综合运用 FMM、GRU 模型和 FPMC 模型解决到达时间预测和下一跳预测问题

李佳恒[1] 周靖宇[1] 曾元坤[1]

[1]（北京航空航天大学计算机学院 北京 100191）

Integrative Approach of FMM, GRU Model, and FPMC Model for Arrival Time Prediction and Next-Hop Prediction

Li Jiaheng[1] , Zhou Jingyu[1] , and Zeng Yuankun[1]

[1]（*College of Computer Science and Technology, Beihang University, Beijing* 100083）

Abstract In this study, we employed the Fast Marching Method (FMM) for network matching to address the first question. For the second question, a customized similarity function was introduced, allowing for a more flexible and personalized approach to similarity calculation. In the third question, the Gated Recurrent Unit (GRU) model was utilized to predict eta, enabling modeling of dynamic characteristics. Lastly, for the fourth question, the Factorized Personalized Markov Chains (FPMC) model was adopted for next-hop prediction, enhancing prediction accuracy and robustness by integrating both static and dynamic user features. The comprehensive application of these methods offers a holistic and effective solution to the research problem.

Key words FMM algorithm, GRU model, modeling dynamic characteristics, FPMC model, next-hop prediction, clustering

摘要

在本研究中，我们通过采用 FMM 算法实现了网络匹配，解决了第一问的问题。在第二问中，引入了自定义相似度函数，通过该函数进行相似度计算，为问题的求解提供了更加灵活和个性化的方法。第三问中，我们运用 GRU（门控循环单元）模型对 eta 进行预测，实现了对动态特性的建模。最后，第四问中，我们采用 FPMC（分解个性化马尔可夫链）模型进行下一跳预测，通过综合用户的静态和动态特性，提高了预测的准确性和鲁棒性。这些方法的综合运用为研究问题提供了全面而有效的解决方案。

关键词 FMM 算法、GRU 模型、动态特性建模、FPMC 模型、下一跳预测、聚类

中图法分类号 TP391

本次作业的主要内容是根据路况数据进行路网匹配，路段聚类，到达时间预测以及下一跳预测。

本文使用的主要方法如下:

1) 使用 fmm 进行网路匹配

2) 使用自定义的相似度系数利用 DBSCAN 算法进行聚类

3) 通过自定义路径编码以及 GRU 实现到达时间预测

4) 使用 FPMC 模型进行下一跳预测

1 任务一：路网匹配

我们选用 fmm 算法完成路网匹配任务。

1. 1 fmm 算法简要介绍

FMM 算法是一种快速地图匹配算法，主要用于处理由全球定位系统（GPS）传感器生成的大量数据，以在交通研究等领域进行应用。由于 GPS 轨迹中的观测误差，通常需要进行地图匹配（MM）过程，以推断在道路网络上的路径。为了设计高效可扩展的 MM 算法，文章提出了 FMM 算法，该算法整合了隐藏马尔可夫模型与预计算，并提供了开源实现。

FMM 算法分为两个阶段：预计算阶段和 MM 阶段。在预计算阶段，通过计算在道路网络中长度不超过某一阈值的所有最短路径，并将其存储在上界原目标表（UBODT）中。UBODT 以压缩的方式存储，包含了从起始节点到目标节点的最短路径信息。在 MM 阶

段，通过查询 UBODT 中的信息，替代了计算昂贵的路由查询，从而提高了算法的性能。

1. 2 fmm 算法的使用

使用一个现有的 FMM 算法的实现(https://github.com/cyang-kth/fmm)来进行路网匹配。

使用前先进行安装。根据其教程页进行下载和安装(https://fmm-wiki.github.io/docs/installation/)，其中 Linux 环境下安装较方便。Windows 下安装可以按照官方教程中的做法，在 cyg-win 中运行，也可以用 WSL 安装虚拟 Linux 环境。本次实验使用 cyg-win 安装运行。由于 cyg-win 安装包的缺陷，会发生大量应该已经安装的运行时库丢失，需要根据运行过程中的报错手动查找安装。由于它的 python 扩展只支持 python2.7，已经过时且与本地 python 环境冲突，故修改 CMakeLists.txt，去掉编译 python 扩展的 target，只编译 C++命令行程序。最后通过命令行调用相关程序。

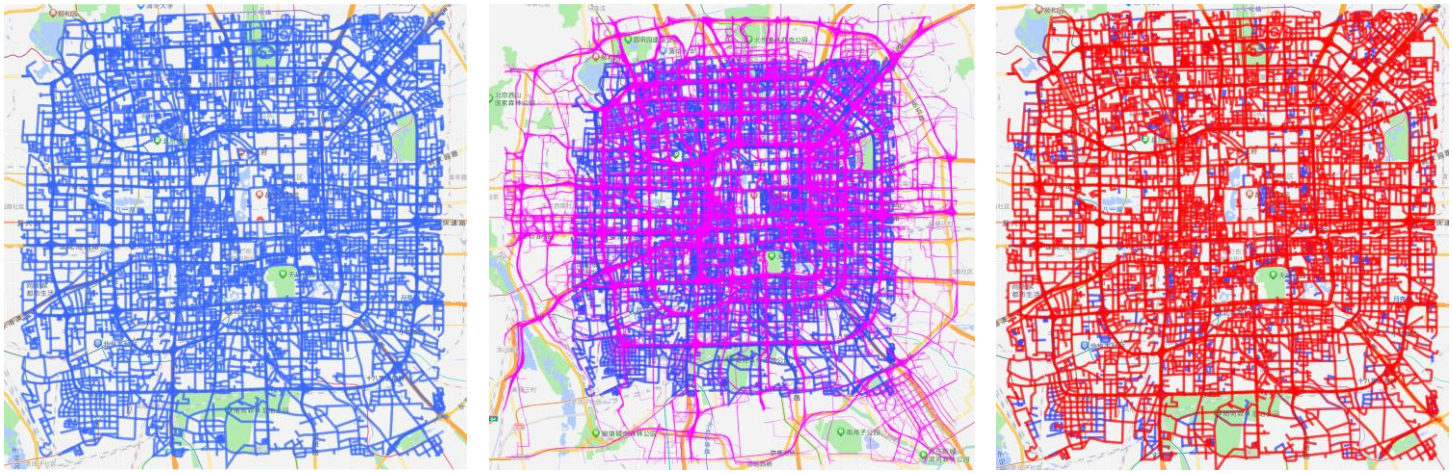
数据预处理。使用 fmm 前需要对数据预处理，主要是将路网、轨迹组织成它需要的格式。fmm 的几何图形（这里是每一条道路或轨迹形成的线）用 WKT 格式存储，需要提前吧路段/轨迹折线中的点的坐标以 WKT 折线格式存储下来。WKT 折线格式为“LINESTRING(x1 y1,x2 y2…)”。fmm 要求路网必须以.shp 的格式保存，而轨迹可以是.shp 也可以是.csv。然而经过尝试发现其.csv 读取器存在各种问题，故都转化为.shp，用 GDAL 的 ogr2ogr 命令完成转化。

fmm 预计算阶段。将准备好的路网.shp 文件输入给 ubodt_gen.exe，生成用于加速的 ubodt.txt 查询表。这个表的大小高达 10GB 以上，并且在之后运行 fmm 的时候要求全部读入内存，所以一般在虚拟机上不容易运行 fmm。实际上，尝试的时候也曾跑崩过一个 Ubuntu 虚拟机。

执行 fmm。准备好 fmm 所需的.shp 格式的轨迹、.shp 格式的路网、ubodt 查询表之后，直接运行 fmm.exe。在配置中选择输出所有结果域，结果包含每个轨迹 GPS 点匹配后的点、轨迹途径的道路编号等。输出格式是以分号分隔的.csv。

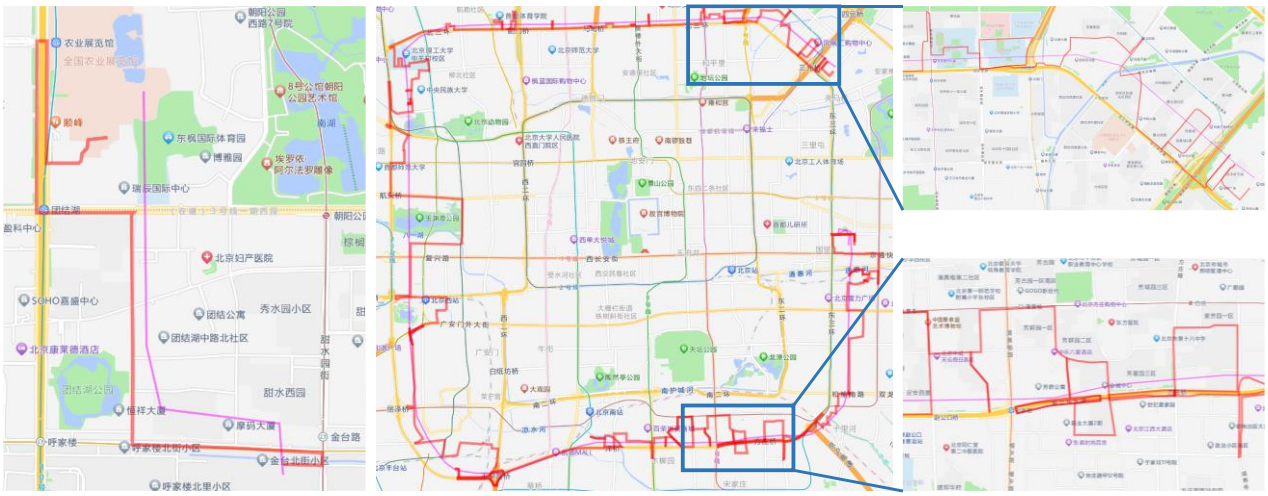
1. 3 fmm 路网匹配结果

traj.csv 中给出的一共 17601 条轨迹中，匹配成功了 16967 条，占比 96.40%。



(a) 给出的路网　(b) 给出的路网(蓝)与原始轨迹(紫)　(c) 路网中被匹配到的路(红)

图表 1 路网与原始轨迹可视化



图表 2 匹配结果的两例。紫色为原始路径，红色为匹配结果。

将提供的路网可视化后，发现其大约是北京四环内的道路。如图表 1 所示。图表 1(b)的中紫色线条为给出的原始的没有匹配的路径，可以看出其中有一部分超出了路网的范围，应当不能匹配成功，这也是匹配失败的不到 5%条路径的来源之一。图 1(c)中红色线条表示这条路段被匹配到了至少一个轨迹中去，可见绝大部分轨迹经过匹配后都至少出现在一条轨迹中。

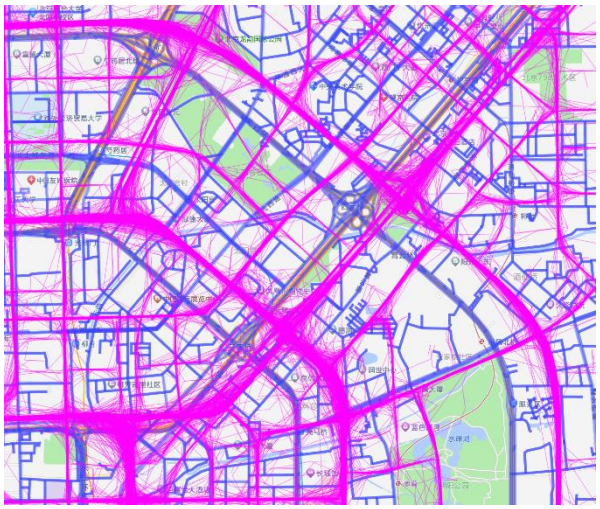
从细节来看，也就是去考察个别路径的匹配结果，图表 2 给出了两条示例轨迹与匹配结果的可视化，(a)匹配结果较好，(b)匹配结果较差。可以看出给出的数据和成功匹配的结果的一些问题。

1、轨迹数据有较大噪声。这是真实传感器得到的轨迹数据，有较大噪声。而从几个环路这样明显的细节来看，如 图表 3 所示，轨迹首先在整体上就存在一个较统一的偏差。而一些较粗的轨迹线周围大量很细且凌乱的轨迹线，是噪声和稀疏采样点的一个表现。

2、路网非常密集，轨迹在偏移后附近仍有大量道路，很可能诱导匹配到这些道路上，如图表 2(b,c)所示。肉眼来看，这条轨迹大致是沿着三环行驶，但偏移后，轨迹被大量匹配到了三环附近的小路，诱导出现了一些奇奇怪怪的路线。

3、接续 2 中所述，我们发现 fmm 容易被这些小路所“欺骗“，同时，其结果中也出现了许多难以解释的奇怪环路，如图表 2(c)中的两个局部所示，出现了很多环路，这在正常驾驶中应该是较少存在的，而类似的环路在所有路径的匹配结果中存在许多。fmm 的这些缺陷，在其 github 仓库的 issue 中也有其它人提及。

4、存在超出给出路网范围许多的轨迹，它们将匹配失败或者匹配出相当奇怪的路径，已在前文讨论过。



图表 3 环路折角附近的轨迹

1.4 总结

了解了一些经典的路网匹配算法，跑通了 fmm 算法，成功在所给出的路网和轨迹上运行 fmm 并得到了绝大部分轨迹的匹配结果，并进行了可视化的探究。虽然匹配结果并不完全理想，但至少也能继续进行接下来的任务了。

2 任务二：路段聚类

我们选择路段聚类任务作为任务二。所提供的路段属性与定义在 road.csv 中。

2.1 特征

提供的路段数据中包含了许多属性，包括车道数、宽度、限速、是否为隧道桥梁等，但这些属性都几乎无法使用。除了路段坐标、长度、路段类型外，剩下的属性要么几乎完全缺失，要么只有极少为 1，没有使用价值。具体而言，所提供的共 38027 个路段中，对于非 0-1 的属性，路段类型 highway 缺失 2669

个，车道数 lanes 缺失 36425 个，限速 maxspeed 缺失 37859 个，宽度 width 缺失 38021 个；对于 0-1 属性，若将全部 0 当作否，则只有 98 个为隧道，1658 个为桥梁，28 个为小巷，88 个为环岛。0-1 属性中为 1 的数量非常稀少，和现实路段中这些特殊路段客观上缺失很少一致。我们最后只保留路段坐标、路段类型等基本属性。

2.2 定义路段距离

定义距离是聚类问题的核心。应当注意，路段中天然存在一种分类，也就是其中的路段类型 highway 属性。所提供的文件中 road 类型的定义遵循 OSM 地图数据库中的定义，只涉及 13 种。同种类型的数据已经拥有相当程度的相似度，这种相似主要来自于道路的功能定位(比如，是城区主干道、乡道等)、位置(比如居民区、公园学校等)，也就是来自对其类型的定义方式。在此基础上，任务二中我们尝试加入对路段几何形态的考虑，让几何形态相近的路段有更高的相似度。

几何形态相似度. 几何形态相同的两条道路，应该只需要刚性变换就可以完全变成与对方几何形态一致。所谓刚性变化就是只涉及旋转和平移的变换。这一想法来自于图形学中经典的“尽量是刚体”(as rigid as possible, arap)思想，对于 mesh 中以任意顶点为中心、包含其邻接顶点的变换单元，arap 思想试图在对 mesh 做变形后仍保证每个这样的变换单元(每一个顶点都可以作为中心形成变换单元)都尽量只进行了刚性变换，从而使得变形更加自然。

下面定义两个路段间关于几何形态的距离。对于路段$A:<(x_{a1},y_{a1}),(x_{a2},y_{a2})\dots,(x_{am},y_{am})>=<\boldsymbol{a_1},\dots,\boldsymbol{a_m}>$与路段$B:<(x_{b1},y_{b1}),(x_{b2},y_{b2})\dots,(x_{bn},y_{bn})>=<\boldsymbol{b_1},\dots\boldsymbol{b_n}>$，每个坐标是路段折线中的折点。首先，将路段起点对齐到(0,0)，也就是$A^*:<(x^*_{a1},y^*_{a1}),(x^*_{a2},y^*_{a2})\dots,(x^*_{am},y^*_{am})>=<\boldsymbol{a^*_1},\dots,\boldsymbol{a^*_m}>$，$B^*:<(x^*_{b1},y^*_{b1}),(x^*_{b2},y^*_{b2})\dots,(x^*_{bm},y^*_{bm})>=<\boldsymbol{b^*_1},\dots,\boldsymbol{b^*_n}>$，其中$x^*_{ti}=x_{ti}-x_{t1},y^*_{ti}=y_{ti}-y_{t1}$。然后将两者折点的数量对齐，对于折点数量较少的路段，在相邻折点间插入新点来使得折点数量一致，插入点后，两个路段中的折点数量都是$l=\max(m,n)$。在实际实现中，折点数量相差过大的没有强行插新点，而是直接把几何形态相似度置 0，也就是形态距离为 1，任务二中将这个阈值设置为折点数量较少的那个路段的一半。最后，将其中的点转化为向量：

$$\mathrm{A}'=<\boldsymbol{a^*_{i+1}}-\boldsymbol{a^*_i}>=<\boldsymbol{a'_{i-1}}>,i=2,3,\dots,l$$

对路段 B 也做类似操作。最后形成的向量序列A',B'的长度是$l-1$。

最晚上述处理后，寻找一个变换，将对齐后的A'变换为B'，也就是寻找一个变换$\mathbf{T}_{\boldsymbol{A\to B}}$，使得$\mathbf{B}=\mathbf{AT}_{\mathbf{A\to B}}$即：

$$\begin{bmatrix}\boldsymbol{b'_1} & 1\\ \boldsymbol{b'_2}_{\dots} & 1\\ \boldsymbol{b'_{l-1}} & 1\end{bmatrix}=\begin{bmatrix}\boldsymbol{a'_1} & 1\\ \boldsymbol{a'_2}_{\dots} & 1\\ \boldsymbol{a'_{l-1}} & 1\end{bmatrix}\mathbf{T}_{\mathbf{A\to B}}\tag{2.1}$$

其中，矩阵中最后一列的 1 是将它变为齐次坐标。$\mathbf{T}_{\mathbf{A\to B}}$是一个三维方阵。

当$l-1\geq 3$的时候，矩阵$\mathbf{A}$可以求逆或者伪逆，则$\mathbf{T}_{\mathbf{A\to B}}=\mathbf{A^{-1}B}$，此时，如果路段 A 和 B 的几何形态一致，由于提前转化为了向量，只用一个旋转变换就能将路段 A 变为路段 B，此时矩阵$\mathbf{T}_{\mathbf{A\to B}}$应当是正交矩阵，矩阵每个元素的平方和应该是 3。当$l-1<3$的时候，无法求逆和伪逆，使用最小二乘法估计矩阵$\mathbf{T}_{\mathbf{A\to B}}$，也就是$\mathbf{T}_{\mathbf{A\to B}}=\mathrm{argmax}_{\mathbf{T}_{\mathbf{A\to B}}}\left|\left|\mathbf{B}-\mathbf{AT}_{\mathbf{A\to B}}\right|\right|_2$，此时，如果 A 和 B 的几何形态一致，最小二乘法得到的结果$\mathbf{T}_{\mathbf{A\to B}}$的元素平方和为$l-1$。

定义路段形态距离$\mathrm{Dist}_{\mathrm{shp}}(A,B)$与相似度$\mathrm{Sim}_{\mathrm{shp}}(A,B)$为：

$$\begin{aligned}\mathbf{T}_{\mathbf{A\to B}}&=\begin{cases}\mathbf{A^{-1}B}, & l-1\geq 3\\ \mathrm{argmax}_{\mathbf{T}_{\mathbf{A\to B}}}\left|\left|\mathbf{B}-\mathbf{AT}_{\mathbf{A\to B}}\right|\right|_2, & l-1<3\end{cases}\\ d&=\begin{cases}\left|\left|\mathbf{T}_{\mathbf{A\to B}}\right|\right|_2-3, & l-1\geq 3\\ \left|\left|\mathbf{T}_{\mathbf{A\to B}}\right|\right|_2-(l-1), & l-1<3\end{cases}\\ \mathrm{Dist}^{\mathrm{AB}}_{\mathrm{shp}}&=\frac{1}{1+e^{d+\beta}}\\ \mathrm{Dist}_{\mathrm{shp}}(A,B)&=\frac{\mathrm{Dist}^{\mathrm{AB}}_{\mathrm{shp}}+\mathrm{Dist}^{\mathrm{BA}}_{\mathrm{shp}}}{2}\\ \mathrm{Sim}_{\mathrm{shp}}(A,B)&=1-\mathrm{Dist}_{\mathrm{shp}}(A,B)\end{aligned}\tag{2}$$

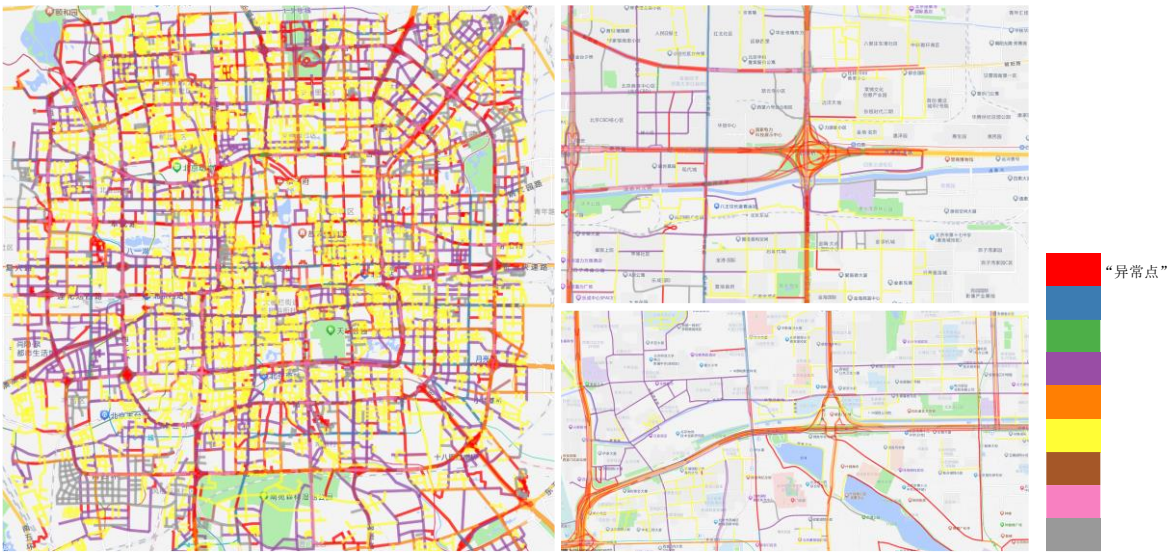
公式(2)中的第三个式子用于归一化，任务二中取$\beta=-5$；由于第二个式子的 d 是不对称的，所以计算距离的时候将两个顺序都算一次然后求平均。

总相似度. 考虑几何形态相似度后再算上对路段类型的分类(也就是 highway 属性)，定义总相似度为：

$$\begin{aligned}\mathrm{Sim}_{\mathrm{type}}(A,B)&=[highway_A=highway_B]\\ \mathrm{Sim}(A,B)&=\frac{\mathrm{Sim}_{\mathrm{shp}}(A,B)+\mathrm{Sim}_{\mathrm{type}}(A,B)}{2}\end{aligned}\tag{3}$$

2.3 聚类算法

由于很难给路段定义一个所谓的“平均路段”出来，所以以 KMeans 为代表的基于聚类中心的聚类方法都无法使用。使用 DBSCAN 进行聚类。用 1 减去相似度就是距离。设置 DBSCAN 的邻域的半径为 0.4，认为成为核心点的最小邻域元素数量设置为 500. 也就是说，只有是同一种类型的道路，并且两者之间有一



图表 4 聚类结果

定的几何形态相似度，才会认为在邻域内；而形状相似的路段，即使不是同一种道路类型也可能被放在一起。

由于定义的路段相似度计算起来较复杂（涉及到计算伪逆和最小二乘法等），需要很长时间。计算一个路段相对于其他所有路段的相似度大约需要 7-8 秒，而计算所有路段两两距离，由于只用算上三角，平均每个路段需要计算 3-4 秒，硬算需要 40 小时以上，无法接受。后来发现计算时 CPU 占用率和内存占用都非常低，性能瓶颈不在计算，而很可能在 io 上。计算两两间相似度的时候，互相之间没有任何关系，非常适合并行处理，使用并行手段在 5 小时内计算出路段之间的两两距离。

计算完路段两两间距离后，直接使用 sklearn 的 DBSCAN 算法得到聚类结果。

2.4 聚类结果

DBSCAN 聚类后得到 9 类，其中 1 类是“异常点”，但在此距离的定义下，这个“异常点”类也有其特殊的意义。DBSCAN 聚类结果的可视化见图表 4。主干道和周边小路能看出比较明显的区分。值得讨论的是“异常点”，它有一个特殊意义是形状比较奇怪的路段，图 4 中红色线是 DBSCAN 中的异常点，而这些形状比较奇怪的路一般就是立交桥交汇处的各种引路，如图 4 左上角的图，它们可能拥有相似的性质，比如比较容易发生拥堵等。但是这类路段中还会包含一些其它的形状比较奇怪的路段，需要进一步根据道路类型等进行筛选。

2.5 总结与讨论

自定义了衡量路段间相似度的度量，并实践了 DBSCAN 算法，得到了初步的聚类结果。但是，所定义的度量计算较慢，不便使用，且其未必有很强的实际意义，因为形状相似的路段不意味着其它性质也相似，甚至可能完全没有关系，使得这个度量更像是一种数学上的游戏和尝试。此外，没有深入讨论基于此度量和 DBSCAN 的聚类结果与道路类型分类的关系。

3 任务三：ETA

ETA 的任务是，在已经有一条路径的情况下，预测需要多长时间走完这条路径。在本任务中，只给出了起点和终点的坐标位置和速度等信息，所以本任务使用了两阶段法解决 ETA 问题，即先实现路径规划，再实现速度预测。事实上，本方法也是符合直觉逻辑的，现实世界中，司机也往往是根据导航或者想好的路线进行驾驶，因此该方法是合理的。

本任务的神经网络和训练使用 PyTorch 实现。

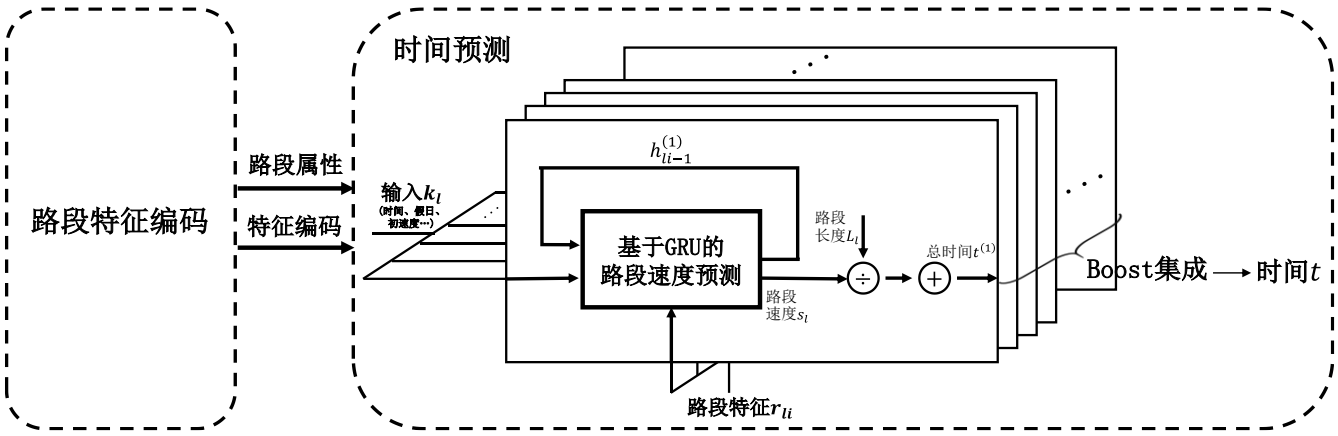
3.1 路径规划

对于路径的规划，我们采用了常见的路径规划，即使用 Dijkstra 算法进行简单的路径规划，我所需要做的只需要把起点匹配到对应路径相应的结点，使用相应算法得到最短路即可。

3.2 ETA 旅行时间预测管线

时间预测管线如图表 5 所示，分为路段特征编码和时间预测两个阶段。在原先的构想中，我们试图用 GAT 作为路段特征编码的学习器，与时间预测一起学习，实现一个完整的端到端的表征学习，但是后来因为一些原因放弃，将在本节的最后简述。最后我们的路段特征编码是人工设计的。时间预测模块则使用基于单层 GRU 的简单循环神经网络，配合 Adaboost 策略训练和使用。

符号定义。路网 **R** 中的每个路段都有其特征编码，路段R_l的特征编码为$\mathbf{r_l}$，为 h 维向量。每个轨迹



图表 5 ETA 管线

同样有一个编码$\mathbf{k}$作为 GRU 的初始状态，同样是 h 维向量。用其经过的路段定义一个轨迹$\mathrm{T}_l = < R_{l1}, R_{l2}, \dots, R_{lm} >$，其出发时间为$t_l^T$，只记录小时数，出发速度为$s_l^T$，是否为假日$holiday_l^T$，相应的编码为$\mathbf{k}_l$。轨迹$\mathrm{T}_l$所经过的路段的路段特征为$< \mathbf{r_{l1}}, \mathbf{r_{l2}}, \dots, \mathbf{r_{lm}} >$，最终预测的速度为$\mathrm{t}(\mathrm{T}_l)$。

3.3 特征编码

我们最后使用的特征编码是人工设计的，包括路段特征编码和用作 GRU 初始状态的轨迹特征编码。和任务二类似，我们只使用路段的位置、类型和路网图上拓扑性质为路段设计特征编码，而对于轨迹则只使用轨迹时间、是否假日、初始速度这些给出的 ETA 任务中可获取的属性进行编码。

路段R_l的特征编码$\mathbf{r_l}$的定义为：

$$\begin{aligned} \mathbf{r_{type}} &= \mathrm{onehot}(highway_l) \\ \mathbf{r_d} &= \mathrm{sinencode}^{(\mathrm{h})}(d_l) \\ \mathbf{r_\rho} &= \mathrm{sinencode}^{(\mathrm{h})}(\rho_l) \\ \mathbf{r_l} &= [\mathbf{r_{type}} || d_l || \mathbf{r_d} || \rho_l || \mathbf{r_\rho} || L_l] \end{aligned} \tag{4}$$

其中，d为路段在路网图中的度（包括入度和出度）。ρ表示路段的密度，密度以如下方式定义：将所给范围的路网分为 35*28 的网格(每个格子对应实际大约 600m*600m 范围)，将每个路段的起点和终点按照坐标放入所属的网格，路段的密度为其起点和终点所处的格子中含有的点数量的平均。$\mathrm{sinencode}^{(\mathrm{h})}$是一个简单的编码方程，定义为：

$$\mathrm{sinencode}^{(\mathrm{h})}(x) = (\sin(2^0 x), \sin(2^1 x), \dots, \sin(2^{h-1} x))$$

任务三中取 h=8，最后路段特征编码的长度为 32.

轨迹$\mathrm{T_l}$的编码$\mathbf{k_l}$定义为：

$$\begin{aligned} \mathbf{k_{hour}} &= \mathrm{triencode}^{(\mathrm{p_1})}(hour_l) \\ \mathbf{k_{holiday}} &= \left(||_{i=1}^{p_2} holiday_l \right) \\ \mathbf{k_s} &= \mathrm{posencode}^{(\mathrm{p_1})}(s_l^T) \\ \mathbf{k_l} &= (\mathbf{k_{hour}} \left|\left| \mathbf{k_{holiday}} \right|\right| s_l^T || \mathbf{k_s}[1:]) \end{aligned} \tag{5}$$

其中，holiday是 0-1 变量，hour 是出发时的小时数。Triencode 和 posencode 分别是 NeRF 中和 Transformer 中使用过的两种位置编码，定义为：

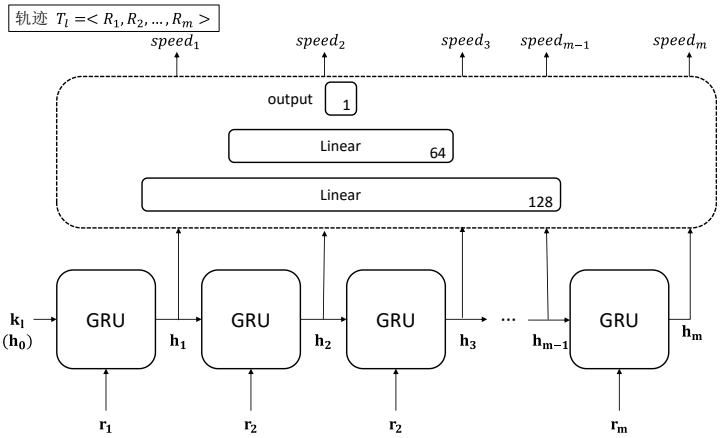
$$\mathrm{triencode}^{(\mathrm{p})}(x) = \left(\sin 2^0 x, \cos 2^0 x, \dots, \sin 2^{\frac{p-2}{2}} x, \cos 2^{\frac{p-2}{2}} x \right)$$

$$\mathrm{posencode}^{(\mathrm{p})}(x) = \left(\sin \frac{x}{10000^{\frac{2i}{p}}}, \cos \frac{x}{10000^{\frac{2i}{p}}} \right), i = 1, \dots, \frac{p-2}{2}$$

由于$\mathbf{k_l}$用作 GRU 的初始状态，它的维度和 GRU 状态的维度相同。本任务设置 GRU 的状态维度为 128，设置$\mathrm{p_1} = 60, \mathrm{p_2} = 8$。

3.4 基于 GRU 的速度路段预测

任务三使用一个简单的单层 GRU 进行轨迹经过的每个路段的平均速度的预测。模块设计如图表 6 所示。使用 GRU 预测轨迹所经过的各个路段的隐状态，然后用一个共享的 2 层 MLP 将其转换为速度，最后得到各个路段上的平均速度。得到平均速度后，用各个路段的长度除以此平均速度得到各个路段的时间，最后加和得到预测的总时间。在实现中，为了进行非负约束，这个模块直接输出的速度会被认为是速度的自然对数$(\ln speed)$。



图表 6 基于 GRU 的路段速度预测模块。注意图中的 GRU 是同一个单层 GRU，将 GRU 状态转换为速度的全连接层神经网络也是共享的，全局只有一个。

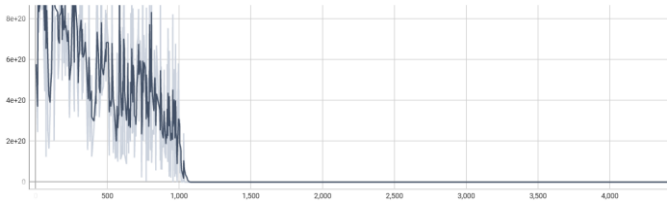
3. 5 损失函数与初步结果

直接使用预测出的时间与实际时间的均方误差作为损失函数，同时加上 L2 正则化项。损失函数为：

$$\mathrm{Loss}\big(label(T_l), t(T_l)\big) = \mathrm{MSE}\big(label(T_l), t(T_l)\big) + \alpha\big|\big|\omega\big|\big|_2 \quad (6)$$

在本实验中，$\alpha = 0.001$。

将匹配成功的一共 16967 个轨迹分为训练集、验证集、测试集，占比分别为 0.4,0.3,0.3，数量分别为 6786、5090，5091.首先尝试直接用单一的上述预测模块进行训练。



图表 7 时间的均方误差

从图 7 可以看出，网络收敛地非常迅速，几乎只用 1000 多次 Minibatch 的训练就能收敛，最后损失值收敛在 100~300 左右。在验证集和测试集上，该网络几乎取得了一样的结果，没有发生过拟合。从测试集上来看，最后得到的结果与真实时间的误差平均在 9.13，误差标准差在 9.12.但若细致考察每个预测结果，我们发现许多轨迹的时间预测结果是 0，训练集、验证集、测试集中都有这种现象。在测试集中，一共 5091 条轨迹，有高达 1098 条轨迹预测出的时间小于 0.1min，这是无法接受的。

我们进而尝试用集成方法加以改进。

3. 5 AdaBoost 集成

我们发现，如果只使用一个上述模块进行预测，虽然平均上来看似乎能得到比较好的结果，但是所预测的结果中时而出现预测的时间值几乎为 0 的情况。可能是时间的值本就不大，即使直接预测为 0 其 loss 值也不会很大。

我们尝试使用集成的方式解决这一问题，得益于网络非常快的收敛速度（一般 1 分钟以内就收敛 ），我们进行了许多集成尝试。首先使用最简单的 Bagging 集成，但随着网络越来越多，类似的现象也没有完全被消除，遂决定放弃 Bagging 集成。

我们继而尝试 Boost 集成。Boost 集成中 Adaboost 最简单也最好实现，但是 Adaboost 针对的是分类问题，即使有找到针对回归问题的 Adaboost 版本，但结果都很差，其回归版本在数学上可能也不如其在分类问题上那样完备。我们尝试做一些转化，来让用于二分类问题的原始 Adaboost 应用在我们的网络上。

首先，我们用集成方法的根本动机是消除那些被预测为 0 的时间结果，而这些 0 之所以能大量出现就是因为时间数值本来就比较小，所以即使直接预测出 0 值也不会产生多大损失值，所以不应直接用原本的损失函数来判断下一轮要更加关注哪个样本。我们使用相对误差，即：

$$\mathrm{RL}\big(label(T_l), t(T_l)\big) = \frac{|label(T_l) - t(T_l)|}{label(T_l)} \quad (7)$$

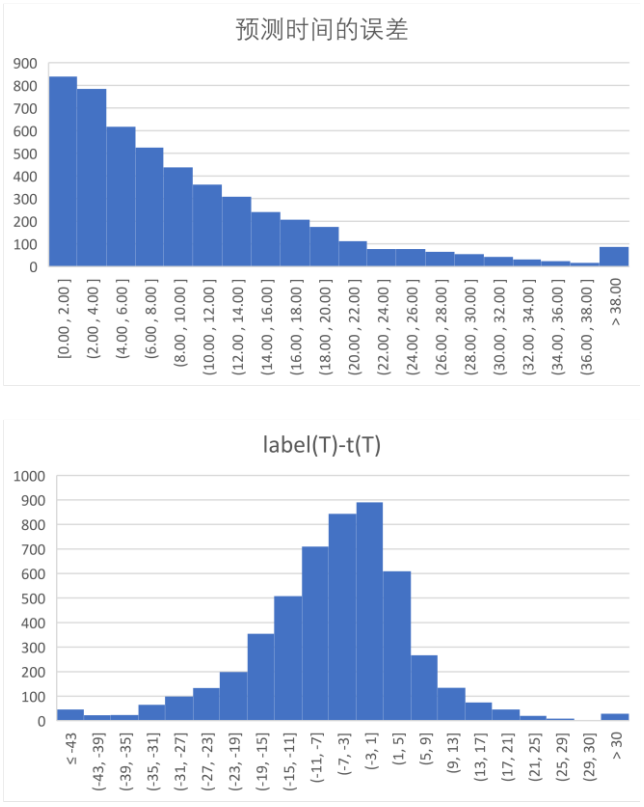
可见，对于那些预测值为 0 的轨迹样本，其相对误差一定是 1.

Adaboost 原始版本是针对二分类问题的，我们对时间回归的结果做一点手脚让它“变成”分类问题。我们设定一个阈值$\tau = 0.7$，样本中相对误差大于τ的被认为是“分类错误”，而小于τ的则认为是“分类正确”。由于τ的取值，那些预测结果为 0 的肯定被包含在分类错误里面。而在训练集 6787 条样本中，属于“分类正确”的样本占比大约 65%，正是一个很弱的“二分类器”。

随后，我们实现了 Adaboost 算法，并用上述分类标准进行迭代，得到我们最终的网络。实际上我们对 Adaboost 的实现是半自动的，每轮迭代何时停止与何时停止迭代是我们人为控制的——因为收敛速度非常快，所以这不构成负担。

3. 6 ETA 结果

采用 AdaBoost 后，我们得到了 9 个前述的网络，以及相应的系数$\alpha_i, i = 1,2,\ldots,9$。同样的，在训练集‘验证集、测试集下得到的结果非常接近，没有产生过拟合。下面以测试集中 5091 条轨迹分析预测结果。



图表 8 测试集上误差直方图

预测结果的平均偏差为 9.80，偏差的标准差为 9.54，误差值的分布直方图见图表 8，形态上来看它大致是个正态分布，而且其中心应当不会离 0 太远。虽然相比只有一个网络的情况其误差的平均值和标准差没有提升，但预测结果中没有任何预测结果小于 0.1 的样本，包括在训练集、验证集中也是如此，并且预测给出的 eta_task 后也未发现这样的极端结果，这说明我们使用的 AdaBoost 方法和相应的修改策略很好地达到了目的：消除那些预测时间为 0 的结果。

但是基于现有网络和人工编码的预测结果很难再往上提升，可学习的特征编码本是深度学习能取得好结果的关键之一，但遗憾我们未能成功运行并接入计划中 GAT 进行路段特征编码的学习。

3.7 总结

我们实现了简单的路径规划，并基于简单的一层 GRU、MLP 和自己人工设计的编码完成了 ETA 任务。最终的参数文件每个不到 300KB，取得了较合理的预测结果。但所使用的方法也有很多缺陷导致结果难以再提升。比如损失函数只对最终的时间做监督，而未对预测出的每个路段上的平均速度做一些约束；只考虑路段上的速度，而没有考虑路口可能的延迟；人工设计的固定特征编码不利于学习；使用的 Adaboost 方法虽然简单易实现但毕竟不是用于回归问题，而且已经很“老”了，有不少更新的方法如 XGBoost 等可以借鉴尝试。

但无论如何，我们对于我们的预测结果仍是较为满意的，任务三对我们来说也是一次宝贵的尝试。

4 任务四：下一跳预测

4.1 问题分析

任务四的下一跳预测本质上就是一个对状态转移预测的分类问题。

首先对已有的数据进行分析，在第一问中给，我们已经得到了轨迹匹配路段的数据。而事实上，一辆车的行进轨迹与其速度往往关系不大，其主要取决于其车主的常见行驶路线和路段的分流情况。因此对于路段的转移，我们使用了 Fast Probabilistic Matrix Completion（FPMC）模型来建模路段的转移。FPMC 模型是一种用于处理序列数据的概率矩阵完成模型，常用于推荐系统和序列建模任务，但是事实上，他也是一个经典的下一跳预测模型。

在这个场景中，我们可以将路段的转移建模为一个概率矩阵，其中每一行表示起始路段，每一列表示目标路段。FPMC 模型通过学习这个矩阵的概率分布，可以捕捉到车辆在不同路段之间的转移模式。具体而言，FPMC 考虑了车辆在当前路段的行为以及其之前的历史行为，从而能够对下一跳路段的可能性进行预测。

模型的输入主要是车辆的历史轨迹数据。通过对这些数据进行分析和特征工程，我们可以构建一个适用于 FPMC 的输入矩阵。然后，模型通过学习这个矩阵的参数，能够在给定当前路段情况下，预测车辆下一跳可能停留的路段。

然而本题的问题并不仅仅是路段转移，我们需要得到更为具体的坐标，因此本题在考虑简单的路段转移的同时需要考虑地理信息即车辆的空间特征，即其所在的坐标。

事实上，这部分处理是比较容易的，在轨迹的最后一个采样点可以得到此时车辆的瞬时速度，同时可以得到下一跳的时间，我们只需要计算在下一跳之前是否能走到预测的状态即可。而具体的坐标，只需要进行简单的代数运算就可以得到。

4.2 FPMC 模型原理

FPMC 模型主要用于推荐系统中预测未知物品引起用户兴趣的可能性并以此排出物品推荐列表。FPMC 的模型灵感主要来源于 MC（马尔可夫链模型）和 MF（矩阵分解）/LFM（隐性因子模型）。

将用户购物这个事件视作马尔可夫过程。例如：用户在已经买了面包的情况下买酱汁的概率会很高。这种是所有用户的共性。用户 a 买了面包后经常买番茄酱，而用户 b 买了面包后经常买沙拉酱。这是不同用户的个性。FPMC 模型能够兼顾两者，使得其能够根据当前用户已经购进的物品预测他下一步会买什么。

引入以下符号：

$$U = \{u_1, ..., u_{|U|}\}$$

$$I = \{i_1, ..., i_{|I|}\}$$

$$B \in I$$

$$\beta^u = (B_1^u, ..., B_{t_u - 1}^u)$$

$$\beta = \{\beta^{u_1}, \beta^{u_2}, ..., \beta^{u_{|U|}}\}$$

其中，U 是用户集，I 是项目集，B_t^u 表示某用户 u 在 t 时刻的购物篮，β^u 是用户 u 的购买历史，β 是全体用户的购买历史。

考虑所有用户的共性，即对全体用户，计算在 t-1 时刻篮子里存在物品 l 时，t 时刻篮子里存在物品 i 的转移概率。

$$a_{l,i} = p(i \in B_t \mid l \in B_{t-1}) = \frac{p(i \in B_t \wedge l \in B_{t-1})}{p(l \in B_{t-1})}$$

此时采用最大似然估计，则有

$$\hat{a}_{l,i} = \frac{\hat{p}(i \in B_t \wedge l \in B_{t-1})}{\hat{p}(l \in B_{t-1})} = \frac{|\{(B_t, B_{t-1}) : i \in B_t \wedge l \in B_{t-1}\}|}{|\{(B_t, B_{t-1}) : l \in B_{t-1}\}|}$$

同时，考虑用户的个性，即计算在 t-1 时刻用户 u 篮子里有物品 I 时，t 时刻该用户篮子里存在物品 i 的概率。同样有：

$$\hat{a}_{u,l,i} = \frac{\hat{p}(i \in B_t^u \wedge l \in B_{t-1}^u)}{\hat{p}(l \in B_{t-1}^u)} = \frac{|\{(B_t^u, B_{t-1}^u) : i \in B_t^u \wedge l \in B_{t-1}^u\}|}{|\{(B_t^u, B_{t-1}^u) : l \in B_{t-1}^u\}|}$$

从而得到个性化三维转移矩阵。

至此，马尔可夫模型构建完毕。为了解决转移矩阵的稀疏问题，并打破参数之间和估计时的独立性，考虑相似用户、项目，转移情况之间的相互影响，我们引入了矩阵分解模型。我们对转移矩阵进行张量分解，最终得到 FPMC 模型：

$$\hat{p}(i \in B_t^u \mid B_{t-1}^u) = \langle v_u^{U,I}, v_i^{I,U} \rangle + \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} (\langle v_i^{I,L}, v_l^{L,I} \rangle + \langle v_u^{U,L}, v_l^{L,U} \rangle)$$

在本题中，用户就是车辆，物品就是路段。比如路段 1 可以通往路段 2、3、4，则 1 转移到 2，3，4 的概率较大而往其它状态转移则为 0。这是共性。而车辆 253 往往在开到路段 1 上后转移到路段 2 上，车辆 254 往往在开到路段 1 上后转移到路段 3。这就是个性。FPMC 模型兼顾共性和个性，是轨迹下一跳预测的常用模型。

4. 3 FPMC 设计

在对 FPMC 模型有了基本的了解之后，我们就可以进行代码的具体撰写了。

4. 4 模型初始化

我们的模型通过正态分布的随机初始化，为模型的参数赋予一个初始状态，包括用户和项目的潜在因子矩阵。这为后续的训练提供了基础，确保模型能够从数据中学到有效的表示，在本题中，每条路段可以到达的新路段显然是有限的，我引入了潜在状态转移矩阵用来限制状态的转移。

4.5 兴趣点数值设计

为了考虑静态特性，使用用户嵌入向量（VUI[u]）表示用户的静态特性，其中包含了关于用户的静态信息，如用户的偏好和兴趣。

为了动态特性的考虑，引入状态转移相关项(acc_val)，该项考虑了用户行为轨迹中之前的项目状态对当前项目兴趣的影响。通过考虑允许的状态转移列表，限制了状态的转移范围，更符合用户行为的动态特性。

为了综合静态与动态，最终的 x 值是用户嵌入向量和状态转移相关项的叠加，使得用户对项目的兴趣分数既包含了用户的静态特性，又考虑了用户行为轨迹中的动态变化。

对于每个之前的项目状态 l，如果 l 在允许的状态转移列表中（self.allowed_trans[i]），计算项目 i 和状态 l 之间的兴趣分数并累加到 acc_val 上：

$$\text{accval} = \sum_{l \in b_{tm1}} (dot\_product(VIL[i], VLI[l]))$$

最终用户对项目的兴趣分数为用户嵌入向量和状态转移相关项的叠加，考虑了动态特性

$$\text{final\_score} = z_1 + \frac{accval}{len(b_{tm1})}$$

通过这样的设计和数学原理，我们能够灵活地结合用户的静态和动态特性，为模型提供更全面的用户行为建模能力。

4.6 用户项目兴趣计算

我们的用户项目兴趣计算与上述类似，首先是用户的嵌入向量。

$$\text{former} = \text{VUI[u]}$$

然后我们通过对可达状态的限制得到

$$\text{latter} = \frac{1}{|allowed\_b\_tm1|} \sum_{l \in allowed\_b\_tm1} dot\_product(VIL\_m\_VLI[:, l])$$

将两个向量相加既可以得到此时对每个项目的兴趣系数。

4.7 其他常见方法

其他方法的设计我们参考了 github 上 FPMC 经典的实现 https://github.com/khesui/FPMC，事实上，我们将几乎所有的重要方法都几乎重写。

对 learn_epoch 方法，我对负样本的选择逻辑进行了修改，以更符合任务要求。在原始 FPMC 模型中，负样本的选择逻辑并没有特别考虑到下一跳预测的任务特性。我的修改是为了确保从最新状态的可达状态中选择负样本，以更好地反映用户行为轨迹的动态特性。在训练中获得更好的效果。

在原始实现中，负样本的选择是从整个项目集合中随机选择的。

我的修改中，通过检查用户最新状态允许的状态转移，仅从这个允许的状态转移集合中选择负样本。这使得负样本更加符合用户当前的行为模式，更好地适应了下一跳预测任务的需求。

通过这个修改，模型在训练时更加关注用户当前状态的动态特性，更好地捕捉了用户行为的演变。这

有助于提高模型对用户行为轨迹的学习效果，从而提升下一跳预测的性能。

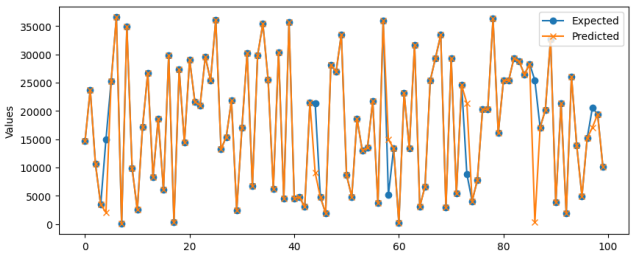
同时我对模型的 evaluate 函数也进行了修改，在机器学习的过程中，我们往往是通过随机数种子对矩阵进行初值赋值，但是在本题中，由于出现了过多的无关状态，我将评估函数的范围缩小在了其可转移状态，对模型的性能产生了巨大的提升。

对于用户实体的矩阵转移，我们不需要做修改。

4.8 实验结果

对于路段转移模型，我们在进行了一系列超参数调整之后，在每个路段平均有 3-4 个可转移状态的情况下，我们对路段转移的预测提升到 60%左右。

在一些数据较少的路段预测概率相对较低，但是在高频次路段的预测准确率达到了 80%



图表 9 部分测试集拟合效果

4.9 坐标转换

对任意一条路径，设 road_id$_1$ 为最后一条经过的路段 id，road_id$_2$ 为由 FPMC 预测的下一条路段 id。c_1，speeds$_1$ 分别为最后观测到的坐标和速度，speeds$_2$ 为预测位置的观测速度。Δt 为预测时间差。

我们可以根据 rel.csv 中的数据找到路段 road_id$_1$ 和路段 road_id$_2$ 的交点 cross.利用 road.csv 中的路段信息，不难根据 road_id$_1$,c_1 得出此时车辆在路段 1 上的剩余距离 remain_dis。

计算 $t_0 = \frac{remain\_dis}{speeds_1}$ 。

若 Δt>t$_0$,则说明车辆能够通过 road_id$_1$ 而到达 road_id$_2$。车辆在路段 road_id$_2$ 上能够行驶的距离的估计值为 $d_2 = speeds_2 * (\Delta t - t_0)$.根据 road.csv 中的路段信息，不难根据 road_id$_2$,交点坐标 cross 和距离交点的距离 d_2 计算出预测时刻的车辆坐标 c_2。

若 Δt<t$_0$，则说明车辆在预测时间内无法转移到我们所预测的路段 road_id$_2$ 上。此时预测车辆在 road_id$_1$ 上沿原方向继续行驶 $d_1 = speeds_1 * \Delta t$.根据 d_1，c_1,不难计算出预测时刻的车辆坐标 c_2.

这里我们都使用了在路段上的观测速度 speeds$_1$,speeds$_2$ 来进行距离计算。我们原本也采用了使用 eta 模型得到的每个路段的速度平均值作为速度估计值来进行距离计算，但考虑到本身用观测到的数据进行预测比用预测的数据预测更加合理，所以最终采用前者。

4.10 设计难点

本题的第一个问题在于如何将问题转化为一个分类问题，如果将本题视为一个回归问题的话，其最终着眼的元素必然是在于坐标，然而坐标所包含的特征过少，于是我们决定采用两阶段法，先进行路段的预测，再进行坐标的具体计算，此时可以有效利用前几问的结果和路段的拓扑特征进行合理的预测，提高结果的准确性。

本题的第二个难点在设计模型的过程，FPMC 算不上什么太高深的模型，事实上他是一个非常经典的模型，但是在设计的过程中仍然出现了较大的困难，由于 FPMC 往往是对所有状态同时进行计算，同时由于随机数种子的存在，导致一些无关状态的噪声非常大。我的解决方案是将几乎所有计算参数相关的方法进行重写，引入潜在转移矩阵，大大降低了无关状态的噪声，使得模型取得了较好的拟合效果。

第三个问题在于模型的性能问题，FPMC 模型的 evaluate 函数速度极慢，我主要采用了两种思路进行优化：1.使用 numba 库将代码直接转化为机器码加速，但是事实上，其对训练的速度有明显的提升，但是对于评估部分兼容性不佳，而且效果并不明显。2.重写 evaluate 方法，提高检索效率并且使用 numpy 库对计算向量化，同时采用了并行优化，在这些优化的帮助下，我们的模型的运行时间由十余小时一次提升到了五分钟以内。

5 总结

由于时间问题，我们的作业并没有做到尽善尽美写下这篇报告的时候已经是烤漆了，作业也不太可能再继续完善了，总的来说，本次作业是充满遗憾但有着很大的收获，无论是哪一问我们都仍然有着很大的进步空间，第一问 fmm 有着一些奇怪的路径匹配，第三问我们也没有很好的把 GAT 实现导致第三问只能直接训练人工设计的编码，并不利于学习，第四问对地理信息的处理过于粗糙，FPMC 在给定的轨迹下对于某些情况的预测过于随机，某些轨迹并没有达到某些路段，在这种情况下可能无法得到一个很好的结果。对于深度学习编程的经验还是过于青涩，设计模型时对某些特征的考虑也不够全面，做了很多无用功。

但是这次作业总的来说也是收获很大，我们团队的协作能力也达到了提升，希望本次作业作为在数据挖掘上的初体验可以为之后的学习科研工作生涯开一个好头。

参 考 文 献

[1] https://github.com/cyang-kth/fmm

[2] https://github.com/khesui/FPMC

[3] Yang C, Gidofalvi G. Fast map matching, an algorithm integrating hidden Markov model with precomputation[J]. International Journal of Geographical Information Science, 2018, 32(3): 547-570.