上一周的实验情况

日期：July 13, 2020

目录

上一周按照王老师的要求重新进行了实验的验证，这次是一点点从微小的地方开始的验证。下面是进行的实验：

验证对象	验证方法	验证结论
基于神经网络的匹配函数	单独对基于神经网络的匹配函数进行匹配的精度测试，并将之与传统的相似度进行比较	正常
search on graph 模块	人为设计了一些节点，之后将 inner product，l2，cosine 等度量进行了测试	正常
hnsw+NN	使用 MLP 作为度量函数进行实验，在数据集上进行实验	召回率接近 0
NN+ 遍历搜索	同上	召回率接近 0

结论：问题出在数据集上。如果将数据集中所有的 item 看作是被搜索的节点，然后使用这些 item 进行 graph 的构建，那么当输入一个 user 时，并不一定可以找到与之对应的 item.(概率很低.) 我在 60400 个 item 上进行了实验, 发现无论是训练好的神经网络还是传统相似度都很难对特定的 user 输入找到特定的 item. 因此, 排查完模型没有问题之后, 我将注意力集中在数据集上.

数据集的格式是:

user-序号	item-序号	rating(也就是星级, 打分)
0	1	5
0	100	4
1	3	5
...	...	...

我参考了 Fast Item Ranking 那篇论文所引用的之前的工作对数据集的处理, 将数据集的处理设定为如下步骤:

1. 信息整合. 假设有 X 个 user, Y 个 item. 那么对该数据集的信息进行读取其实就是构造一个 X*Y 维的矩阵 M. 对于矩阵 M, 如果 user x 与 item y 出现在了数据集中 (也就是上面的表格里), 那么 M[x,y]=1(无论在数据集 rating 的评分是多少), 如果没有出现, 就记作 0. 这样就可以生成一个 0-1 的关系矩阵 M.
2. 训练, 生成嵌入式向量. 这一步的想法是通过前面的信息, 生成每个 user, 每个 item 的 embedding. pytorch 有一个 Embedding 函数, 可以将一个输入的索引 (序号, 也可以理解为 one-hot) 转化成一个 d 维的 embedding. 之后, 将 user 与 item 的 embedding 输入到匹配函数里 (经典的一些相似度, 或者 MLP), 就可以输出一个结果. 对于这个结果, 损失函数的设计应遵循: 如果输入的 user 与 item 在矩阵 M 中对应的数值为 1, 则希望结果越大越好. 否则, 则希望越小越好. 通过这个函数, 反向训练 MLP 和 embedding 的生成.

当我对这个流程了解了之后, 我的思索是: 为什么一个 user 对它没有进行评分的 item 有如此高的 matching score(或者说: 如此小的距离).

我觉得解决这个问题肯定要从损失函数入手. 也就是添加大量的无关无关样本结果是 0 的标签. 目前的工作就在这里进行.