红楼梦知识图谱

组　　长: 162150221 包文鑫
组　　员: 162150205 冯敏
班　　级: 1621502
指导老师: 戴洪良

目录

1 简介

《红楼梦》是中国古代文学的经典之作，其丰富的人物关系和复杂的情节构成了一个庞大的文学世界。为了解决传统阅读方式无法充分展示和分析这些复杂关系的问题，我们借助现代信息技术，提出了构建《红楼梦》知识图谱的项目。知识图谱不仅是一种可视化的展示手段，更是一种深层次的信息整理和关系推理工具。通过提取书中人物的关系数据，我们旨在通过现代信息技术手段，深度挖掘和整理《红楼梦》这部经典文学作品中的各种人物、事件、地点等信息，构建一个能够有效描述和推理《红楼梦》中各类人物之间关系的结构化图谱。本项目的主要任务包括数据的获取与预处理、实体和关系的抽取、数据的存储与管理，以及基于向量表示的关系推理与评估。我们希望通过这些工作，能够为《红楼梦》的研究者和爱好者提供一种全新的、更加深入的探索方式。

2 知识抽取

2.1 爬虫

我们选择《红楼梦人物列表》维基百科网站中的表格数据文本作为数据爬取的目标。链接如下：红楼梦人物列表

我们提取的到表格中的姓名数据作为实体，提取得到表格中的人物简介，并对半结构化数据进行分析，从中抽取出人物关系三元组。最终我们可以得到包含两个实体一个关系的三元组格式，作为后续深度关系抽取技术结果的补充和比对。

```
    def append_excel(data, excelname, sheetname, insert_type):
    """
    将DataFrame数据追加到Excel文件中指定的表。
    """
        if not os.path.exists(excelname):
            print(f"文件 {excelname} 不存在，创建新文件。")
            # 如果文件不存在，创建一个新的工作簿
            wb = openpyxl.Workbook()
            wb.save(excelname)

        try:
            # 加载现有工作簿
            book = openpyxl.load_workbook(excelname)

            if sheetname not in book.sheetnames:
```

```
            print(f"工作表 {sheetname} 不存在，创建新的工作表。")
            # 如果指定的工作表不存在，创建一个新的工作表
            book.create_sheet(title=sheetname)

        sheet = book[sheetname]

        # 获取原数据的行数
        original_row = sheet.max_row

        if insert_type == 'w': #
            选择写入excel数据方式，w为覆盖模式，a+为追加模式
            startrow = 0 # 覆盖模式从头开始写入
        elif insert_type == 'a+':
            startrow = original_row # 追加模式从原数据行数之后开始写入

        # 如果是覆盖模式，需要清空原有数据
        if insert_type == 'w':
            for row in sheet.iter_rows():
                for cell in row:
                    cell.value = None

        # 将DataFrame数据写入到指定的起始行
        for row_idx, row_data in enumerate(data.values, start=startrow +
            1):
            for col_idx, cell_value in enumerate(row_data, start=1):
                sheet.cell(row=row_idx, column=col_idx,
                    value=cell_value)

        # 保存工作簿
        book.save(excelname)
        book.close()
        print(f"数据成功写入 {excelname} 的 {sheetname} 表中。")
    except Exception as e:
        print(f"Error writing to {excelname}: {e}")
```

实现的效果如下表1(我们截取了部分表格)

姓名	人物简介	首次出场回目	备注
贾演	宁国公，贾源胞兄	第二回	也作“贾寅”
贾源	荣国公，贾宝玉的曾祖父	第二回	也作“贾法”
贾代化	京营节度使世袭一等神威将军。贾演长子。	第二回	
贾代善	贾源长子，贾宝玉的祖父	第二回	
贾代儒	贾家塾中司塾，贾瑞的祖父	第八回	
贾代修	贾府“代”字辈宗族远亲	第十三回	
贾敷	贾代化长子，贾敬之兄。早夭	第二回	

表 1: raw data

2.2 实体抽取

接着将获取到的表格中的“姓名”列当作实体数据，保存到 entities.txt 文档中。代码如下：

```
import pandas as pd

# 读取 Excel 文件中的数据
df = pd.read_excel('./raw_data.xlsx', sheet_name='Sheet1')

# 提取实体
all_entities = []
for i in range(len(df)):
    # 获取第i行，第0列的数据
    value = df.iloc[i, 0]
    if pd.notna(value):  # 检查是否不是NaN
        all_entities.append(str(value))  # 将所有值转换为字符串并追加到列表

# 将实体写入到文本文件中
with open('entities.txt', 'w', encoding='utf-8') as f:
    for entity in all_entities:
        f.write(entity.strip() + '\n')
```

2.3 关系抽取

接着对提取得到的表格数据中的“人物简介”列的半结构化数据进行分析，从中抽取得到包含两实体一关系的人物关系三元组数据，保存到 *raw_relations.txt* 文档。代码如下：

```
# read excel
import pandas as pd
df = pd.read_excel('./raw_data.xlsx', sheet_name='Sheet1')
```

```
# extract relations
all_relations = []
for i in range(len(df)):
    line = df.iloc[i, 1]
    # line是float类型
    if isinstance(line, float):
        continue
    if line == "贾府远房宗族":
        relation = df.iloc[i, 0]
        relation += ",贾府,远房宗族"
        all_relations.append(relation)
    # split by ', ' and '。'
    line = line.split(', ')
    for t in line:
        t = t.split('。')
        for s in t:
            relation = df.iloc[i, 0]
            if '的' in s:
                # s前面的部分, s后面的部分
                s = s.split('的')
                for r in s:
                    relation+=","+r
            elif '之' in s:
                s = s.split('之')
                for r in s:
                    relation+=","+r
            elif '长子' in s:
                relation += ","+s[:-2]+",长子"
            elif '次子' in s:
                relation += ","+s[:-2]+",次子"
            else:
                continue
            all_relations.append(relation)
with open('raw_relations.txt', 'w', encoding='utf-8') as f:
    for t in all_relations:
        f.write("".join(t).strip() + '\n')
```

实现的效果如下表2:

head	relation	tail
贾宝玉	曾祖父	贾源
贾演	长子	贾代化
贾源	长子	贾代善
贾宝玉	祖父	贾代善
贾瑞	祖父	贾代儒
贾代化	长子	贾敷
贾敬	兄	贾敷
贾代化	次子	贾敬
贾敬	子	贾珍
贾赦	子	贾琏
贾府	远房宗族	贾珩
贾府	远房宗族	贾瑞
贾敬	女	贾惜春
贾珍	胞妹	贾惜春
贾瑞	妹妹	喜鸾
贾代善	妻	史太君
保龄侯尚书令史公	后	史鼐
保龄侯尚书令	后	史鼎
忠靖侯	弟	史鼎
史鼐	弟	史鼎
史鼐	侄女	史湘云
史鼎	侄女	史湘云

表 2: 关系三元组

3 知识存储

3.1 数据库选择

我们选择的是图数据库管理系统 Neo4j

Neo4j 是原生图数据库，其使用的存储后端专门为图结构数据的存储和管理进行定制和优化

3.2 数据存储形式

- 以节点（nodes）和关系（relationships）来组织数据
  - 节点可以代表知识图谱中的实体
  - 关系可以用来代表实体间的关联，关系是有向的，两端对应开始节点和结束节点
- 节点上可加一个或多个标签（label）表示实体的分类
- 关系要有一个类别（type）来明确是何种类型的关系

- 节点和关系都可以有额外描述它们的属性（properties），即键值对（key-value pairs）

部分数据如下表3

head	relation	tail
贾琏	女	贾巧姐
王熙凤	女	贾巧姐
贾代善	妻	史太君
保龄侯尚书令史公	后	史鼐
保龄侯尚书令	后	史鼎
忠靖侯	弟	史鼎
史鼐	弟	史鼎
史鼐	侄女	史湘云
史鼎	侄女	史湘云
都太尉统制县伯	后	王子腾
都太尉统制县伯	后	王子胜

表 3: 数据三元组

3.3 关键代码

为了创建图数据库，首先要导入必要的 python 库 *py2neo*

```
import csv
import py2neo
from py2neo import Graph,Node,Relationship,NodeMatcher
```

接着使用 *Graph* 语句连接 *Neo4j*

```
g=Graph('bolt://localhost:7687',auth=("neo4j", "****")) #密码涉及机密
```

最后读取文件内容并添加节点和关系

```
with open('D:\desktop\知识表示\project\hlm.csv','r',encoding='utf-8') as f:
    reader=csv.reader(f)
    for item in reader:
        if reader.line_num==1:
            continue
        print("当前行数：",reader.line_num,"当前内容：",item)
        start_node=Node("Person",name=item[0])
        end_node=Node("Person",name=item[1])
        relation=Relationship(start_node,item[3],end_node)
        g.merge(start_node,"Person","name")
        g.merge(end_node,"Person","name")
        g.merge(relation,"Person","name")
```

3.4 实现效果

生成知识图谱如下图1所示:

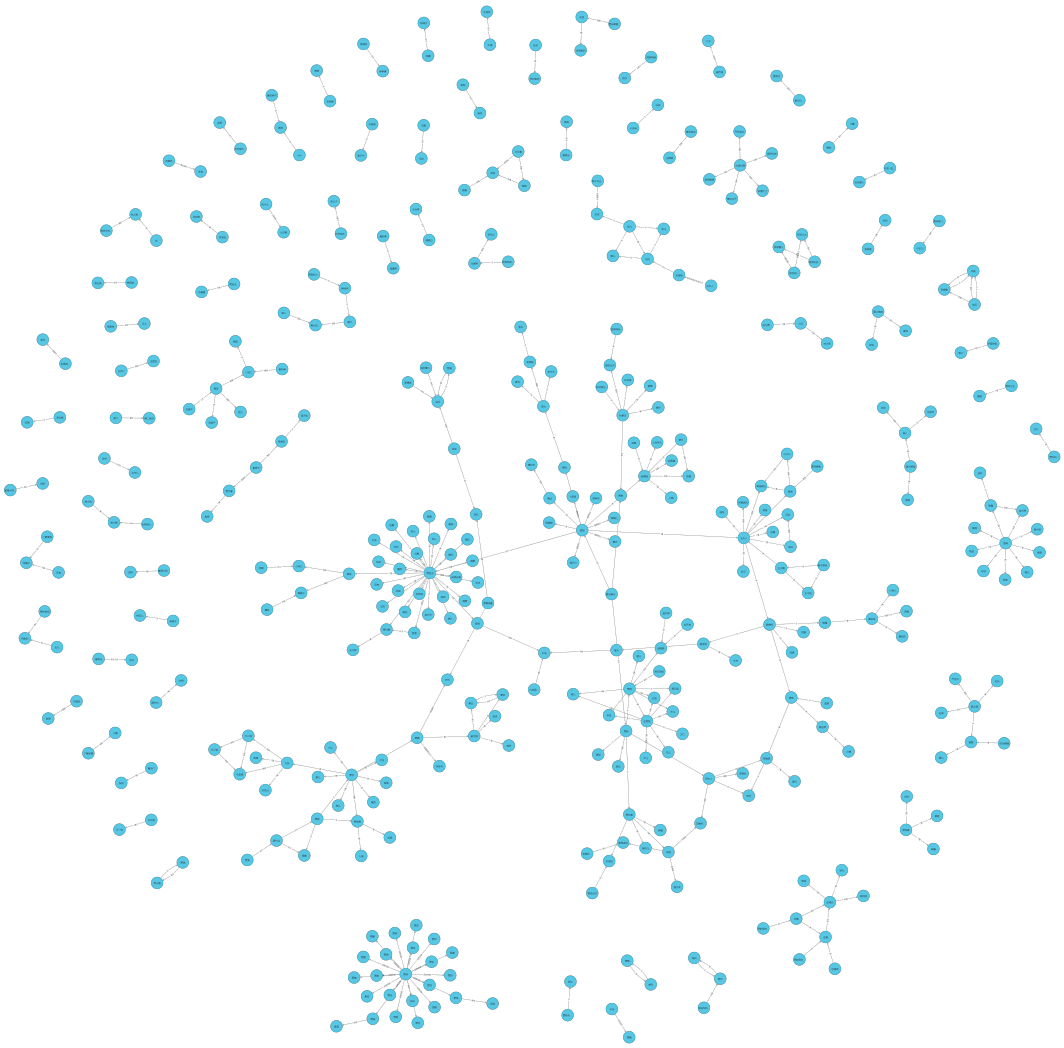

图 1: 知识图谱

我们也可以在 Neo4j 中进行查询
例如我们可以查询五个人的名字

```
MATCH (n:Person) RETURN n.name LIMIT 5
```

可以得到结果图2

```
neo4j$ MATCH (n:Person) RETURN n.name LIMIT 5
```

Table | Text | Code

	n.name
1	"贾代善"
2	"贾源"
3	"娄氏"
4	"贾母"
5	"老姨奶奶"

图 2: 查询结果 1

我们也可以查询“贾宝玉”和“叶妈”之间的最近路径

```
MATCH p=shortestPath(
(bacon:Person {name:"贾宝玉"})-[*]-(meg:Person {name:"叶妈"}))
RETURN p
```

可以得到如下结果图3：

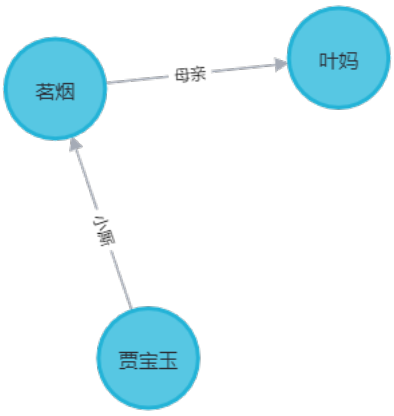

图 3: 查询结果 2

我们也可以查询“贾宝玉”通过最多两跳可到达的节点

```
MATCH (bacon:Person {name:"贾宝玉"})-[*1..2]-(hollywood)
RETURN DISTINCT bacon, hollywood
```

可以得到如下结果图4

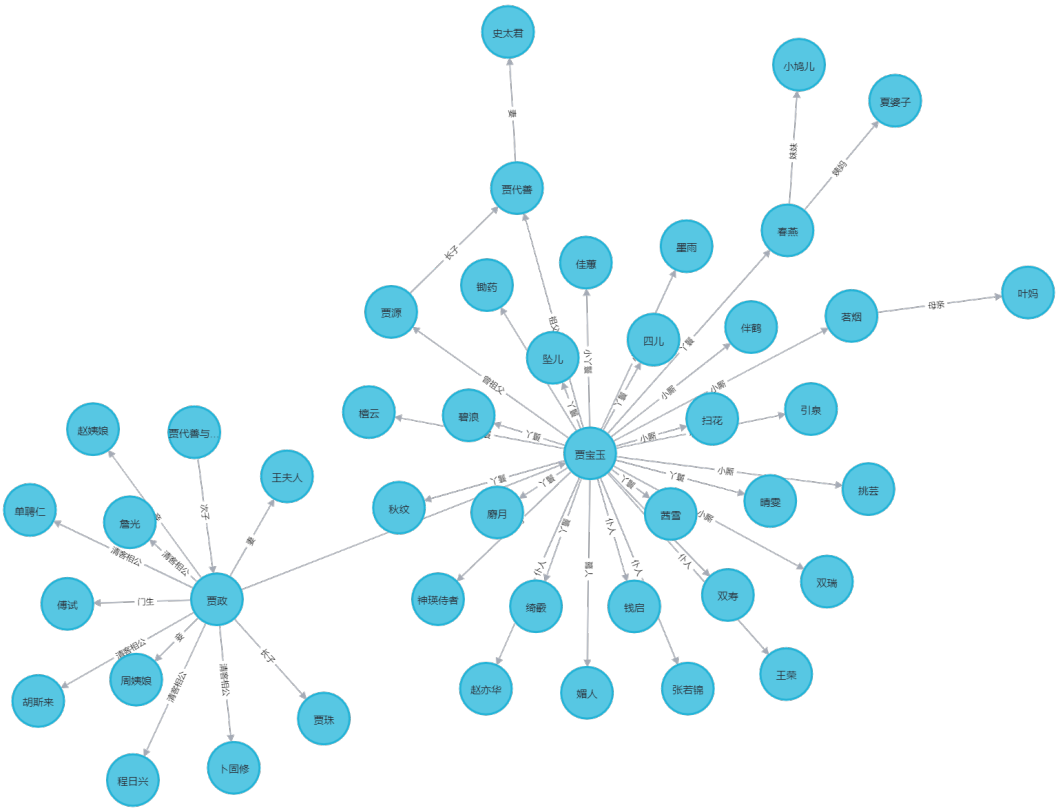

图 4: 查询结果 3

4 向量表示与推理

4.1 数据预处理

- 从 $raw_relations.csv$ 文件中读取数据，忽略任何不符合格式的错误行，并确保所有列都为字符串类型，处理任何可能的缺失值。
- 使用 pandas 读取 CSV 数据，并打印前几行，验证数据读取是否正确。

代码如下：

```
df = pd.read_csv('raw_relations.csv', on_bad_lines='skip')
df = df.fillna('').astype(str)
print(df.head())
```

4.1.1 构建三元组

- 使用 pykeen.triples.TriplesFactory 从清洗后的数据中创建三元组工厂，这是将关系数据转换为知识图谱的关键步骤。

代码如下：

```
    triples_factory = TriplesFactory.from_labeled_triples(
        df[['head', 'relation', 'tail']].values
    )
```

4.1.2 数据集划分

- 由于数据集大小的原因，如果将训练集划分太小不能包括大部分实体，所以我们将数据集按 90% 5% 5% 的比例划分为训练集、测试集和验证集。

```
    training, testing, validation = triples_factory.split([0.9, 0.05, 0.05])
```

4.2 使用 TransE 进行向量表示学习

4.2.1 训练 TransE 模型

- 使用 PyKEEN 提供的 pipeline 方法，训练 TransE 模型。TransE 模型是一种常见的嵌入模型，它将实体和关系嵌入到同一个向量空间，并假设关系可以通过向量的加减法来表示。
- 我们设置了 100 个训练周期，并指定在 CPU 上训练。

代码如下:

```
    result = pipeline(
        training=training,
        testing=testing,
        validation=validation,
        model='TransE',
        training_kwargs=dict(num_epochs=100),
        random_seed=42,
        device='cpu'
    )
```

4.2.2 获取训练后的模型

- 训练完成后，我们从结果中提取了训练好的 TransE 模型，以便进行后续的推理和评估。

```
    model = result.model
```

4.3 模型推理与评估

4.3.1 模型在测试集上的评估

- 定义 *evaluate_model_on_test_set* 函数，通过对测试集中的每个三元组进行评分来评估模型。
- 使用 *predict_triples_df* 函数对测试集中的每个三元组进行评分，获得每个三元组的打分，并按得分从高到低排序。

代码如下：

```
def evaluate_model_on_test_set(model, testing, triples_factory, top_k=10):
    testing_df = pd.DataFrame(
        testing.mapped_triples.numpy(),
        columns=['head_id', 'relation_id', 'tail_id']
    )

    id_to_entity = {v: k for k, v in triples_factory.entity_to_id.items()}
    id_to_relation = {v: k for k, v in
        triples_factory.relation_to_id.items()}

    testing_df['head_label'] = testing_df['head_id'].apply(lambda x:
        id_to_entity[x])
    testing_df['relation_label'] = testing_df['relation_id'].apply(lambda
        x: id_to_relation[x])
    testing_df['tail_label'] = testing_df['tail_id'].apply(lambda x:
        id_to_entity[x])

    triples = [(id_to_entity[row['head_id']],
        id_to_relation[row['relation_id']], id_to_entity[row['tail_id']])
            for index, row in testing_df.iterrows()]
    df = predict_triples_df(model=model, triples=triples,
        triples_factory=triples_factory)

    df_sorted = df.sort_values(by='score', ascending=True).head(top_k)
    return df_sorted
```

4.3.2 输出测试集的打分排序

- 调用上述函数对模型进行评估，并输出测试集中得分前 10 的三元组。

```
test_scores = evaluate_model_on_test_set(model, testing, triples_factory)
print("Test Set Scores Sorted:")
print(test_scores)
```

```
print("Top 10 Scored Triples from the Test Set:")
print(test_scores[['head_label', 'relation_label', 'tail_label',
    'score']].head(10))
```

4.4 实现结果

测试集的打分结果如下表4所示

head_id	head_label	relation_id	relation_label	tail_id	tail_label	score
149	智能	56	师傅	34	净虚	-8.992147
153	李嬷嬷	43	子	159	李贵	-8.3484535
184	狗儿	74	父	197	王成	-10.783632
191	王夫人	2	丫鬟	364	金钏	-8.057949
196	王子腾	61	弟	195	王子胜	-10.450742
221	秦业	43	子	224	秦钟	-9.641461
222	秦可卿	61	弟	224	秦钟	-10.587315
263	薛姨妈	24	女	265	薛宝钗	-10.217941
263	薛姨妈	43	子	267	薛蟠	-10.344901
266	薛蝌	35	妻	355	邢岫烟	-9.481351
284	贾代善与贾母	100	长子	341	贾赦	-10.152328
299	贾政	35	妻	191	王夫人	-9.106514
307	贾源	100	长子	283	贾代善	-8.252777
310	贾珍	43	子	336	贾蓉	-10.878411
316	贾琏	49	小厮	116	庆儿	-8.40028
336	贾蓉	35	妻	222	秦可卿	-11.380879
337	贾蓉续娶	35	妻	275	许氏	-9.440635
341	贾赦	24	女	342	贾迎春	-11.96797

表 4: 打分结果

我们给出头节点 (贾政) 和关系 (次子)，让训练好的模型预测尾节点得到的打分序列如下表5：

head_id	head_label	relation_id	relation_label	tail_id	tail_label	score
299	贾政	69	次子	299	贾政	-5.4965963
299	贾政	69	次子	287	贾宝玉	-7.2197566
299	贾政	69	次子	232	紫鹃	-7.715256
299	贾政	69	次子	43	卜固修	-7.7826433
299	贾政	69	次子	42	卜世仁	-7.8776665
299	贾政	69	次子	233	绛珠仙子	-7.920293
299	贾政	69	次子	31	冯渊	-7.923109
299	贾政	69	次子	360	金彩	-7.9267545
299	贾政	69	次子	117	度恨菩提	-7.9645643
299	贾政	69	次子	112	尤老娘	-8.020677

表 5: 预测打分结果排序

4.5 模型评估

我们得到模型输出的最高指标如下表6

Mean Rank	Hits@10
139.11111450195312	0.16666666666666666

表 6: 评估指标 1

可以看到，本实验的模型评估结果显示，TransE 模型在知识图谱上的性能较差。这些结果表明模型在预测头实体和尾实体之间的关系时准确率非常低。

可能的原因如下：

- 数据集规模小: 我们检查了读取的有效三元组，只有三百多个，数据集规模不足以捕捉到足够的模式和关系，导致模型难以泛化。尽管处理后的数据集有上千条三元组，但有效的训练样本不足。
- 数据质量: 数据集的噪声和不一致性也可能影响模型性能。我们抽取到三元组中的某些关系标签或实体名称存在错误或不一致，可能会对模型的训练和预测造成干扰。
- 模型过于简单: TransE 模型虽然计算效率高，但对于复杂的多关系数据可能表现不佳。TransE 通过向量平移来表示关系，可能不足以捕捉到复杂关系的模式。
- 数据稀疏性: 我们得到的红楼梦数据集中实体和关系的分布非常不均匀，导致某些实体和关系的训练样本很少，严重影响模型的学习效果。
- 数据性质和复杂性：《红楼梦》中的人物关系非常复杂，包含了家庭关系、社会地位、情感纠葛等多种类型。TransE 仅使用一个向量来表示每种关系，无法捕捉这些复杂关系的多样性。而且《红楼梦》中的人物往往有多重关系交织在一起，这种关系嵌套和交织难以通过 TransE 的简单平移操作来准确表示。

4.6 K-fold

由于得到的数据集规模过小，我们考虑了使用 K 折交叉验证，并将 k 设置为 5.

关键代码如下：

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
for train_index, test_index in kf.split(triples):
    train_triples, test_triples = triples[train_index], triples[test_index]
```

```
    # 创建训练和测试三元组工厂
    train_factory = TriplesFactory.from_labeled_triples(train_triples)
    test_factory = TriplesFactory.from_labeled_triples(test_triples)

    # 使用 PyKEEN 的 pipeline 训练 TransE 模型
    result = pipeline(
        training=train_factory,
        testing=test_factory,
        model='TransE',
        training_kwargs=dict(num_epochs=200),
        random_seed=42,
        device='cpu'  # 或 'cuda' 来使用 GPU
    )

    # 获取训练好的模型
    model = result.model

    # 评估模型
    evaluator = RankBasedEvaluator()
    evaluation_results = evaluator.evaluate(model,
        mapped_triples=test_factory.mapped_triples)

    mean_rank = evaluation_results.get_metric('mean_rank')
    hits_10 = evaluation_results.get_metric('hits@10')

    mean_ranks.append(mean_rank)
    hits_at_10.append(hits_10)

    print(f"Fold {len(mean_ranks)} - Mean Rank: {mean_rank}, Hits@10:
        {hits_10}")
```

得到的结果如下表7

Fold	Mean Rank	Hits@10
1	196.76589965820312	0.046242774566473986
2	191.93466186523438	0.03125
3	187.9476776123047	0.0377906976744186
4	211.07830810546875	0.015060240963855422
5	191.51170349121094	0.017543859649122806
avera	195.84765014648437	0.029577514570774167

表 7: 评估指标 2

但是从结果中我们发现，我们得到的模型效果依然不好。

4.7 结论

通过上述步骤，我们使用 TransE 模型对知识图谱进行了嵌入和推理。以下是模型评估的几个关键点：

- 向量表示：
  - TransE 模型将每个实体和关系映射到低维向量空间中，使得复杂的关系可以通过向量运算来表示。例如，如果有一个三元组 (‘贾政’, ‘次子’, ‘贾宝玉’)，那么模型会学习到向量‘贾政’与向量‘贾宝玉’之间的差异近似于向量‘次子’。
- 推理能力:
  - 通过对测试集的评分，我们可以看到模型对未见过的三元组进行了评分，并根据评分预测了最有可能的尾实体和头实体。这种能力在知识图谱补全任务中非常有用，可以帮助填补缺失的关系。
- 结果排序:
  - 评估过程中，模型根据三元组的得分对可能的关系进行了排序，得分高的三元组更有可能是真实的关系。通过输出得分最高的三元组，我们可以验证模型的预测能力。

本次实验成功地展示了如何使用 PyKEEN 框架进行知识图谱的向量表示和推理，但是由上面的结果我们可以得到，TransE 的简单平移操作虽然在处理简单关系时有优势，但面对《红楼梦》这样复杂且多样的人物关系时显得力不从心。要提升对这类复杂关系的预测效果，可能需要更先进的模型，如考虑上下文信息的图神经网络（GNN）或利用预训练的语言模型来捕捉文本中的丰富语义信息。

5 总结

在本项目中，我们成功地构建了一个《红楼梦》人物关系的知识图谱，并尝试利用 TransE 模型进行关系推理。然而，尽管我们应用了 K-fold 交叉验证等技术来提升模型的准确性，效果依然不尽理想。主要原因在于 TransE 模型对复杂关系的表示能力有限，而《红楼梦》中的人物关系复杂多样，简单的向量加减无法准确地捕捉这种复杂性。此外，数据的不均衡性和部分人物关系的模糊性也对模型的训练和评估带来了挑战。尽管如此，我们的工作为

《红楼梦》知识图谱的进一步研究打下了基础，未来可以尝试引入更复杂的模型和更精细的数据处理方法，以提高对人物关系的预测能力。

我们希望通过本项目，展示将经典文学作品与现代信息技术相结合的可能性，也为其他类似的文学作品提供一个有价值的参考案例。