实验二 豆瓣电影的知识感知推荐

小组成员：PB22000257 阎昶澍 PB22061323 汪若贤 PB22071456 储溦

本次实验的提交文件夹组织如下，主目录位于`PB22071456-储溦-实验二`下：

```
D:.
|   实验报告.pdf //本次实验二的实验报告
├─KG_distill //抽取知识图谱子图的源码
|       match_triple.py
|       toindex.py
└─Rec //这是感知推荐的源码，在框架上进行补充得到
    |   .DS_Store
    |   main_Embedding_based.py
    |   main_KG_free.py
    ├─data
    |   └─Douban
    |           kg_final.txt  //KG_distill得到的图谱文件
    |           test.txt
    |           train.txt
    ├─data_loader
    |   |   loader_base.py
    |   |   loader_Embedding_based.py //有修改
    |   |   loader_KG_free.py
    |   └─__pycache__
    ├─model
    |   |   Embedding_based.py //有修改
    |   |   KG_free.py
    |   └─__pycache__
    ├─parser
```

```
|  |  parser_Embedding_based.py //有修改
|  |  parser_KG_free.py
|  └─__pycache__
├─trained_model //不同超参数配置训练得到的不同模型
└─utils
```

一、基于 Freebase 的知识图谱抽取

实验目标

首先从 `Freebase` 中抽取出与 `578` 部电影有关的三元组，进而抽取出多跳三元组来构建包含这些电影实体的小规模图谱，以获取电影实体间的丰富信息，从而辅助知识感知推荐系统的建模。

实验方案

- 先提取所有可匹配的电影 `id` 列表备用。
- 对所给数据文件，由于大小过大，单纯遍历的方式效率低下，故采用分块的方法，采用 `cpu` 并行处理多个数据块，提高效率。
- 同时在运行过程中阶段性保存已处理的数据并清除缓存，避免内存溢出。采用 `tqdm` 库可视化进度，保证程序运行时间在可接受范围。

获得所有匹配的三元组后，利用 `movie_id_map.txt` 将其映射到索引值。

代码文件 `match_triple.py`

- 电影 `id` 处理：函数 `load_mapfreebase_id()`

  将 `douban2fb.txt` 中的 `freebase id` 加入可匹配 `fb_ids` 列表，方便后续进行三元组的匹配。

- 数据块处理：`process_chunk()`

  对每个数据块，提取其中所有三元组，去除无用的字符。

  匹配前缀，提取头尾元素其中的有效 `id`。

  对于所有可匹配的id，将其对应的三元组加入匹配三元组列表。

- 数据的块划分和并行处理：`extract_matched_entities()`

  将 `fb_ids` 载入 `matched_fb_entities`。

  确定 `cpu` 核心数，建立相应数量的进程，确定每块大小。

  每次读取多个内存块，并行处理，同时更新进度条，定期保存结果，并清理内存。

代码文件 `toindex.py`

- 电影映射关系建立：`load_movie_id_map()`

  将 `movie_id_map.txt` 中的索引对照与 `fb_id` 相对应，存入 `movie_id_map` 字典，建立映射关系。

- 加载图谱数据：`load_graph_data()`

  将上一步处理的三元组载入。

- 将图谱映射为索引：`map_entities_and_relations()`

先将所有的电影`id`映射为相应索引值。

再在电影索引值基础上，完成所有实体`id`的映射。

最后再完成所有关系的映射。

- 根据先前建立的索引映射，处理三元组：`convert_to_indexed_triples()`

实验结果

- 处理得到一跳索引三元组文件：`kg_final.txt`
- 其中共匹配了`240073`个相关三元组。
- 数据处理时间为`05:20`

二、基于知识感知的推荐系统

1. 实现内容

1. 基于框架完成了三种物品对应实体的知识嵌入方式，分别是**逐元素相加**，**逐元素相乘**，**kg-embed 与 ID-based embed 的拼接。**
    - 为了实现三种方式的自由切换，我们在`parser_Embedding_base.py`中加入了一个额外命令行参数`--inject_manner`，用来指定知识的注入方式，具体如下：

```
## 增加的一个命令行参数，指定知识嵌入方式
 # 0 for concat
 # 1 for add
 # 2 for multi
 # default is 0
 parser.add_argument('--inject_manner', type=int, default=0,
                     help='Manner of injecting the KG to
item_emb.')
```

    - 对于直接相加和相乘，我们只需将从`item_embed`和`entity_embed`中得到的同一个电影的`embed`直接相加和相乘即可，然后将结果与`user_embed`进行内积得到评分。
    - 而对于拼接方法，由于拼接后的特征维度为`2 * embedding_dim`所以无法直接与用户特征做内积，所以我们加入了一个线性层`nn.Linear`来进行维度的变化，将拼接后的特征映射到原来的维度，从而进行线性内积产生评分。
2. 实现了两种图谱表示学习算法，分别是**TransE**和**TransR**。
3. 将知识感知推荐结果与基础`MF`进行了比较。

2. 实验结果

基于上述实现内容，我们做了扩展性的比较实验和消融实验来说明知识感知推荐的优势以及其与普通`MF`推荐之间的区别。具体而言：我们会通过详细的实验结果分析来回答下面三个问题：

- **（RQ1）哪种知识嵌入方式的知识感知推荐效果最好?**
- **（RQ2）哪种图谱表示学习方法更好?**
- **（RQ3）我们的知识感知推荐是否能够优于普通的MF算法?**

2.1 实验设置

2.1.1 数据集和评价指标

我们采用`BPR loss`来作为优化目标，以此来加大正样本和负样本的评分差距，从而完成对电影进行个性化排序和推荐的任务。数据集在工作目录`./data`下，其中`train.txt`是训练集，`test.txt`是测试集，数据集格式为每一行是一个用户`idx + 他的正例电影样本idx`，同时还有我们提取的知识图谱子图`kg_final.txt`，其中的索引与前面的一致。我们选取`Recall@5 NDCG@5`作为评价指标，因为我们经过试验发现`Recall@10 NDCG@10`的结果与`top-5`的推荐结果具有一致性，所以使用`@5`的指标结果不失普遍性。并且由于我们的模型参数均采取`Xavier_init`，所以对于每一个超参数配置，其运行结果应为相同，我们不需多次实验取平均值，一次实验即可得出可信的结果。

2.1.2 超参数设置实现细节

对于所有实例，我们设置学习率为**0.001**，`L2loss`的正则化系数为**0.0001**保持不变，并且采用早停机制，收集每次的最佳指标结果。并且我们注意到对于`MF-based`推荐算法来说，用户和物品的嵌入表示的维度`embedding_dim`是一个重要的超参数，太小可能不足以表达特征，太大又可能造成过拟合现象，所以我们在**24-52**的范围内以每次增加**4**来改变`emdedding_dim`的值，从而寻找不同嵌入大小对应的实验结果背后隐藏的规律。值得一提的是，对于`TransE`算法，由于其没有将实体嵌入通过关系投影矩阵投影到关系的嵌入空间，所以必须保证关系的嵌入维度`relation_dim`与物品和实体的嵌入维度一致，所以在变化`emdedding_dim`时，需保证`relation_dim`跟随其变化，同样对于`TransR`算法而言，虽然两种维度可以不同，但是为了方便与`TransE`算法进行比较，我们也将其设为一致，并且这种设置是不失普遍性的。

2.2 不同知识嵌入方式对于推荐结果的影响(RQ1)

我们对三种知识嵌入方法（**逐元素相乘、逐元素相加、拼接**）在两种图谱表征学习算法上均做了实验，我们将结果汇总为图表，最终结果如**Figure 1 2 3 and 4**所示：

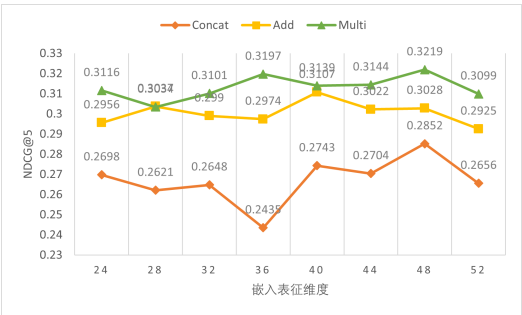


Figure 1 : NDCG@5 w.r.t. inject_manner in TransE

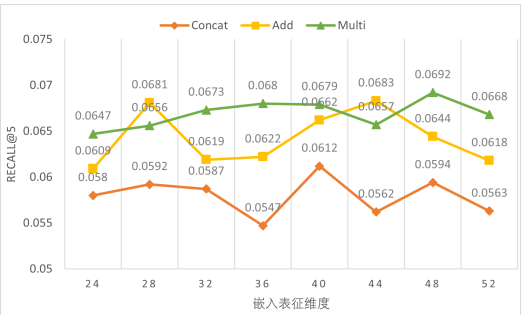
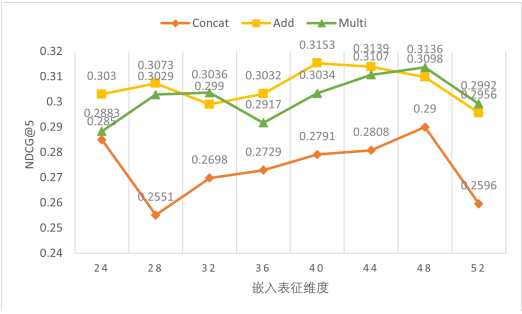


Figure 2 : Recall@5 w.r.t. inject_manner in TransE



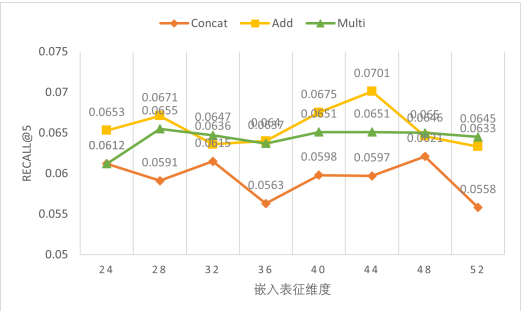


Figure 3 : NDCG@5 w.r.t. inject_manner in TransR

Figure 4 : Recall@5 w.r.t. inject_manner in TransR

从图中我们可以总结到以下几点结论：

- 推荐精度的差异：

1. 无论对于`TransE`还是`TransR`算法，`Concat`组（**Red line**）的`NDCG@5`和`Recall@5`指标较于其他两种嵌入方式均效果最差，说明`Concat + Linear Transform`嵌入方式可能不适于此场景下的知识感知推荐。
2. 针对`Add`和`Multi`方式来说，可以发现在`TransE`算法中，`Multi`（**Green line**）在大部分情况下的表现要高于`Add`（**Yellow line**）的嵌入方式，特别是在`embed_dim>=32`时，说明`Multi`方式适合配合`TransE`算法使用，而在`TransR`算法中，情况刚好相反，即`Add`方式更适合配合`TransR`使用。
3. 在`embed_dim`变大时，`Multi`方式更倾向于超过`Add`方式，说明在嵌入维度较大时，使用`Multi`会更好。

- 嵌入表征维度的影响：
  随着嵌入表征维度的增加，`Recall@5`分数通常会有所提升，但提升幅度会逐渐减小。且当`embed_dim`增加到`48`左右时，再往后增加时两种指标均出现了明显下降（**所有实例均如此**），我们有理由怀疑是出现了过拟合现象，因为随着嵌入维度增加，模型复杂度也会增加。
- 模型稳定性：
  从图表中的数据点分布来看，`TransE`和`TransR`模型在不同注入方式和嵌入表征维度下的`Recall@5`分数相对稳定，没有出现极端波动的情况。这表明这两个模型都具有一定的鲁棒性和稳定性，能够在不同的实验设置下保持相对一致的性能表现。
- 数据指标的差异：
  对比`NDCG@5`和`Recall@5`这两个数据指标，我们可以发现它们在不同情况下的表现存在差异。`NDCG@5`更注重排名靠前的结果的准确性，因此更适合用于评估模型的排序性能；而`Recall@5`则关注于能否正确召回相关结果，因此更适合用于评估模型的记忆能力。

2.3 不同图谱表示学习算法的比较（RQ2）

同时在`2.2`的基础上，我们进一步横向分析了`TransE`和`TransR`的差异，结果总结在**Table 1 and 2**中（**加粗的数据点为表现更好的实例**）：

Table 1 : NDCG@5 w.r.t.Trans(E/R)

	Embedding_dim							
	24	28	32	36	40	44	48	52
Concat								
TransE	0.2698	**0.2621**	0.2648	0.2435	0.2743	0.2704	0.2852	**0.2656**
TransR	**0.2850**	0.2551	**0.2698**	**0.2729**	**0.2791**	**0.2808**	**0.2900**	0.2596
Add								
TransE	0.2956	0.3037	0.2990	0.2974	0.3107	0.3022	0.3028	0.2925
TransR	**0.3030**	**0.3073**	**0.2990**	**0.3032**	**0.3153**	**0.3139**	**0.3098**	**0.2956**
Multi								
TransE	**0.3116**	**0.3034**	**0.3101**	**0.3197**	**0.3139**	**0.3144**	**0.3219**	**0.3099**
TransR	0.2883	0.3029	0.3036	0.2917	0.3034	0.3107	0.3136	0.2992

Table 2 : Recall@5 w.r.t.Trans(E/R)

	Embedding_dim							
	24	28	32	36	40	44	48	52
Concat								
TransE	0.0580	**0.0592**	0.0587	0.0547	**0.0612**	0.0562	0.0594	**0.0563**
TransR	**0.0612**	0.0591	**0.0615**	**0.0563**	0.0598	**0.0597**	**0.0621**	0.0558
Add								
TransE	0.0609	**0.0681**	0.0619	0.0622	0.0662	0.0683	0.0644	0.0618
TransR	**0.0653**	0.0671	**0.0636**	**0.0640**	**0.0675**	**0.0701**	**0.0646**	**0.0633**
Multi								
TransE	**0.0647**	**0.0656**	**0.0673**	**0.0680**	**0.0679**	**0.0657**	**0.0692**	**0.0668**
TransR	0.0612	0.0655	0.0647	0.0637	0.0651	0.0651	0.0650	0.0645

从表中我们可以得到以下几点结果：

- 使用`Add`嵌入方式时，使用`TransR`算法基本一直效果更好；而使用`Multi`方式时，`TransE`算法更好，这与我们在**2.2节**的讨论是一致的。
- 在使用`Concat`嵌入方式时，基本上是`TransR`图谱学习算法效果更好，说明将实体特征映射到关系空间再进行比较是有必要的。

2.4 知识感知的推荐与普通MF的结果比较（RQ3）

我们选取`MF`作为`baseline`，并且将经过**2.2 2.3**的分析得到的最坏和最好的知识感知推荐模型与其做了比较，这里我们选取`Multi + TransE`和`Concat + TransE`分别作为最好和最坏的知识感知推荐模型，得到结果如**Figure 5 and 6**所示：

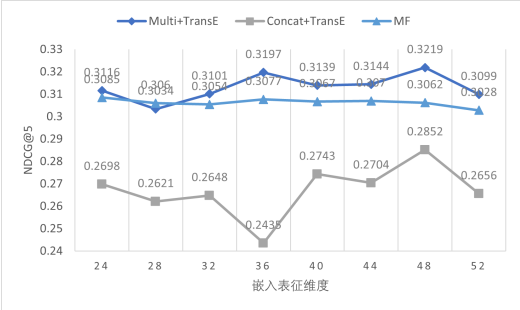


Figure 5 : NDCG@5 w.r.t. Rec Model

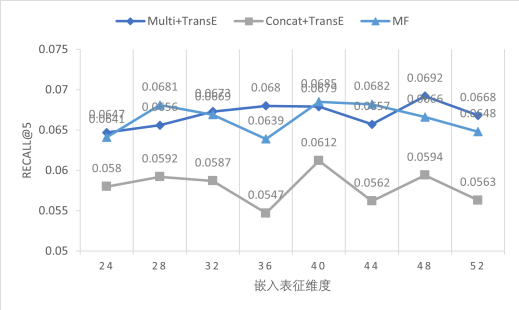


Figure 6 : Recall@5 w.r.t. Rec Model

我们可以得到以下结果：

- 最坏的知识嵌入推荐模型无论是在`NDCG@5`还是`Recall@5`上较于`MF`一直较差，但是随着`embed_size`增大，差距逐渐缩小。强烈说明了选好知识嵌入和图谱表征学习算法的重要性。
- 最好的知识嵌入推荐模型较于`MF`在`embed_dim`较小时表现相当，但是当嵌入表征维度增大时，我们的`Multi + TransE Model`逐渐超过了`MF`，说明我们的模型是有效的。
- 从模型泛化性角度看，`MF`在`embed_size`为`40`左右时精度就开始持续下降，出现过拟合现象，而我们的知识感知推荐模型，一直保持着一个不错的稳定数值，说明我们的改进模型泛化性更好。

3. 实验总结与未来改进

1. 通过这次实验，我学习到了使用知识图谱完成知识感知的可理解的推荐模型的方法，并进行了横向和纵向的对比实验分析，得出了在选择适当嵌入方式和图谱学习方式的情况下，知识感知推荐模型可以表现出比`MF`更好的推荐效果
2. 未来，我们将继续研究知识感知推荐的其他方法，通过知识图谱的丰富语义知识，使得推荐精度得到提升，具体而言：例如，我们会考虑在`Concat + Transform`之后，对结果进行激活，如使用`nn.ReLU()`，来更好的模拟信息处理情况。