 0.3253 | 0.1963 | 0.3798 | 0.4464 | 0.5119 | 0.2852 | 0.1104 | 0.2348 | 0.3018 | 0.3502 | 0.1869 |
| Ours | **0.2435** | **0.5089** | **0.6143** | **0.6880** | **0.3621** | **0.2254** | **0.4417** | **0.5287** | **0.5829** | **0.3262** | **0.1357** | **0.2852** | **0.3590** | **0.4241** | **0.2103** |

图 3:

- Results: 图三
- Inactive users and active users: 本文根据用户的活跃情况 (check-in 数量) 进行排序，分析不同活跃程度的用户对模型所带来的影响：图 4

**Table 3: Cold Start (due to inactive users) performance on NYC**

| User Groups | Model | Acc@1 | Acc@5 | Acc@10 | Acc@20 |
|---|---|---|---|---|---|
| Inactive | STGN | 0.0926 | 0.2435 | 0.3124 | 0.3763 |
| Normal | STGN | 0.1778 | 0.3576 | 0.4189 | 0.5125 |
| Very active | STGN | 0.1813 | 0.3832 | 0.4274 | 0.5398 |
| Inactive | Ours | 0.1224 | 0.3471 | 0.4394 | 0.4529 |
| Normal | Ours | 0.2421 | 0.4739 | 0.5422 | 0.6430 |
| Very active | Ours | 0.2692 | 0.5639 | 0.6995 | 0.7762 |

图 4:

- Short trajectories and long trajectories:
  - 本文对短轨迹下的挑战进行了实验：图 5

**Table 4: Cold Start (due to short trajectory) performance on NYC**

| Trajectory | Model | Acc@1 | Acc@5 | Acc@10 | Acc@20 |
|---|---|---|---|---|---|
| Short trajs | STGN | 0.0723 | 0.2117 | 0.2784 | 0.3328 |
| Middle trajs | STGN | 0.1921 | 0.3471 | 0.4397 | 0.5231 |
| Long trajs | STGN | 0.1934 | 0.3516 | 0.4380 | 0.5322 |
| Short trajs | Ours | 0.2186 | 0.4561 | 0.5269 | 0.5782 |
| Middle trajs | Ours | 0.2441 | 0.4927 | 0.5881 | 0.6519 |
| Long trajs | Ours | 0.2452 | 0.5378 | 0.6698 | 0.7681 |

图 5:

  - 移除 trajectory flow map 后的实验结果：图 6
- 消融实验: 图 7

(15) 未来工作：

**Table 5: Performance of proposed model without trajectory flow map on NYC**

| | Model | Acc@1 | Acc@5 | Acc@10 |
|---|---|---|---|---|
| Inactive users | Ours w/o graph | 0.1105 | 0.2983 | 0.3470 |
| Normal users | Ours w/o graph | 0.2094 | 0.4228 | 0.5010 |
| Very active users | Ours w/o graph | 0.2488 | 0.5271 | 0.6815 |
| Short trajs | Ours w/o graph | 0.1914 | 0.4059 | 0.4721 |
| Middle trajs | Ours w/o graph | 0.2129 | 0.4490 | 0.5232 |
| Long trajs | Ours w/o graph | 0.2012 | 0.4882 | 0.6186 |

图 6:

**Table 6: Ablation study: Comparing the full model with 6 variants**

| | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR |
|---|---|---|---|---|---|
| **Full Model** | **0.2435** | **0.5089** | **0.6143** | **0.6880** | **0.3621** |
| w/o Graph | 0.2163 | 0.4512 | 0.5248 | 0.6402 | 0.3398 |
| w/o Transformer | 0.2221 | 0.4617 | 0.5520 | 0.6261 | 0.3317 |
| w/o Time&Cat | 0.2296 | 0.4804 | 0.5489 | 0.6484 | 0.3495 |
| w/o GCN | 0.2062 | 0.4582 | 0.5601 | 0.6589 | 0.3187 |
| w/o Fusion | 0.2355 | 0.4731 | 0.5726 | 0.6614 | 0.3533 |
| Single decoder | 0.2207 | 0.4721 | 0.5722 | 0.6591 | 0.3443 |

图 7:

- 进一步探索和区分时间模式，例如工作日和周末之间
- 根据用户的行为对用户进行分类，并为每种类型的用户构建轨迹流图

(16) 总结：本文的框架是 transformer，主要是通过构建 POI 之间的转移权重图 (trajectory flow map) 并通过 GCN 进行 POI Embedding，最后，又同时预测 POI，时间，类别，加强了损失函数

**2.Hierarchical Multi-Task Graph Recurrent Network for Next POI Recommendation**

(1) 研究内容：在可能的 POI 的搜索空间很大的情况下，预测用户接下来将访问哪个兴趣点 (POI)，为每个用户预测一个排名的 POI 集

(2) 文章整体要点：

- 现有的问题：用户兴趣点矩阵很稀疏，仅学习 User-POI 矩阵进行预测，使得学习和表现变得困难
- 提出了分层多任务图递归网络（HMT-GRN）方法，该方法通过在多任务设置中学习不同的低稀疏用户区域矩阵来缓解数据稀疏问题。
  - 对不同的区域和 POI 分布执行分层波束搜索 (HBS)，以随着空间粒度的增加分层减少搜索空间并预测下一个 POI。本文的 HBS 通过减少搜索空间来提高效率 (加速)
  - 提出了一种新颖的选择性层来预测用户之前是否访问过下一个 POI，以在 POI 推荐的个性化和探索之间取得平衡

- 现有工作中的问题是 User-POI 矩阵的高度稀疏性 (因为用户通常只会访问作为搜索空间的数据集中所有 POI 中的几个首选 POI)，这使得学习和准确预测用户未来将访问的下一个 POI 变得困难。
- 提出了一种分层多任务图循环网络（HMT-GRN）来学习用户-兴趣点矩阵，同时也学习了几个不同粒度级别的用户-区域矩阵，以更好地学习稀疏的用户-兴趣点关系。
  - 提出了 HMT-GRN 模型，以多任务学习的形式学习 User-POI 和 User-G@P 矩阵，预测下一个 POI 和 G@P 区域，然后执行 Hierarchical Beam 对学习到的任务分布进行搜索（HBS），以分层减少搜索空间并提高预测下一个 POI 的效率
  - 为了在个性化和探索之间取得平衡, 提出了一个选择性层来预测下一个 POI 是历史访问过的 POI，还是用户未访问过的 POI，从而分别执行个性化和探索, 从而在个性化和探索之间取得平衡
  - 提出了图形递归网络（GRN）模块，通过考虑 POI-region 和 POI-timeslot 关系的空间和时间图，同时学习 POI 访问序列中的顺序依赖性和全局时空 POI-POI 关系，以缓解稀疏性
- 本文的贡献：
  - 提出了一种新颖的 HMT-GRN 模型，对 next-POI 推荐任务通过学习 User-POI 和不同的 User-Region 矩阵来缓解数据稀疏问题
  - HMT-GRN 模型包括 next-POI 和下一个区域或 G@P 的多任务学习，作为搜索空间缩减方法的 HBS，以及在个性化和探索之间平衡的选择性层。此外，我们的 GRN 模块同时学习顺序依赖和全局时空 POI-POI 关系同时进行。

(3) 问题描述:

- 给定大小为 Q 的 POI 集合 $L = l_1, l_2, ..., l_Q$, 大小为 M 的的用户集合 $U = u_1, u_2, ..., u_M$ 和大小为 M 的所有用户的访问序列集 $S = S_{u_1}, S_{u_2}, ..., S_{u_M}$, 其中每个用户的序列 $S_{u_m}$ 由连续的 POI 访问 $S_{u_m} = \{(l_{t_1}, loc_{t_1}, time_{t_1}), (l_{t_2}, loc_{t_2}, time_{t_2}), ..., (l_{t_i}, loc_{t_i}, time_{t_i})\}$ 组成，其中 $l_{t_i}$ 是在时间步 $t_i$ 上访问的 POI，其对应的位置坐标为 $loc_{t_i}$ 对应的时间戳为 $time_{t_i}$, 将每个用户划分为训练集 (train) 和测试集 (test),(eg.$S_{u_m}^{train}, S_{u_m}^{test}$)
- Next POI Recommendation: 对给定的 $u_m$，其中 $S_{u_m}^{train}$=$\{(l_{t_1}, loc_{t_1}, time_{t_1}), (l_{t_2}, loc_{t_2}, time_{t_2}), ..., (l_{t_{i-1}}, loc_{t_{i-1}}, time_{t_{i-1}})\}$,next-POI 推荐任务是考虑 POI 的搜索空间, 以计算时间步 $t_i$ 的 netx-POI 排序集 $y_{t_i}$ , 其中 next-POI:$l_{t_i}$ 为排序集 $y_{t_i}$ 中排名靠前的 POI

(4) HMT-GRN 模型架构：图 8

(5) HMT-RN:

- Learning next POI and region distributions:
  - 如图 9，通过学习稀疏的 User-POI 矩阵以及具有较低数据稀疏性的 User-G@P 矩阵来更好地执行 next-POI 推荐任务。其中不仅预测 next-POI，还预测 next-POI 所在的下一个区域 G@p

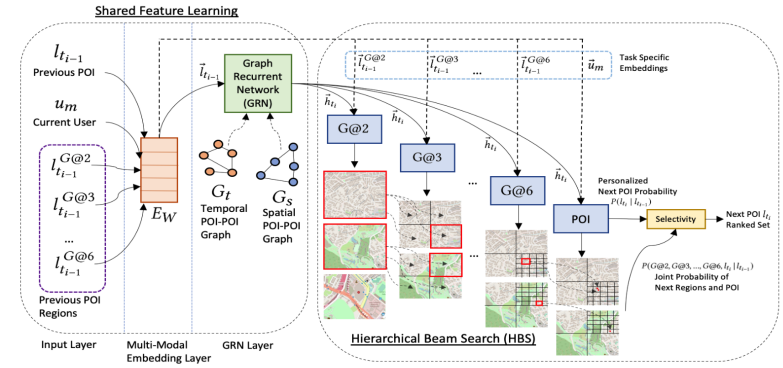
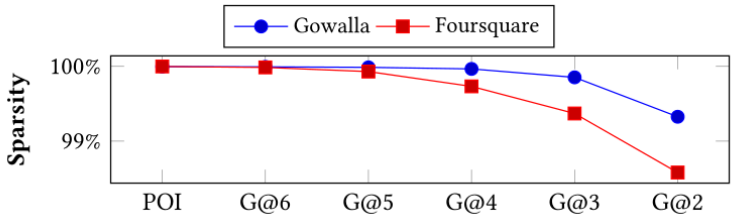



**Figure 3: Illustration of the HMT-GRN model that includes shared feature learning by our GRN module, followed by the multi-task learning of next regions and POI, then performing our HBS and selectivity layer. An example of top-$\beta = 2$ (red boxes) is used by HBS to traverse the multi-task distributions and reduce search space. Maps © OpenStreetMap contributors, CC BY-SA.**

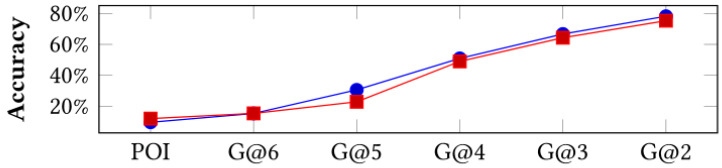
图 8:

- 在时间 $t_i$，使用多模态嵌入层 $E_W$ 映射到 $u_m, l_{t_{i-1}}, l_{t_{i-1}}^{G@P}$ 的可训练向量表示:
  - $\vec{u}_m, \vec{l}_{t_{i-1}}, \vec{l}_{t_{i-1}}^{G@P} = E_W(u_m, l_{t_{i-1}}, l_{t_{i-1}}^{G@P})$
- 使用 LSTM 层 $\Phi$ 来学习 POI 序列之间的顺序依赖关系，之前的 POI 嵌入 $\vec{l}_{t_{i-1}}$ 作为输入:
  - $\vec{h}_{t_i} = \Phi(\vec{l}_{t_{i-1}})$
- 执行 softmax 归一化以计算 next-POI 任务分布的条件概率 $tk^{POI} = P(l_{t_i}|l_{t_{i-1}})$:
  - $tk^{POI} = softmax(DL_{W_{L1}}(DO(\vec{h}_{t_i} \bigoplus \vec{u}_m)))$
- 因此，排序集 $y_{t_i} = \Psi(tk^{POI})$ 可以通过应用排序函数 $\Psi(\cdot)$ 对 $tk^{POI}$ 进行降序排序来计算，以用于 next-POI 推荐任务
- 除了 next-POI 任务外，还执行多任务学习来预测 next G@P $l_{t_i}^{G@P}$，在这里，使用前一个任务特定的嵌入 POI G@P 代表公式 (7) 中的 $l_{t_{i-1}}^{G@P}$，而非用户嵌入 $\vec{u}_m$ 来预测下一个对应的 $l_{t_i}^{G@P}$，从而计算任务分布 $tk^{G@P} = P(l_{t_i}^{G@P}|l_{t_{i-1}})$
  - $tk^{G@P} = softmax(DL_{W_{L@P}}(DO(\vec{h}_{t_i} \bigoplus \vec{l}_{t_{i-1}}^{G@P})))$
- 这可以解释为使用前一个区域来帮助预测下一个具有相同 P 或网格单元大小的区域，用于每个 next G@P 任务。对于所有任务 TK，使用 $\vec{h}_{t_i}$ 作为共享特征学习的通用表示

- Training:
  - 对于每个任务 $tk \in G@2, G@3, ..., G@6, POI$，交叉熵损失为 $L_{tk} = 1\sum_{v=1}^{N_{train}} log(pb_v)$
  - 整体监督损失 $L = \frac{1}{|TK|}\sum L_{tk}$ 以相等的权重计算，不会偏向任何任务
- Hierarchical Beam Search:
  - 不是单独的使用稀疏的 next-POI 任务分布 $tk^{POI} = P(l_{t_i}|l_{t_{i-1}})$ 进行预测，而是利用学习到的 next-POI 和 G@P 任务分布，通过计算所有任务的联合概率对来自 L 的 POI 的搜索空间进行排序，并预测下一个 POI
  - 首先，提出了一个层次空间图 $G_{hs}$，表示为 $G_{hs} = (V_{hs}, E_{hs})$ 的有向图，其中 $V_{hs}$ 和 $E_{hs}$ 分别是所有任务的顶点和边的集合,$G_{hs}$ 将多任务分布表示为具有空间层次




**(a) Decreasing sparsity levels as region size increases: Comparison of the User-POI and User-G@$P$ matrices where $P \in \{2, 3, 4, 5, 6\}$. As $P$ increases, the region or cell size decreases.**




**(b) Increasing predictive accuracy as sparsity levels decreases: Comparison of classification accuracy of the next POI and G@$P$ region tasks with a LSTM baseline of the same experimental setup.**

**Figure 1: Incorporating different region sizes alleviates the data sparsity problem for next POI recommendation.**

图 9:

结构的图，在层次结构中。每个任务顶点 $v_{hs} \in V_{hs}$ 使用其各自任务分布的概率得分加权，将用于搜索算法。

- 通过将任务分布表示为分层空间图，可以执行穷举搜索，或等效的广度优先搜索 (BFS)，通过遍历图 $G_{hs}$ 的所有路径来对 L 中的所有 POI 进行排名以计算对数概率的总和 P, 但时间开销较大
- 本文提出了一种分层光束搜索 (HBS) 方法，仅在遍历期间扩展每个任务分布的 top-$\beta$ 有希望的顶点，具体来说，给定每对连续的 $(tk_{i-1}, tk_i)$ 任务分布，计算：
  - $tk_i^{\beta} = f(\{log(tk_{i-1,b}^{\beta}) + log(tk_{i,j}) | tk_{i,j} \in N(tk_{i-1,b}^{\beta}), b \in 1, 2, ..., \beta\})$
- 对于先前输入任务分布 $tk_{i-1,b}^{\beta}$ 的每个顶梁 b，从下一个输入任务分布 $tk_i$ 识别其（有向）邻域 $tk_{i,j}$，并计算其每个分层连接的 j 顶点的对数概率之和
- 在计算了当前迭代的所有部分解之后，使用函数 f( · ) 只考虑 top-$\beta$ 部分解来计算 $tk_i^{\beta}$ ，其保留了其遍历顶点的总对数概率，并将用于为下一次迭代
- 由图 10 看出 HBS 可以显着减少搜索空间以减少噪声。即使用最后一次迭代的输入对 $(tk_{i-1}^{G@6}, tk_i^{POI})$，搜索空间减少到仅考虑 $top - \beta$ 至 $tk_{i-1}^{G@6}$ 的 POI 或单元格，而不是全部 |L| 的 POI，以及计算它们各自的联合概率 P 以得出排名集

- Selectivity Layer:
  - 为了在个性化和探索之间取得平衡，提出了一个新颖的选择性层，通过预测用户之前是否访问过 next-POI，从而确定模型进行个性化或探索

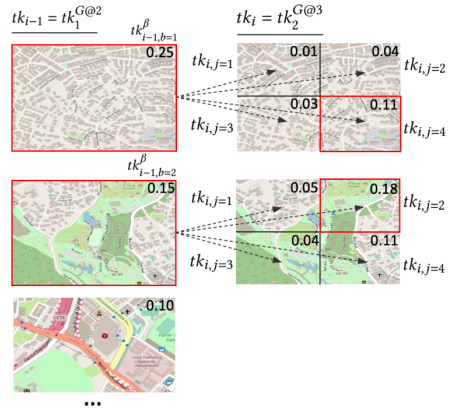



**Figure 2: Hierarchical Beam Search performed with top-$\beta = 2$ (red boxes) for the sequential pair of input tasks $(tk_{i-1} = tk_1^{G@2}, tk_i = tk_2^{G@3})$. Each vertex is weighted with its respective task probability score (top right of each vertex), for the computation of partial solutions, then ranking them to output $tk_i^{\beta}$. Maps © OpenStreetMap contributors, CC BY-SA.**

图 10:

- 只需从方程中的 next-POI 任务分布 $tk^{POI}$ 检索下一个预测的 POI$\hat{l}_{t_i} = argmax_{l_{t_i}}(tk^{POI})$ 并计算:

$$y_{t_i} = \begin{cases} \Psi(P(l_{t_i}|l_{t_{i-1}})), & \hat{l}_{t_i} \in s_{u_m}^{train} \\ \Psi(P(G@2, G@3, ..G@6, l_{t_i}|l_{t_{i-1}})), & otherwise \end{cases}$$

- 从 POI 的概率分数中降序排列，若用户之前已经访问过预测的 next-POI，则选择性层通过使用 next-POI 任务分布 $tk^{POI}$ 来个性化计算排名集，因为其包括使用用户嵌入来从 $S_{u_m}^{train}$ 中更好地捕获用户偏好。否则，其会通过执行 HBS 使用多个任务的联合概率 P 来探索基于区域上下文

(5) Graph Recurrent Network：

- 提出了 GRN 模块来替换 HMT-RN 模型中的 LSTM，以允许额外学习全局 POI-POI 关系, 通过增加了循环结构和通过使用区域和时隙连接时空图中的 POI 来缓解数据稀疏性从而扩展了 Dimensional GAT (DGAT) 变体
- 首先, DGAT 变体定义为:
  - $\vec{\alpha}_{l_{t_{i-1}},j} = \frac{exp(LeakyReLU(a[W_p\vec{l}_{t_{i-1}} \oplus W_p\vec{j}]))}{\sum_{\vec{k}\in\hat{N}_G[l_{t_{i-1}}]} exp(LeakyReLU(a[W_p\vec{l}_{t_{i-1}} \oplus W_p\vec{k}]))}$
  - $p_{t_i} = \sum_{\vec{j}\in\hat{N}_G[l_{t_{i-1}}]} \vec{\alpha}_{l_{t_{i-1}},j} \bigodot W_p\vec{j}$
- 接下来, 分别基于 POI-Region 和 POI-Timeslot 关系连接 POI，以减少数据稀疏性的影响。具体来说：
  - Spatial Graph: 一个无向且未加权的 POI-POI 图，表示为 $G_s = (V_s, E_s)$。如果每对 POI 在同一个 G@4 网格单元内，则它们具有邻接关系

  - Temporal Graph: 首先将每天划分为 8 个时隙, 每个时隙为 3h, 总共 56 个时隙, 然后将 $S^{train}$ 中的每次访问映射到其相应的时间段, 如果每对 POI 时隙集的 Jaccard 相似度 $\frac{v_{t_i}^{slot} \bigcap v_{t_j}^{slot}}{v_{t_i}^{slot} \bigcup v_{t_j}^{slot}}$ 在 0.9 以上，则它们具有邻接关系

- 为了对全局 POI-POI 关系建模，将方程式中的 DGAT 层缩写:
  - $p_{t_i}^{G_s} = \Gamma_s(\vec{l}_{t_{i-1}})$
  - $p_{t_i}^{G_t} = \Gamma_t(\vec{l}_{t_{i-1}})$
- 为了同时学习顺序依赖关系，通过修改等式将计算表示包含在循环结构中:
  - $i_{t_i} = \sigma(W_i x_{t_i} + U_i h_{t_{i-1}} + b_i + V_i p_{t_i}^{G_s} + Z_i p_{t_i}^{G_t})$
  - $f_{t_i} = \sigma(W_f x_{t_i} + U_f h_{t_{i-1}} + b_f + V_f p_{t_i}^{G_s} + Z_f p_{t_i}^{G_t})$
  - $o_{t_i} = (W_o x_{t_i} + U_o h_{t_{i-1}} + b_o + V_o p_{t_i}^{G_s} + Z_o p_{t_i}^{G_t})$
  - $\widetilde{c}_{t_i} = (W_c x_{t_i} + U_c h_{t_{i-1}} + b_c + V_c p_{t_i}^{G_s} + Z_c p_{t_i}^{G_t})$

(6) 实验:

- 数据集:
  - Foursquare
  - Gowalla
- 评价指标:
  - Acc@k=$\frac{1}{m}\sum_{i=1}^{m} 1(rank <= k)$
  - MRR=$\frac{1}{m}\sum_{i=1}^{m} \frac{1}{rank}$
  - 使用 $N^2 - Acc@K$ 和 $N^2 - MRR$ 的 $N^2$(确保推荐系统不仅总是正确地推荐历史访问过的 POI，而且还推荐用户将来访问的新的未访问过的 POI) 扩展来仅评估下一个未访问或新的 POI 推荐
- Result: 图 11

**Table 2: Performance in Acc@$K$ and MRR for all next POI test samples (i.e. visited and unvisited POIs), as well as the corresponding $N^2$ metrics of $N^2$-Acc@$K$ and $N^2$-MRR for only unvisited next POI test samples.**

| Gowalla | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR | $N^2$-Acc@1 | $N^2$-Acc@5 | $N^2$-Acc@10 | $N^2$-Acc@20 | $N^2$-MRR |
| TOP | 0.0084 | 0.0351 | 0.0678 | 0.1022 | 0.0270 | 0.0068 | 0.0281 | 0.0574 | 0.0874 | 0.0227 |
| U-TOP | 0.1423 | 0.2767 | 0.3035 | 0.3110 | 0.1986 | 0 | 0 | 0 | 0 | 0 |
| MF | 0.0644±0.001 | 0.0785±0.001 | 0.0825±0.001 | 0.0879±0.001 | 0.0736±0.001 | 0.0015±0.001 | 0.0032±0.001 | 0.0046±0.001 | 0.0073±0.001 | 0.0040±0.001 |
| RNN | 0.0844±0.002 | 0.1873±0.001 | 0.2440±0.001 | 0.3050±0.001 | 0.1381±0.001 | 0.0356±0.001 | 0.1045±0.001 | 0.1496±0.001 | 0.2034±0.001 | 0.0746±0.001 |
| GRU | 0.0865±0.001 | 0.1869±0.001 | 0.2489±0.001 | 0.3161±0.001 | 0.1406±0.001 | 0.0367±0.001 | 0.1064±0.001 | 0.1563±0.001 | 0.2150±0.001 | 0.0773±0.001 |
| LSTM | 0.0968±0.001 | 0.1968±0.001 | 0.2575±0.001 | 0.3276±0.002 | 0.1510±0.001 | 0.0419±0.001 | 0.1140±0.001 | 0.1641±0.001 | 0.2291±0.001 | 0.0845±0.001 |
| HST-LSTM | 0.0087±0.001 | 0.0366±0.001 | 0.0636±0.002 | 0.1004±0.001 | 0.0279±0.001 | 0.0069±0.001 | 0.0293±0.001 | 0.0545±0.002 | 0.0854±0.001 | 0.0233±0.001 |
| STGCN | 0.0313±0.001 | 0.0909±0.003 | 0.1351±0.005 | 0.1955±0.004 | 0.0684±0.001 | 0.0126±0.001 | 0.0460±0.002 | 0.0777±0.003 | 0.1269±0.003 | 0.0374±0.001 |
| LSTPM | 0.1297±0.001 | 0.2282±0.001 | 0.2720±0.001 | 0.3200±0.002 | 0.1803±0.001 | 0.0353±0.001 | 0.0869±0.002 | 0.1199±0.002 | 0.1613±0.003 | 0.0633±0.001 |
| STAN | 0.0939±0.002 | 0.1928±0.003 | 0.2440±0.003 | 0.3039±0.006 | 0.1460±0.002 | 0.0143±0.001 | 0.0513±0.002 | 0.0843±0.002 | 0.1323±0.005 | 0.0398±0.001 |
| STP-UDGAT | 0.1194±0.001 | 0.2374±0.001 | 0.2783±0.001 | 0.3202±0.002 | 0.1770±0.001 | 0.0251±0.001 | 0.0712±0.001 | 0.1014±0.001 | 0.1393±0.002 | 0.0517±0.001 |
| Flashback | 0.1266±0.001 | 0.2342±0.001 | 0.2770±0.002 | 0.3285±0.001 | 0.1821±0.001 | 0.0153±0.001 | 0.0517±0.002 | 0.0825±0.003 | 0.1288±0.001 | 0.0412±0.001 |
| HMT-HN | 0.1434±0.001 | 0.2677±0.001 | 0.3213±0.001 | 0.3781±0.001 | 0.2053±0.001 | 0.0525±0.001 | 0.1334±0.001 | 0.1888±0.001 | 0.2536±0.001 | 0.0987±0.001 |
| HMT-GRN | **0.1455±0.001** | **0.2783±0.001** | **0.3394±0.001** | **0.4033±0.001** | **0.2120±0.001** | **0.0539±0.001** | **0.1369±0.001** | **0.1920±0.001** | **0.2579±0.001** | **0.1008±0.001** |
| Relative Improvement | 2.2% | 0.6% | 11.8% | 22.8% | 6.7% | 28.6% | 20.1% | 15.6% | 12.6% | 19.6% |
| Foursquare | | | | | | | | | | |
| | Acc@1 | Acc@5 | Acc@10 | Acc@20 | MRR | $N^2$-Acc@1 | $N^2$-Acc@5 | $N^2$-Acc@10 | $N^2$-Acc@20 | $N^2$-MRR |
| TOP | 0.0082 | 0.0353 | 0.0546 | 0.0869 | 0.0263 | 0.0056 | 0.0247 | 0.0373 | 0.0604 | 0.0192 |
| U-TOP | **0.1690** | 0.3297 | 0.3796 | 0.3979 | 0.2382 | 0 | 0 | 0 | 0 | 0 |
| MF | 0.0687±0.001 | 0.0859±0.001 | 0.0905±0.001 | 0.0954±0.001 | 0.0789±0.001 | 0.0009±0.001 | 0.0028±0.001 | 0.0045±0.001 | 0.0061±0.001 | 0.0031±0.001 |
| RNN | 0.1078±0.001 | 0.2246±0.001 | 0.2973±0.002 | 0.3752±0.001 | 0.1700±0.001 | 0.0444±0.001 | 0.1272±0.001 | 0.1858±0.001 | 0.2536±0.001 | 0.0914±0.001 |
| GRU | 0.1103±0.001 | 0.2300±0.001 | 0.3027±0.001 | 0.3852±0.002 | 0.1740±0.001 | 0.0459±0.001 | 0.1306±0.001 | 0.1908±0.001 | 0.2644±0.001 | 0.0945±0.001 |
| LSTM | 0.1191±0.001 | 0.2437±0.001 | 0.3174±0.001 | 0.4032±0.002 | 0.1854±0.001 | 0.0505±0.001 | 0.1409±0.001 | 0.2035±0.001 | 0.2828±0.001 | 0.1023±0.001 |
| HST-LSTM | 0.0076±0.001 | 0.0307±0.001 | 0.0500±0.001 | 0.0806±0.001 | 0.0244±0.001 | 0.0058±0.001 | 0.0218±0.001 | 0.0369±0.001 | 0.0591±0.001 | 0.0181±0.001 |
| STGCN | 0.0276±0.002 | 0.0948±0.005 | 0.1531±0.006 | 0.2323±0.005 | 0.0703±0.003 | 0.0114±0.001 | 0.0497±0.002 | 0.0896±0.003 | 0.1503±0.003 | 0.0400±0.001 |
| LSTPM | 0.1478±0.001 | 0.2671±0.002 | 0.3214±0.002 | 0.3778±0.001 | 0.2078±0.001 | 0.0426±0.001 | 0.1052±0.001 | 0.1466±0.001 | 0.1951±0.001 | 0.0782±0.001 |
| STAN | 0.1066±0.004 | 0.2382±0.007 | 0.3136±0.008 | 0.3987±0.010 | 0.1759±0.005 | 0.0249±0.002 | 0.0870±0.007 | 0.1384±0.008 | 0.2070±0.011 | 0.0643±0.004 |
| STP-UDGAT | 0.1397±0.001 | 0.2926±0.002 | 0.3556±0.002 | 0.4187±0.001 | 0.2136±0.001 | 0.0382±0.001 | 0.1156±0.002 | 0.1699±0.002 | 0.2324±0.001 | 0.0811±0.001 |
| Flashback | 0.1442±0.001 | 0.2768±0.002 | 0.3547±0.002 | 0.4012±0.001 | 0.2118±0.002 | 0.0229±0.001 | 0.0742±0.001 | 0.1185±0.001 | 0.1820±0.001 | 0.0577±0.001 |
| HMT-HN | 0.1617±0.001 | 0.3257±0.001 | 0.3961±0.001 | 0.4673±0.001 | 0.2415±0.001 | 0.0670±0.001 | 0.1738±0.001 | 0.2486±0.001 | 0.3357±0.001 | 0.1269±0.001 |
| HMT-GRN | 0.1673±0.001 | **0.3357±0.002** | **0.4148±0.001** | **0.4983±0.001** | **0.2510±0.001** | **0.0686±0.001** | **0.1756±0.002** | **0.2507±0.001** | **0.3386±0.001** | **0.1288±0.001** |
| Relative Improvement | -1.0% | 1.8% | 9.3% | 19.0% | 5.4% | 35.8% | 25.4% | 23.2% | 19.7% | 25.9% |

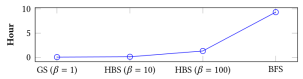
图 11:

-

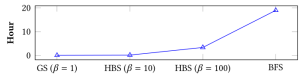
- Efficiency:
  - 比较了贪婪搜索（GS）和 HBS 关于使用 $N^2$ 指标的图 $G_{h_s}$ 的层次结构中在多个任务分布上的性能
  - 图 12

**Table 3: Performance comparison of search methods.**

| Dataset | Search Method | $N^2$-Acc@1 | $N^2$-Acc@5 | $N^2$-Acc@10 | $N^2$-Acc@20 | $N^2$-MRR |
|---|---|---|---|---|---|---|
| Gowalla | GS ($\beta = 1$) | 0.0408 | 0.0762 | 0.0911 | 0.1122 | 0.0592 |
| | BFS | 0.0537 | **0.1381** | 0.1913 | 0.2566 | **0.1008** |
| | HBS ($\beta = 10$) | 0.0536 | 0.1366 | 0.1857 | 0.2354 | 0.0966 |
| | HBS ($\beta = 100$) | **0.0539** | 0.1369 | **0.1920** | **0.2579** | **0.1008** |
| Foursquare | GS ($\beta = 1$) | 0.0473 | 0.0745 | 0.0820 | 0.0872 | 0.0615 |
| | BFS | 0.0672 | 0.1750 | 0.2504 | 0.3385 | 0.1285 |
| | HBS ($\beta = 10$) | 0.0671 | 0.1742 | 0.2444 | 0.3017 | 0.1205 |
| | HBS ($\beta = 100$) | **0.0686** | **0.1756** | **0.2507** | **0.3386** | **0.1288** |




**(a) Gowalla.**




**(b) Foursquare.**

**Figure 4: Efficiency comparison of search methods.**

图 12:

- Importance of proposed tasks：评估了 HBS 在 $N^2$ 指标上的性能: 图 13

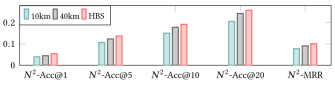
**Table 4: Effectiveness of proposed tasks for HBS.**

| Dataset | | $N^2$-Acc@1 | $N^2$-Acc@5 | $N^2$-Acc@10 | $N^2$-Acc@20 | $N^2$-MRR |
|---|---|---|---|---|---|---|
| Gowalla | All Tasks | **0.0539** | **0.1369** | 0.1920 | 0.2579 | **0.1008** |
| | G@2 + POI | 0.0490 | 0.1335 | 0.1883 | 0.2611 | 0.0980 |
| | G@3 + POI | 0.0517 | 0.1364 | **0.1943** | **0.2652** | 0.1004 |
| | G@4 + POI | 0.0471 | 0.1299 | 0.1871 | 0.2588 | 0.0946 |
| | G@5 + POI | 0.0459 | 0.1294 | 0.1815 | 0.2479 | 0.0931 |
| | G@6 + POI | 0.0462 | 0.1255 | 0.1739 | 0.2401 | 0.0912 |
| | POI | 0.0461 | 0.1215 | 0.1711 | 0.2347 | 0.0898 |
| Foursquare | All Tasks | **0.0686** | **0.1756** | **0.2507** | **0.3386** | **0.1288** |
| | G@2 + POI | 0.0597 | 0.1621 | 0.2303 | 0.3221 | 0.1183 |
| | G@3 + POI | 0.0605 | 0.1637 | 0.2341 | 0.3229 | 0.1193 |
| | G@4 + POI | 0.0593 | 0.1605 | 0.2295 | 0.3183 | 0.1167 |
| | G@5 + POI | 0.0589 | 0.1586 | 0.2253 | 0.3088 | 0.1152 |
| | G@6 + POI | 0.0582 | 0.1566 | 0.2211 | 0.3046 | 0.1137 |
| | POI | 0.0563 | 0.1535 | 0.2178 | 0.3009 | 0.1113 |

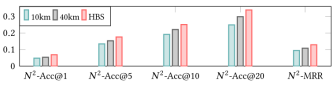
图 13:

–

- Search space reduction: 将 HBS 与现有的搜索空间缩减方法进行 $N^2$ 指标的比较: 图 14




**(a) Gowalla.**




**(b) Foursquare.**

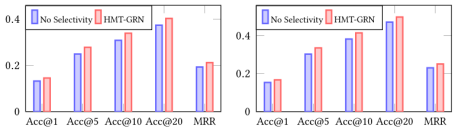
**Figure 5: Comparison of search space reduction methods.**

图 14:

–

- Selectivity layer: 从方程式评估选择性层的重要性: 图 15

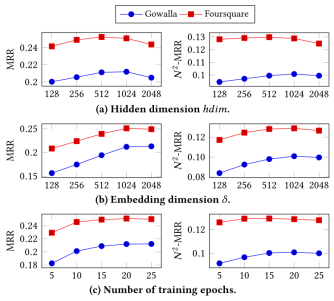
–




(a) Gowalla. (b) Foursquare.

**Figure 6: Impact of the selectivity layer.**

图 15:




(a) Hidden dimension $hdim$.

(b) Embedding dimension $\delta$.

(c) Number of training epochs.

**Figure 7: Sensitivity analysis of HMT-GRN.**

图 16:

- Sensitivity: 研究了 HMT-GRN 模型对不同超参数的敏感性: 图 16

  –

- Case Study: 看到了由 HMT-GRN 模型进行的真实世界测试样本预测，该模型正确预测了在其历史序列中经常访问机场的用户的下一个 POI: 图 17

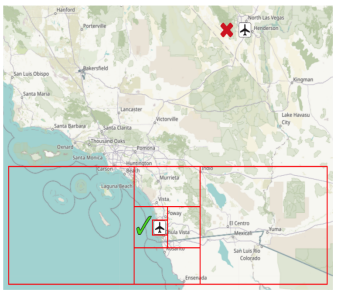


**Figure 8: Test sample prediction from the Gowalla dataset using our trained HMT-GRN model, predicts the next POI (airport) correctly, contrasting with the incorrect airport predicted by using the sparse POI task distribution $tk^{POI}$ directly. Maps © OpenStreetMap contributors, CC BY-SA.**

图 17:

–

(7) 未来工作：探索时间集中的任务，以帮助进一步减少数据稀疏性