1. LoadCorpus.py
功能：
读取训练文件，问题 id、问题、sparql、答案实体。
问题被去除一些杂项。
sparql 可以提取用于查询的 gold entity 和 gold relation
数据项为 dict，键有'id'、'question'、'answer'、'gold_entities'、'gold_relations'、'sparql'。

这里提供了关于查询的统计信息，可能是该系统能解决的问题类型：
一个实体一个关系，e r ?x
一个实体两个关系，e r1 ?a . ?a r2 ?x
两个实体两个关系，{ e1 r1 e2 . e2 r2 ?x }
两个实体一个以内的关系
其他

2. train_ner.py
功能：
建立 BERT-BiLSTM-fc 的序列标注模型 NERBERT，用于提取问题中的 mention
ground truth 是 gold entity 与问题字符重合的部分为 1，其他为 0

这里不能保证该 mention 在正确的实体 mention 中

效果：
400 条数据：precision 0.7818, recall 0.7083, f1 0.7432
epoch 13 | train loss 8.6860 | test f1-score 0.7446, precision 0.8025, recall 0.694

3. GetSegmentDict.py
功能：
从知识库文件和实体链接文件抽取实体的 mention，建立 segment_dict.txt
每一行为一个实体的 mention
从实体链接文件抽取 literal（作者将 literal 统称为属性值，property），建立 property_dict.json 文件，
每一行为 literal，及其对应的频率

问题：
pkubase-complete 中多个实体有下划线，mention2ent 有不标准的三元组，因此 segment_dict 有无效实体

4. mention_extractor.py
功能：
对每一个数据项，从问题中，利用 jieba 分词和 NERBERT 模型抽取**实体的 mention**。
jieba 分词的实体保证在 segment_dict 里，而 NERBERT 模型不保证。
返回一个字典：**{ mention: mention }**

5. GetChar2Prop.py

功能：
从 property_dict.json 中，读取每个 literal，建立每个字符到 literal 的映射，char2property.json 文件。

6. property_extractor.py
功能：
对每个问题，抽取 5 个类型的 literal：
双引号、书名号内容 mark_properties
日期 time_properties
数字 digit_properties
　　以上用 re 抽取
其他类型 other_properties
　　按字符匹配查找属性值，并过滤掉较短的、不是地名的属性值，保留符合一些自定义条件的属性值
模糊匹配 fuzzy_properties
　　从问题的字符查找属性值，并根据频率，保留前三的属性值
返回**{class: {property: mention}}**

统计单个属性值作为单个 gold entity 的抽取效果

问题：
日期是否全部覆盖？中文年月日、9.8、08.8
stop 属性和词性定义是否有效？

效果：
单主语且主语为属性值问题中，能找到所有主语属性值的比例为:0.69
平均每个问题属性为:7.22
[139, 185, 195, 199, 337, 344, 406, 426, 465, 530, 547, 758, 820, 827, 839, 867, 985, 1036, 1057, 1272, 1276, 1310, 1330, 1357, 1372, 1427, 1463, 1618, 1690, 1699, 1701, 1826, 1859, 1901, 2022, 2126, 2169, 2273, 2344, 2366, 2520, 2531, 2592, 2593, 2642, 2687, 2702, 2719, 2730, 2754, 2763, 2766, 2797, 2799, 2851, 2872, 2882, 2887, 2996]

单主语且主语为属性值问题中，能找到所有主语属性值的比例为:0.79
平均每个问题属性为:7.21
[102, 140, 167, 263, 377, 422, 610, 683, 751, 756, 769, 960]
耗费时间 2685.41s

7. GetMentionDict.py
功能：
从实体链接文件中，提取 mention 到实体的映射，mention2entity.json 文件
每行格式为 mention entity

8. entity_extractor.py
功能：

从实体 mention 和属性值 mention 抽取候选实体，并将抽取失败的数据项的问题、gold entity、候选实体写入文件
对每个实体 mention，根据词性、停用 mention，不在 mention2entity 中过滤一些实体，然后计算每个实体的特征：mention 与问题相关的特征、实体与问题与关系的特征、实体在知识库中的流行度
对属性值 mention，计算相似特征
返回字典：
{entity: feature}
feature:[mention, mention features*3, similar feature*8, popular feature]

问题：
这里这个特征其实很有改进的空间
stop 词性和 mention 是否有效？

效果：
单实体问题可召回比例为：0.00
所有问题可召回比例为：0.70
平均每个问题的候选主语个数为：18.93

9. entity_filter.py
功能：
用 sklearn 线性回归模型对候选实体进行 topn 筛选，
模型输入为实体特征，ground truth 为该实体是否在 gold entity 中

效果：
所有问题候选主语召回率为：0.697 其中单主语问题为：0.761
所有问题候选主语召回率为：0.693 其中单主语问题为：0.757
在验证集上逻辑回归 top20 筛选后，所有问题实体召回率为 0.681，单实体问题实体召回率 0.748

10. similarity.py
功能：
建立 BERT 文本匹配模型，BERTSim
输入是问题和抽取的三元组转换成的句子对，输出为句子对是否匹配

效果：
epoch: 4, train loss: 24.0914, dev acc: 0.6909, dev f1: 0.7041, my f1: 0.7041

11. tuple_extractor.py
功能：
对每个数据项，根据候选实体和问题，抽取答案查询路劲，即实体、关系组成的 sparql 关键查询。
返回字典，{ tuple: feature }
新的 feature 为实体抽取中的特征外，BertSim 模型计算的匹配度

效果：
验证集：
所有问题里，候选答案能覆盖标准查询路径的比例为:0.497
单实体问题中，候选答案能覆盖标准查询路径的比例为:0.653
平均每个问题的候选答案数量为:438.456

12. tuple_filter.py
功能：
用 sklearn 的线性回归模型对候选路径进行筛选
输入特征为匹配相似度，ground truth 是该 tuple 是否为 gold tuple，由于数据杂项特别多，按概率挑选负例

效果：
训练集
单实体问题中，候选答案可召回的的比例为:0.636
候选答案能覆盖标准查询路径的比例为:0.480
验证集
单实体问题中，候选答案可召回的的比例为:0.646
候选答案能覆盖标准查询路径的比例为:0.488

在验证集上逻辑回归筛选后 top30 召回率为 0.64

13. answer_bot.py
功能：
整合前述过程，建立系统的测试流程

效果：
average f1: 0.2635

没有实现补充规则，效果很差