lab2

PB21051012 刘祥辉 PB20050990吕林峰 PB20020574 任呈祥

stage1

读取douban2fb.txt,获得实体映射关系,创建一个空的 Pandas DataFrame，命名为 `pd_triplet`,记录头尾节点和关系信息

```
FreebaseID = set()
with open("douban2fb.txt", "r", encoding="utf-8") as file:
    for line in file:
        FreebaseID.add(line.split("\t")[1].strip())
pd_triplet = pd.DataFrame()
pd_triplet['head'] = None
pd_triplet['relation'] = None
pd_triplet['tail'] = None
pd_triplet.to_csv("freebase_douban.csv", index=False)
```

构建第一跳,将三元组中实体在豆瓣ID映射实体加入pd_triplet,然后生成freebase_douban.csv,包含第一跳未经筛选的三元组信息.

```
j = 0
with gzip.open("freebase_douban.gz") as f:
    for line in f:
        line = line.strip()
        triplet = line.decode().split('\t')[:3]
        if (match_entity(triplet[0].strip("<>").split('/')[-1]) or
match_entity(triplet[2].strip("<>").split('/')[-1])) :
                        if triplet[2].startswith("
<http://rdf.freebase.com/ns/m") and triplet[2].startswith("
<http://rdf.freebase.com/ns/m"):
                    new_row = {"head":triplet[0].strip("
<>"),"relation":triplet[1].strip("<>"),"tail":triplet[2].strip("<>")}
                    pd_triplet=pd.concat([pd_triplet, pd.DataFrame([new_row])],
ignore_index=True)
                    # pd_triplet = pd_triplet.append(new_row,ignore_index=True)
                    FreebaseID.add(triplet[0].strip("<>").split('/')[-1])
                    FreebaseID.add(triplet[2].strip("<>").split('/')[-1])
                j += 1
                if(j % 1000000==0):
                  print(j)


pd_triplet.to_csv('freebase_douban.csv', mode='a', index=False, header = False)
temp = pd.read_csv('freebase_douban.csv')
print(j)
print("Length of pd_triplet:", len(temp))
```

1. **实体与关系出现次数的统计：** 使用字典 `entity_num` 统计了每个实体出现的次数，并使用字典 `relation_num` 统计了每个关系出现的次数。这是通过迭代 DataFrame 中的每一行实现的。

2. **过滤实体和关系：** 根据设定的条件，保留了至少出现在 20 个三元组中的实体，并只保留出现超过 50 次的关系。通过遍历 DataFrame 的每一行，将不符合条件的行的索引添加到 `to_be_delete` 列表中。
3. **删除不符合条件的行：** 使用 Pandas 的 `drop` 方法删除了 `to_be_delete` 列表中的行，以实现过滤。
4. **获取第二跳的实体：** 根据设定条件，获取了第二跳的实体。

```
#统计不同实体出现次数
entity_num = {key: 0 for key in FreebaseID}
relation_num = dict()
for index, row in pd_triplet.iterrows():
    head = row['head']
    relation = row['relation']
    tail = row['tail']
    if head.split('/')[-1] in entity_num:
        entity_num[head.split('/')[-1]] += 1
    if tail.split('/')[-1] in entity_num:
        entity_num[tail.split('/')[-1]] += 1
    if relation in relation_num:
        relation_num[relation] += 1
    else:
        relation_num[relation] = 0
to_be_delete = []
# 保留了至少出现在 20 个三元组中的实体，同时只保留出现超过 50 次的关系
for index, row in pd_triplet.iterrows():
    if relation_num[row['relation']] < 50 :
        to_be_delete.append(index)
pd_triplet = pd_triplet.drop(to_be_delete)
# 得到第二跳的实体
FreebaseID.clear()
for index, row in pd_triplet.iterrows():
    if row['head'].startswith("http://rdf.freebase.com/ns/") and
entity_num[row['head'].split('/')[-1]] >= 20:
        FreebaseID.add(row['head'].split('/')[-1])
    if row['tail'].startswith("http://rdf.freebase.com/ns/") and
entity_num[row['tail'].split('/')[-1]] >= 20:
        FreebaseID.add(row['tail'].split('/')[-1])
```

与一跳类似,根据第一跳获得的实体,重新遍历freebase_douban.gz,获得第二跳未筛选的三元组信息

```
j = 0
entity_num = {key: 0 for key in second_FreebaseID}
with gzip.open("freebase_douban.gz") as f:
    for line in f:
        line = line.strip()
        triplet = line.decode().split('\t')[:3]
        if triplet[0].strip("<>").split('/')[-1]  in second_ori_FreebaseID:
            new_row = {"head":triplet[0].strip("
<>"),"relation":triplet[1].strip("<>"),"tail":triplet[2].strip("<>")}
            pd_triplet=pd.concat([pd_triplet, pd.DataFrame([new_row])],
ignore_index=True)
            if(triplet[2].startswith("<http://rdf.freebase.com/ns/")):
                second_FreebaseID.add(triplet[2].strip("<>").split('/')[-1])
            entity_num[triplet[0].strip("<>").split('/')[-1]] += 1
```

```
            if entity_num[triplet[0].strip("<>").split('/')[-1]] >= 20000:  #有的实体出现太多次了！
                second_ori_FreebaseID.remove(triplet[0].strip("<>").split('/')[-1])
                print("i have removed:",triplet[0].strip("<>").split('/')[-1])
        j += 1
        if(j % 1000000==0):
          temp = pd.read_csv('second_freebase_douban.csv')
          print(j)
          print("Length of pd_triplet:", len(temp))
        if( len(pd_triplet) > 10000):
          pd_triplet.to_csv('second_freebase_douban.csv', mode='a', index=False, header = False)
          pd_triplet = pd_triplet[0:0]

pd_triplet.to_csv('second_freebase_douban.csv', mode='a', index=False, header = False)
temp = pd.read_csv('second_freebase_douban.csv')
print(j)
print("Length of pd_triplet:", len(temp))
```

1. **第二跳实体与关系出现次数的统计：** 使用字典 `second_entity_num` 统计了每个第二跳实体出现的次数，并使用字典 `second_relation_num` 统计了每个关系出现的次数。这是通过迭代 DataFrame 中的每一行实现的。
2. **过滤实体和关系：** 根据设定的条件，保留了至少出现在 18 到 19500 个三元组中的实体，并且只保留出现超过 50 次的关系。通过遍历 DataFrame 的每一行，将不符合条件的行的索引添加到 `to_be_delete` 列表中。
3. **删除不符合条件的行：** 使用 Pandas 的 `drop` 方法删除了 `to_be_delete` 列表中的行，以实现过滤。
4. **获取第二跳的实体：** 根据设定条件，获取了第二跳的实体。

```
pd_triplet = pd.read_csv('second_freebase_douban.csv')
#统计不同实体出现次数
second_entity_num = {key: 0 for key in second_FreebaseID}
second_relation_num = dict()
for index, row in pd_triplet.iterrows():
    head = row['head']
    relation = row['relation']
    tail = row['tail']
    if head.split('/')[-1] in second_entity_num:
        second_entity_num[head.split('/')[-1]] += 1
    if tail.split('/')[-1] in second_entity_num:
        second_entity_num[tail.split('/')[-1]] += 1
    if relation in second_relation_num:
        second_relation_num[relation] += 1
    else:
        second_relation_num[relation] = 0
to_be_delete = []
# 保留了至少出现在 20 个三元组中的实体，同时只保留出现超过 50 次的关系
for index, row in pd_triplet.iterrows():
    flag = (row['head'].split('/')[-1] in second_entity_num and 19500 >= second_entity_num[row['head'].split('/')[-1]] >= 18) and (row['tail'].split('/')[-1] in second_entity_num and 19500 >= second_entity_num[row['tail'].split('/')[-1]] >= 18)
    if second_relation_num[row['relation']] < 50 or not flag :
        to_be_delete.append(index)
```

```
pd_triplet = pd_triplet.drop(to_be_delete)
# 得到第二跳的实体
second_FreebaseID.clear()
for index, row in pd_triplet.iterrows():
    if row['head'].startswith("http://rdf.freebase.com/ns/") :
        second_FreebaseID.add(row['head'].split('/')[-1])
    if row['tail'].startswith("http://rdf.freebase.com/ns/") :
        second_FreebaseID.add(row['tail'].split('/')[-1])

# 得到第二跳剩余的关系:
second_relation_num.clear()
for index, row in pd_triplet.iterrows():
    head = row['head']
    relation = row['relation']
    tail = row['tail']
    if relation in second_relation_num:
        second_relation_num[relation] += 1
    else:
        second_relation_num[relation] = 0
```

stage2

知识图谱映射:

```
import pandas as pd

def MovieMap(givenmap,selected_entity) -> list:
    value = 0
    with open(givenmap, "r") as file:
        given_movie_lines = file.readlines()
    movie_id = {}
    for line in given_movie_lines:
        name = line.strip().split('\t')[1]
        movie_id[name] = value
        value += 1
    with open(selected_entity,"r") as file:
        selected_movie_lines = file.readlines()
    for line in selected_movie_lines:
        name = line.strip()
        if name not in movie_id:
            movie_id[name] = value
            value += 1
    return movie_id
def RelationMap(relaiton_dic) ->dict:
    value = 0
    with open(relaiton_dic, "r") as file:
        lines = file.readlines()
    relation_id = {}
    for line in lines:
        name, _ = line.strip().split(': ')
        relation_id[name] = value
        value += 1
    return  relation_id
def KgMap(KG_Graph,movie_id_map,relation_map):
    with open("stage2/data/Douban/kg_final.txt", "w") as file:
```

```
        for _ ,element in KG_Graph.iterrows():
            file.write(f"{movie_id_map[element['head'].split('/')[-1]]}
{relation_map[element['relation']]} {movie_id_map[element['tail'].split('/')
[-1]]}\n")

givenmap = "douban2fb.txt"
selected_entity = "stage1/second_selected_entity"
relaiton_dic = "stage1/movie_relation"
movie_id_map = MovieMap(givenmap,selected_entity)
relation_map = RelationMap(relaiton_dic)
KG_Graph = pd.read_csv("stage1/second_selected_freebase_douban.csv")
KgMap(KG_Graph,movie_id_map,relation_map)
```

基于图谱嵌入的模型

dataloder的补充:

调用`rename`函数和`concat`函数来实现为KG添加逆向三元组和三元组的拼接

关系数则为`kg_data`中r列的最大值再加上1

实体数为`kg_data`中h列和t列中的最大值加1

`reverse_kg["r"] = reverse_kg["r"].apply(lambda x: x + relation_num)` apply函数得到`r+n_relations`

```
    def construct_data(self, kg_data):
        #TODO:
        '''
            kg_data 为 DataFrame 类型
        '''
        # 1. 为KG添加逆向三元组，即对于KG中任意三元组(h, r, t)，添加逆向三元组 (t,
r+n_relations, h)，
        #    并将原三元组和逆向三元组拼接为新的DataFrame，保存在 self.kg_data 中。
        reverse_kg = kg_data[["t","r","h"]]
        reverse_kg = reverse_kg.rename(columns = {"t":"h","h":"t"})
        relation_num = len(kg_data["r"].unique())
        reverse_kg["r"] = reverse_kg["r"].apply(lambda x: x + relation_num)

        self.kg_data = pd.concat([kg_data,reverse_kg],axis=0,ignore_index=True)

        # 2. 计算关系数，实体数和三元组的数量
        self.n_relations = len(self.kg_data["r"].unique()) + 1
        self.n_entities =
 max(len(self.kg_data["h"].unique()),len(self.kg_data["r"].unique())) + 1
        self.n_kg_data =   len(self.kg_data)

        # 3. 根据 self.kg_data 构建字典 self.kg_dict ，其中key为h, value为tuple(t,
r),
        #    和字典 self.relation_dict，其中key为r, value为tuple(h, t)。
        self.kg_dict = collections.defaultdict(list)
        self.relation_dict = collections.defaultdict(list)

        for _  ,  content in self.kg_data.iterrows():
            h = content["h"]
            r = content["r"]
            t = content["t"]
            self.kg_dict[h].append((t,r))
```

```
            self.relation_dict[r].append((h,t))
```

TransE函数的补充：

需要对关系嵌入、头实体嵌入、尾实体嵌入以及负采样的尾实体嵌入进行L2归一化处理

使用`torch`中的`normalize`函数实现三元组的得分涉及向量距离的运算。

```
        # 5. 对关系嵌入，头实体嵌入，尾实体嵌入，负采样的尾实体嵌入进行L2范数归一化
        r_embed = F.normalize(r_embed,p=2,dim=1)
        h_embed = F.normalize(h_embed,p=2,dim=1)
        pos_t_embed = F.normalize(pos_t_embed,p=2,dim=1)
        neg_t_embed = F.normalize(neg_t_embed,p=2,dim=1)
        # 6. 分别计算正样本三元组 (h_embed, r_embed, pos_t_embed) 和负样本三元组
(h_embed, r_embed, neg_t_embed) 的得分
        pos_score = torch.sum(torch.pow(h_embed + r_embed - pos_t_embed, 2),
dim=1)
        neg_score = torch.sum(torch.pow(h_embed + r_embed - neg_t_embed, 2),
dim=1)
        # 7. 使用 BPR Loss 进行优化，尽可能使负样本的得分大于正样本的得分
        kg_loss = (-1.0) * F.logsigmoid(pos_score - neg_score)
        kg_loss = torch.mean(kg_loss)
```

TransR函数的补充

利用torch中的squeeze函数进行维度运算

```
        # 1. 计算头实体，尾实体和负采样的尾实体在对应关系空间中的投影嵌入
        r_mul_h = torch.bmm(h_embed.unsqueeze(1), W_r).squeeze(1)
        r_mul_pos_t = torch.bmm(pos_t_embed.unsqueeze(1), W_r).squeeze(1)
        r_mul_neg_t = torch.bmm(neg_t_embed.unsqueeze(1), W_r).squeeze(1)
        # 2. 对关系嵌入，头实体嵌入，尾实体嵌入，负采样的尾实体嵌入进行L2范数归一化
        r_embed = F.normalize(r_embed,p=2,dim=1)
        r_mul_h = F.normalize(r_mul_h,p=2,dim=1)
        r_mul_pos_t = F.normalize(r_mul_pos_t,p=2,dim=1)
        r_mul_neg_t = F.normalize(r_mul_neg_t,p=2,dim=1)
        # 3. 分别计算正样本三元组 (h_embed, r_embed, pos_t_embed) 和负样本三元组
(h_embed, r_embed, neg_t_embed) 的得分
        pos_score = torch.sum(torch.pow(r_mul_h + r_embed - r_mul_pos_t, 2),
dim=1)
        neg_score = torch.sum(torch.pow(r_mul_h + r_embed - r_mul_neg_t, 2),
dim=1)
        # 4. 使用 BPR Loss 进行优化，尽可能使负样本的得分大于正样本的得分
        kg_loss = (float)(-1) * F.logsigmoid(neg_score - pos_score)
        kg_loss = torch.mean(kg_loss)
```

注入图谱实体语义信息的方式

```
# 8. 为 物品嵌入 注入 实体嵌入的语义信息
item_pos_cf_embed = item_pos_embed+item_pos_kg_embed
item_neg_cf_embed = item_neg_embed+item_neg_kg_embed
```

```
# 9. 为 物品嵌入 注入 实体嵌入的语义信息
item_cf_embed =  item_embed+item_kg_embed
```

实验结果

```
KG free
2023-12-22 19:26:03,927 - root - INFO - Best CF Evaluation: Epoch 0040 | Precision [0.2966, 0.2532], Recall [0.0660, 0.1094], NDCG [0.3110, 0.2829]
相加、TransE：
2023-12-15 17:37:43,134 - root - INFO - Best CF Evaluation: Epoch 0030 | Precision [0.2931, 0.2530], Recall [0.0650, 0.1106], NDCG [0.3105, 0.2842]
相乘、TransE:
2023-12-15 17:54:13,153 - root - INFO - Best CF Evaluation: Epoch 0030 | Precision [0.2805, 0.2517], Recall [0.0628, 0.1104], NDCG [0.2902, 0.2737]
拼接、TransE:
2023-12-15 18:20:40,686 - root - INFO - Best CF Evaluation: Epoch 0030 | Precision [0.2931, 0.2532], Recall [0.0650, 0.1111], NDCG [0.3108, 0.2846]

相加、TransR：
2023-12-15 19:29:30,751 - root - INFO - Best CF Evaluation: Epoch 0040 | Precision [0.2940, 0.2521], Recall [0.0633, 0.1100], NDCG [0.3097, 0.2816]
相乘、TransR:
2023-12-15 19:14:03,306 - root - INFO - Best CF Evaluation: Epoch 0030 | Precision [0.2832, 0.2503], Recall [0.0634, 0.1104], NDCG [0.2964, 0.2757]
拼接、TransR:
2023-12-15 18:35:14,409 - root - INFO - Best CF Evaluation: Epoch 0040 | Precision [0.2944, 0.2526], Recall [0.0633, 0.1101], NDCG [0.3100, 0.2819]
```

	Recall@5	Recall@10	NDCG@5	NDCG@10
KG free	0.0660	0.1094	0.3110	0.2829
相加的注入实体信息 + TransE	0.0650	0.1106	0.3105	0.2842
相加的注入实体信息 + TransR	0.0633	0.1100	0.3097	0.2816
相乘的注入实体信息 + TransE	0.0628	0.1104	0.2902	0.2737
相乘的注入实体信息 + TransR	0.0634	0.1104	0.2964	0.2757
拼接的注入实体信息 + TransE	0.0650	0.1111	0.3108	0.2846
拼接的注入实体信息 + TransR	0.0633	0.1101	0.3100	0.2819

结果分析

对于不同的注入信息方式，采用拼接的注入效果比相加、相乘的注入方式效果要好

对于TransE和TransR两种方法，在Recall指标上，TransE效果要优于TransR,在NDCG指标上，TranR效果要优于TransE

相较于KG free，采用图谱嵌入的方法对于NDCG指标有一定的提升，但是效果不算明显，分析原因可能因为：

1. **知识图谱问题:**
    - 本次提取的实体可能与待检测信息的关联度不是很高，导致图谱嵌入的有效性降低
2. **超参数调优问题:**
    - 可以适当调整学习率和训练轮数来获得更优的参数
3. **未考虑上下文信息:**

- 有时仅仅考虑实体和关系之间的直接连接关系可能不足以捕捉上下文信息。可以考虑引入更复杂的模型，例如引入注意力机制或者更高级的图神经网络。