《喜羊羊与灰太狼》知识图谱构建

本项目以动画《喜羊羊与灰太狼》为例，构建其中角色的人物关系知识图谱。项目任务一共分为四个部分：

1）数据爬取与预处理

2）命名实体识别

3）实体关系抽取

4）可视化与知识问答

对应项目代码分别在`喜羊羊与灰太狼/数据爬取`，`喜羊羊与灰太狼/实体识别`，`喜羊羊与灰太狼/关系抽取`，`喜羊羊与灰太狼/可视化与知识问答`目录中。

1 数据爬取与预处理

在构建知识图谱之前，需要收集项目需要的文本数据，从而进行实体（人物、场地）和关系的识别抽取。主要从《喜羊羊与灰太狼》维基百科网站和百度梗概文本获得信息，对应项目代码在`喜羊羊与灰太狼/数据爬取`目录下。

1.1 半结构化数据获取

我们选择维基百科（喜羊羊与灰太狼 - 羊羊百科，青青草原的百科全书 - 灰机wiki (huijiwiki.com)）中的半结构化数据文本作为爬取的目标。结构输出为包含两个实体一个关系的三元组格式，作为后面利用抽取技术结果的补充和比对。

1.1.1 实体获取

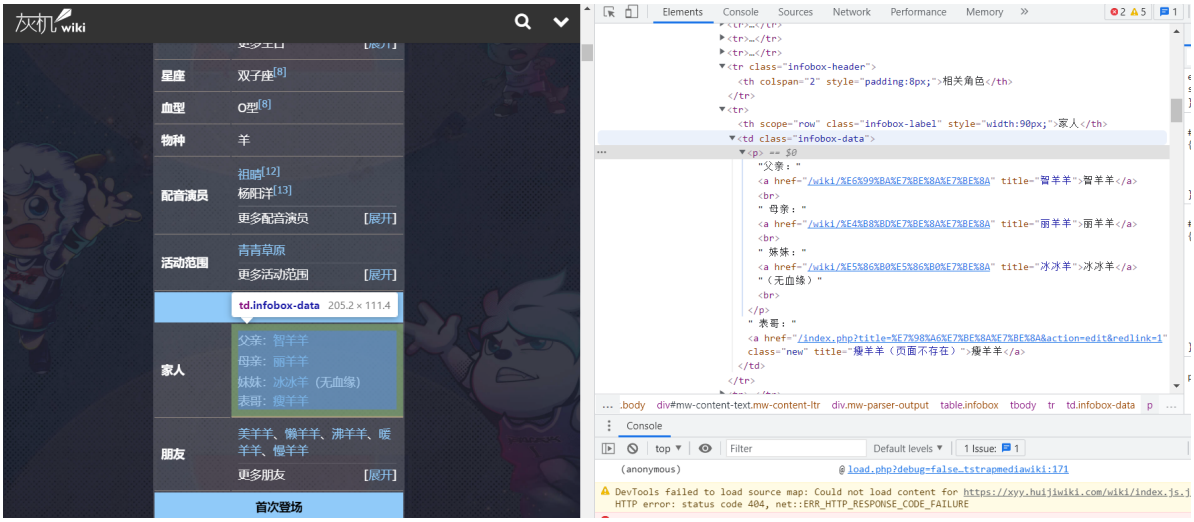
由于每个角色的个人页面都是"`https://xyy.huijiwiki.com/wiki/角色名`"格式，我们选择在"角色目录"下爬取所有出场人物，并保存到`entity.txt`中，具体代码在`喜羊羊与灰太狼/数据爬取/crawler_entity.py`目录下。

```
session = HTMLSession()
response = session.get(url)
a_list = response.html.find('a')
cur_ents = []
for a in a_list:
    herf_ent = a.attrs.get('href','')
    if '/wiki/%' in herf_ent:
        if('title' in a.attrs):
            cur_ents.append(a.attrs['title'])
for t in cur_ents:
    all_entities.append(t)
```

1.1.2 三元组获取

获取所有可以访问主页的结构后，对其半结构化数据进行分析，并从中抽取出关系三元组，具体代码在`喜羊羊与灰太狼/数据爬取/crawler_data.py`目录下。

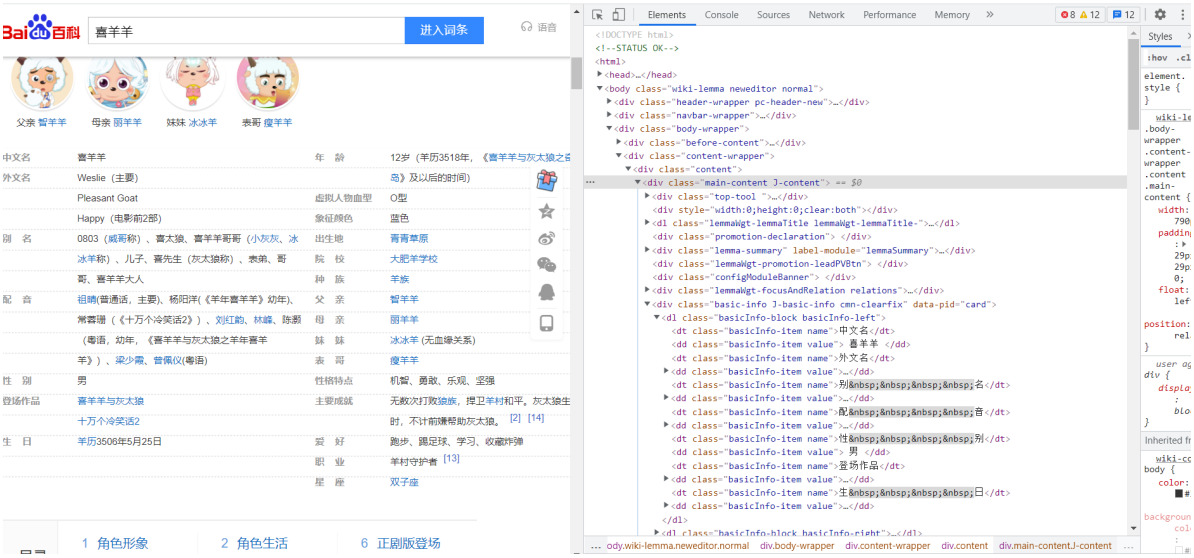
首先，检查wiki角色主页源代码，查看人物关系源代码格式如下：



接着，根据人物关系源代码格式，利用Beautiful Soup工具进行数据爬取，调用`extract_entity(ent)`函数用于实现，并将关系三元组爬取结果保存到`data_raw.txt`中。

```
def extract_entity(ent):
    url = root + ent
    response = requests.get(url=url,headers=headers).content
    soup = BeautifulSoup(response, 'html.parser')
    e1 = ent
    relations = []
    if soup.find('table'):
        tb = soup.find ('table')
        trs = tb.findChildren('tr')
        for tr in trs:
            ths = tr.find_all('th',class_ = "infobox-label" )
            tds = tr.find_all('td',class_ = "infobox-data")
            for t in ths:
                t1 = t.text
                for t in tds:
                    t2 = t.text
                    t2 = t2.replace("\n","")
                    r = (t1,t2)
                    relations.append(r)
    relations = [(e1, r[0], r[1]) for r in relations]
    return relations
```

另外，考虑到百度百科中也存在一些《喜羊羊与灰太狼》的人物关系信息，我们对其进行二次爬取以对三元组关系进行补充。其中，百度百科角色人物网址格式为`"https://baike.baidu.com/item/角色名"`，检查百度百科角色主页源代码，查看人物关系源代码格式如下：

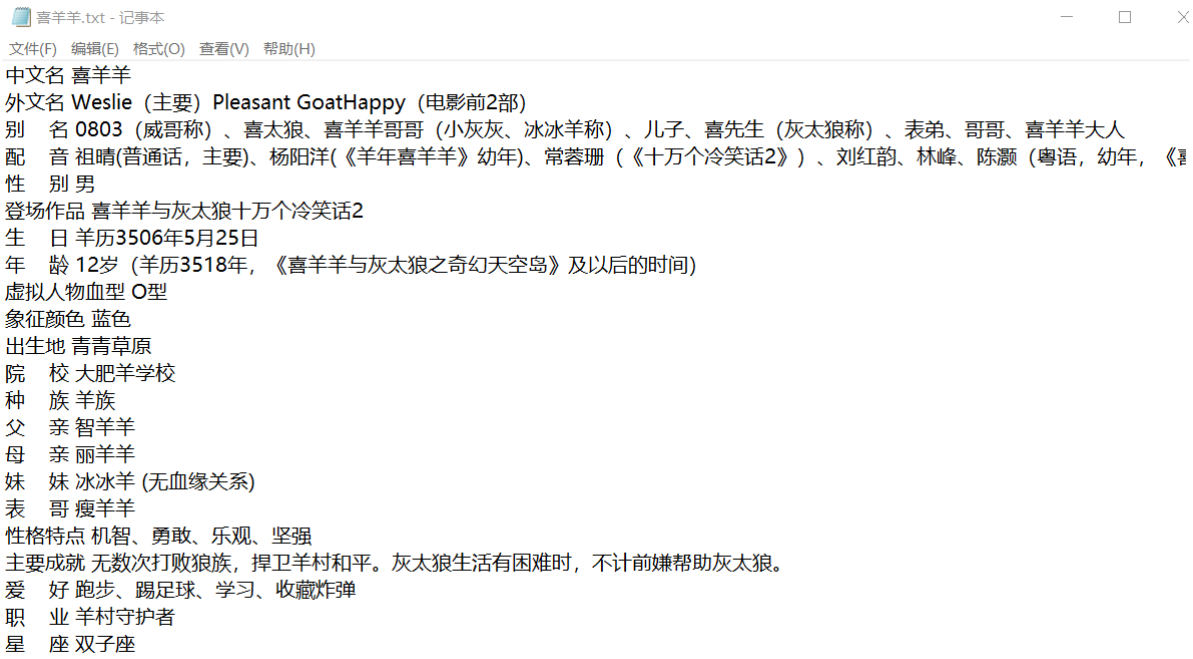


调用 baidu_rec() 函数，对《喜羊羊与灰太狼》中的角色属性进行爬取，并将结果保存到 角色名.txt 文件中，具体代码在 喜羊羊与灰太狼/数据爬取/bdbk_rec.py 目录下。

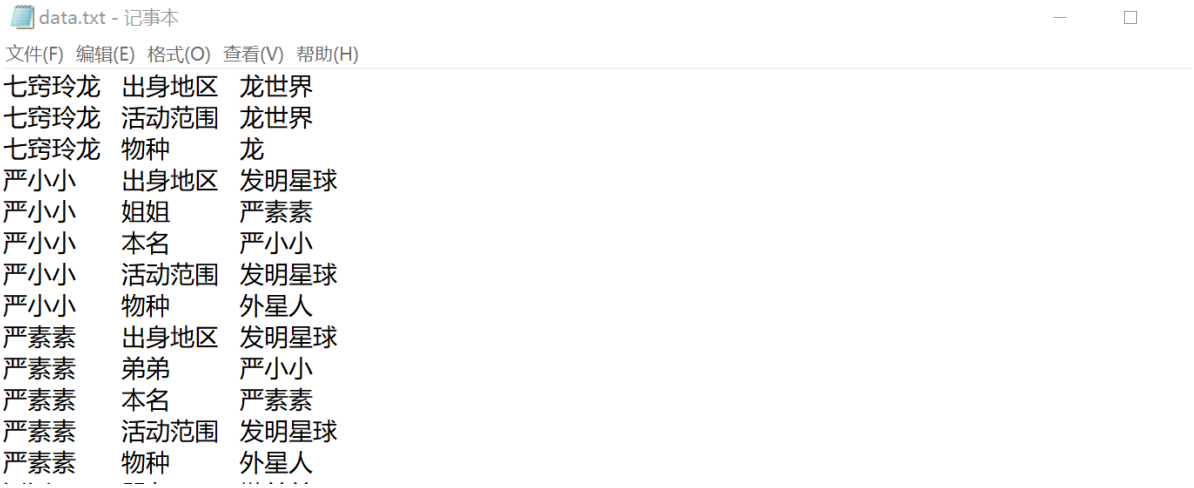
```
def baidu_rec(root,seeds):
    while len(current_category) >= 1 :
        seed = current_category.pop(0)
        print('visiting', seed)
        url = root + seed
        session = HTMLSession()

        response = session.get(url)

        html_code = response.content.decode(response.encoding)
        from lxml import etree
        treeObj = etree.HTML(html_code)
        target4 = treeObj.xpath("string(//dl[@class='basicInfo-block basicInfo-
left'])")
        print(len(target4))
        selector = Selector(text=html_code)
        title = ''.join(selector.xpath('//h1/text()').extract()).replace('/',
'')
        names = selector.xpath('//dt[contains(@class,"basicInfo-item
name")]').extract()
        values = selector.xpath('//dd[contains(@class,"basicInfo-item
value")]').extract()

        lines = ''
        for i, name in enumerate(names) :
            # name
            temp =
Selector(text=name).xpath('//dt/text()|//dt/a/text()').extract()
            name = ''.join(temp).replace('\n', '')
            # value
            temp =
Selector(text=values[i]).xpath('//dd/text()|//dd/a/text()').extract()
            value = ''.join(temp).replace('\n', '')
            lines += name + '$$' + value + '\n'
            print(name, value)
        print('process file:' + str(title))
        output = open(title + '.txt', 'w', encoding='utf-8')
        output.write(lines)
```

```
        output.close()
```

例如，喜羊羊在百度百科爬取关系结果如下：



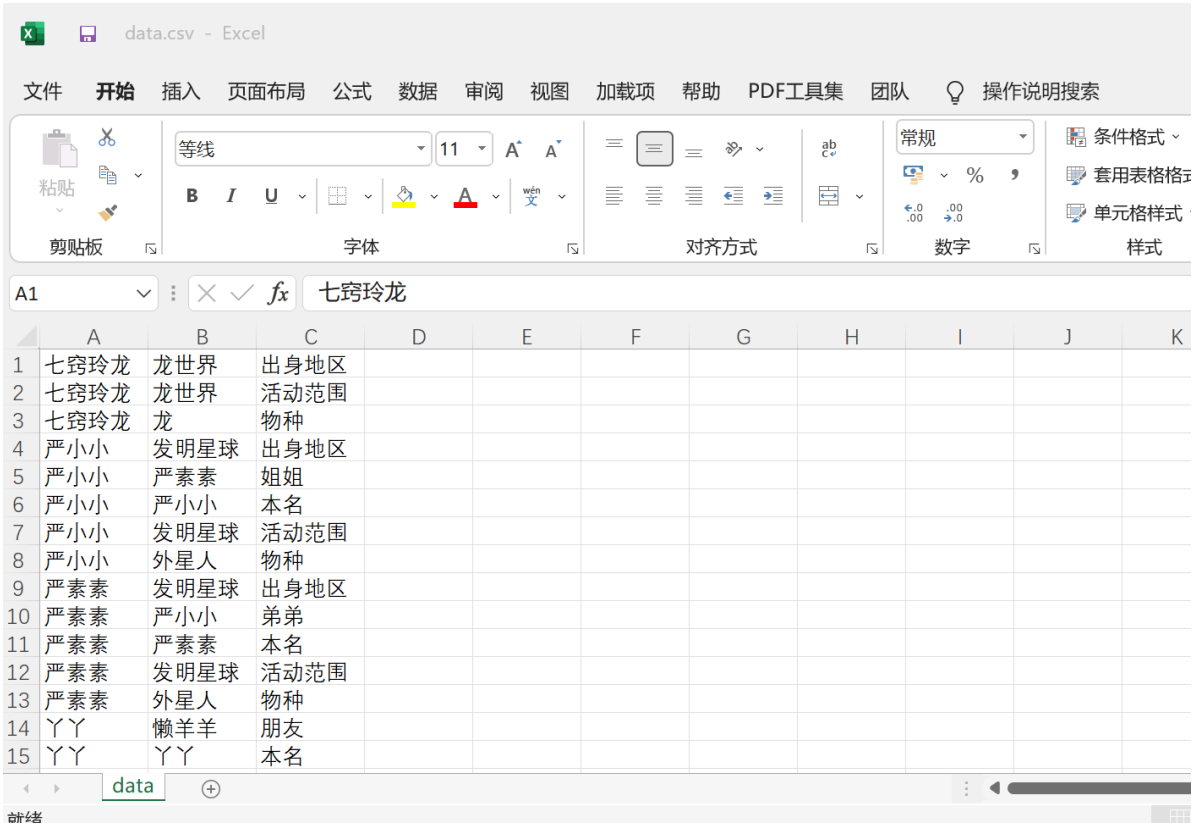
另外，由于爬取数据中存在需要清洗的部分，需要对半结构化数据进行两部分清洗。一方面是，删掉不需要的例如配音演员、发行日期等信息；另一方面，处理不正确的符号，不正确的三元组格式，错别字等。将清洗结束后的最终结果保存在`喜羊羊与灰太狼/数据爬取/data.txt`。



最后，把清洗好的三元组数据输出为csv格式，具体代码在`喜羊羊与灰太狼/数据爬取/tocsv_data.py`。

```
with open(relations_final, 'w', encoding='utf-8') as f:
    for d in data:
        f.write('%s,%s,%s\n'%(d[0],d[1],d[2]))
```

最终关系三元组结果保存至`喜羊羊与灰太狼/数据爬取/data.csv`，部分结果如下所示：



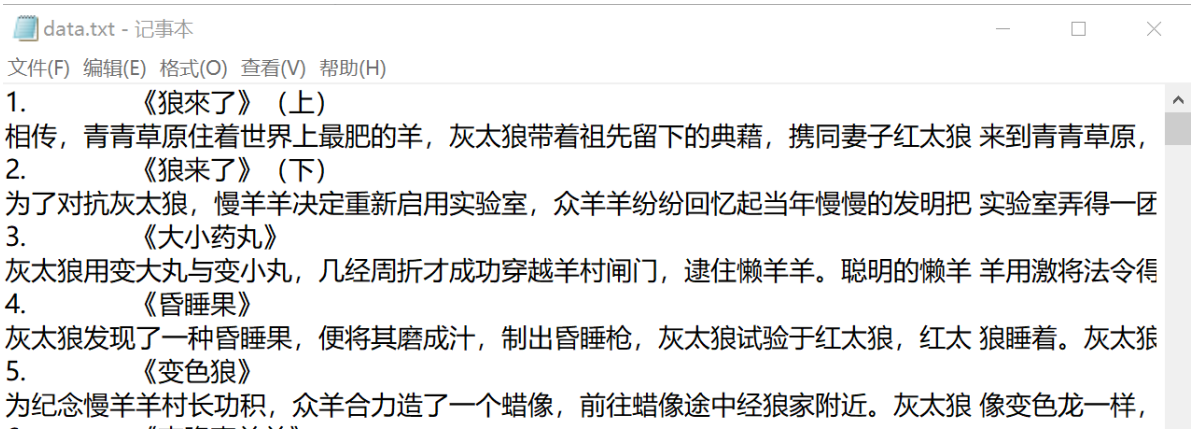
	A	B	C
1	七窍玲龙	龙世界	出身地区
2	七窍玲龙	龙世界	活动范围
3	七窍玲龙	龙	物种
4	严小小	发明星球	出身地区
5	严小小	严素素	姐姐
6	严小小	严小小	本名
7	严小小	发明星球	活动范围
8	严小小	外星人	物种
9	严素素	发明星球	出身地区
10	严素素	严小小	弟弟
11	严素素	严素素	本名
12	严素素	发明星球	活动范围
13	严素素	外星人	物种
14	丫丫	懒羊羊	朋友
15	丫丫	丫丫	本名

1.2 文本数据获取

由于基于深度学习方法抽取人物关系，需要包含实体和关系的文本数据。我们选择百度中《喜羊羊与灰太狼》动漫每一集的剧情介绍（https://wenku.baidu.com/view/2d0edebdac51f01dc281e53a580216fc700a539d.html）作为文本，爬取方式与上文相同，并保存至`喜羊羊与灰太狼/数据爬取/data.txt`中。

由于爬取下来的文本数据存在一些错别字和不正确的符号，例如“灰人狼”，“喜洋洋”，所以先对文本数据进行简单清洗，最后得到

动画片400集数据量的训练数据，结果如下所示。



2 命名实体识别

为了更好把数据中爬取的文本作为关系抽取的输入，我们利用命名实体识别技术对它进行处理，提取出包含两个实体以上的句子作为关系抽取的输入。通过比较`BI-LSTM`模型以及`BI-LSTM+CRF`模型对实体的识别效果，选择识别精确度相对最高的BI-LSTM+CRF模型作为本实验场景的最终实体识别模型。对应项目代码在`喜羊羊与灰太狼/实体识别`目录下。

2.1 训练数据标注

由于现有的NER训练数据都是以人名为主，不适合《喜羊羊与灰太狼》中的动物角色命名，所以需要手动进行数据标注。常见的标注方法有BIO、BIOE、BIOES等，这里我们选择BIOE编码方法。通过标注工具辅助，最终得到训练数据100条，将其输出保存在`喜羊羊与灰太狼/实体识别\entity\train.txt`目录下，部分结果如下：

2.2 基于BI-LSTM的命名实体识别

采用常见的`LSTM`对命名实体进行识别，具体代码为`喜羊羊与灰太狼/实体识别\entity\lstm.py`。

首先，调用`read_data()`和`build_vocab()`函数，构造`word_to_id`和`tag_to_id`两个词典，将训练数据的文字序列和标注序列都转为数字数组，利用pytorch自带的词嵌入对文本序列进行预处理。

```
def read_data():
    sentences = []
    tags = []
    with open('data/train.txt', 'r', encoding='utf-8') as f:
        tmp_sentence = []
        tmp_tags = []
        for line in f:
            if line == '\n' and len(tmp_sentence) != 0:
                assert len(tmp_sentence) == len(tmp_tags)
                sentences.append(tmp_sentence)
                tags.append(tmp_tags)
                tmp_sentence = []
                tmp_tags = []
            elif line == '\n' or line == ' O\n' or line == '     O\n' :
                continue
            else:
                line = line.strip().split(' ')
                tmp_sentence.append(line[0])
                tmp_tags.append(line[1])
        if len(tmp_sentence) != 0:
            assert len(tmp_sentence) == len(tmp_tags)
            sentences.append(tmp_sentence)
            tags.append(tmp_tags)
    return sentences, tags

def build_vocab(sentences):
    global word_to_id
    for sentence in sentences:  # 建立word到索引的映射
```

```
        for word in sentence:
            if word not in word_to_id:
                word_to_id[word] = len(word_to_id)
    return word_to_id
```

其次，通过LSTM层和分类器，输出tag词典大小的数组，得到的数据是某个标识符可能的最大概率，并利用交叉熵和梯度下降法优化，模型框架部分代码如下：

```
class Mymodel(nn.Module):
    def __init__(self,corpus_num,embedding_num,hidden_num,class_num,bi=True):
        super().__init__()
        self.embedding = nn.Embedding(corpus_num,embedding_num)
        self.lstm =
nn.LSTM(embedding_num,hidden_num,batch_first=True,bidirectional=bi)
        if bi :
            self.classifier = nn.Linear(hidden_num * 2,class_num)
        else:
            self.classifier = nn.Linear(hidden_num, class_num)
        self.cross_loss = nn.CrossEntropyLoss()

    def forward(self,batch_data,batch_tag=None):
        embedding = self.embedding(batch_data)
        out,_ = self.lstm(embedding)
        pre = self.classifier(out)
        self.pre = torch.argmax(pre, dim=-1).reshape(-1)
        #找到对应的最大的那个tag
        if batch_tag is not None:
            loss =
self.cross_loss(pre.reshape(-1,pre.shape[-1]),batch_tag.reshape(-1))
            #结果和已知做loss
            return loss
```

但是训练结果并不理想，会出现一些BIO逻辑上的错误，例如以I开头的实体，这显然不符合标注常理。

2.3 基于BI-LSTM+CRF的命名实体识别

由于LSTM模型实体识别效果并不理想，我们引入了概率论中的条件随机场CRF，对原`LSTM`模型进行改进。首先，把双向`LSTM`训练得到的结果作为发射矩阵，CRF作为转移矩阵，然后通过计算每次给定序列得分并找到最大得分路径的方式，最终构建`BI-LSTM+CRF`模型。具体代码在`喜羊羊与灰太狼/实体识别\entity\lstm-crf.py`目录下。

首先，根据CRF的定义，设计辅助函数`log_sum_exp`计算分数，通过调用`log_sum_exp()`函数进行实现：

```
def log_sum_exp(vec):
    max_score = vec[0, argmax(vec)]
    max_score_broadcast = max_score.view(1, -1).expand(1, vec.size()[1])
    return max_score + \
        torch.log(torch.sum(torch.exp(vec - max_score_broadcast)))
```

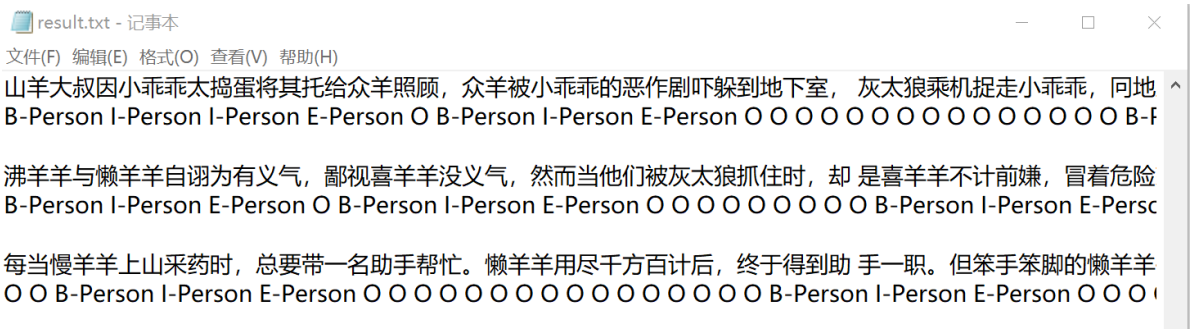
接着，构建`BI-LSTM+CRF`模型，部分代码如下：

```
 def _forward_alg(self, feats):
        init_alphas = torch.full((1, self.tagset_size), -10000.)
```

```
        init_alphas[0][self.tag_to_ix[START_TAG]] = 0.

        forward_var = init_alphas

        for feat in feats:
            alphas_t = []
            for next_tag in range(self.tagset_size):
                emit_score = feat[next_tag].view(
                    1, -1).expand(1, self.tagset_size)
                trans_score = self.transitions[next_tag].view(1, -1)
                next_tag_var = forward_var + trans_score + emit_score
                alphas_t.append(log_sum_exp(next_tag_var).view(1))
            forward_var = torch.cat(alphas_t).view(1, -1)
        terminal_var = forward_var + self.transitions[self.tag_to_ix[STOP_TAG]]
        alpha = log_sum_exp(terminal_var)
        return alpha

    def _get_lstm_features(self, sentence):
        self.hidden = self.init_hidden()
        embeds = self.word_embeds(sentence).view(len(sentence), 1, -1)
        lstm_out, self.hidden = self.lstm(embeds, self.hidden)
        lstm_out = lstm_out.view(len(sentence), self.hidden_dim)
        lstm_feats = self.hidden2tag(lstm_out)
        return lstm_feats

    def _score_sentence(self, feats, tags):

        score = torch.zeros(1)
        tags = torch.cat([torch.tensor([self.tag_to_ix[START_TAG]],
dtype=torch.long), tags])
        for i, feat in enumerate(feats):
            score = score + \
                self.transitions[tags[i + 1], tags[i]] + feat[tags[i + 1]]
        score = score + self.transitions[self.tag_to_ix[STOP_TAG], tags[-1]]
        return score

    def _viterbi_decode(self, feats):
        backpointers = []

        init_vvars = torch.full((1, self.tagset_size), -10000.)
        init_vvars[0][self.tag_to_ix[START_TAG]] = 0

        forward_var = init_vvars
        for feat in feats:
            bptrs_t = []
            viterbivars_t = []

            for next_tag in range(self.tagset_size):
                next_tag_var = forward_var + self.transitions[next_tag]
                best_tag_id = argmax(next_tag_var)
                bptrs_t.append(best_tag_id)
                viterbivars_t.append(next_tag_var[0][best_tag_id].view(1))
            forward_var = (torch.cat(viterbivars_t) + feat).view(1, -1)
            backpointers.append(bptrs_t)

        terminal_var = forward_var + self.transitions[self.tag_to_ix[STOP_TAG]]
        best_tag_id = argmax(terminal_var)
        path_score = terminal_var[0][best_tag_id]
```

```
        best_path = [best_tag_id]
        for bptrs_t in reversed(backpointers):
            best_tag_id = bptrs_t[best_tag_id]
            best_path.append(best_tag_id)
        start = best_path.pop()
        assert start == self.tag_to_ix[START_TAG]
        best_path.reverse()
        return path_score, best_path

    def neg_log_likelihood(self, sentence, tags):
        feats = self._get_lstm_features(sentence)
        forward_score = self._forward_alg(feats)
        gold_score = self._score_sentence(feats, tags)
        return forward_score - gold_score

    def forward(self, sentence):
        lstm_feats = self._get_lstm_features(sentence)

        score, tag_seq = self._viterbi_decode(lstm_feats)
        return score, tag_seq
```

最后，根据训练数据`train.txt`对模型进行训练，并基于维特比解码预测全部数据，得到最终实体识别结果。对数据再次进行清洗后，将文本BIOE编码结果保存至`喜羊羊与灰太狼\实体识别\entity\data\result.txt`中。



3 实体关系抽取

为抽取出文本中的实体关系，首先，基于开源OpenNRE工具包（https://github.com/thunlp/OpenNRE）的数据输入格式，对实体识别后的BIOE编码结果进行格式化处理。其次，调用opennre库，分别训练`wiki80_cnn_softmax`、`wiki80_bert_softmax`、`wiki80_bertentity_softmax`、`tacred_bert_softmax`以及`tacred_bertentity_softmax`，5种关系抽取的预训练模型。接着，调用哈工大，BERT-wwm，中文bert，在20w中文人物关系数据，训练`chinese-BERT-wwm`模型（https://github.com/taorui-plus/OpenNRE）。最后，将《喜羊羊与灰太狼》格式化处理后的数据，输入6种预训练模型进行推理，输出关系抽取结果。对应项目代码在`喜羊羊与灰太狼\关系抽取`目录下。

3.1 输入数据处理

由于OpenNRE数据对文本以及对应实体标注的输入格式如下：

```
`{'text': 'He was the son of Máel Dúin mac Máele Fithrich, and grandson of the high king Áed Uaridnach (died 612).', 'h': {'pos': (18, 46)}, 't': {'pos': (78, 91)}}`
```

因此，需要将实体识别后的数据格式进行转换，运行`喜羊羊与灰太狼\关系抽取/predata.py`，得到处理后的输入数据`new_data`。

```
def read_data():
    sentences = []
    tags = []
    i=-1
    with open('result.txt', 'r', encoding='gbk') as f:
        line = f.readline()                 # 调用文件的 readline()方法
        while line:
            i=i+1

            if len(line)==0 or line=='\n':
                line = f.readline()
                continue
            if i%3==0 or i==0:
                sentences.append(line)
                line = f.readline()
            else:
                tags.append(line)
                line = f.readline()


    return sentences, tags
sentences, tags=read_data()
new_data=[]

for t in range(len(tags)):
    tags[t]=tags[t].replace('-Person','')
    ans=tags[t].replace(' ','')
    #print(len(ans),len(sentences[t]))
    blist=[]
    elist=[]
    for a in re.finditer('B', ans):
        blist.append(a.span())
    for a in re.finditer('E', ans):
        elist.append(a.span())
    if (len(blist)!=len(elist)):
        for k in range(1,len(blist)-1):
            if (blist[k][0]-blist[k-1][1])==0:
                del blist[k-1]
        if (len(blist)!=len(elist)):
            print(blist)
            print(elist)
            print(sentences[t])


    bb=[]
    for kk in range(len(blist)-1):
        aa=[]
        aa.append(blist[kk][0])
        aa.append(elist[kk][0])
        bb.append(aa)
    for ss in range(len(bb)):
        for tt in range(ss+1,len(bb)):
            p1=bb[ss][0]
            q1=bb[ss][1]
```

```
            p2=bb[tt][0]
            q2=bb[tt][1]
            posstr1= '('+str(p1)+','+str(q1)+')'
            posstr2='('+str(p2)+','+str(q2)+')'
            new_data.append({'text':sentences[t].replace('\n',''), 'h':
{'pos':eval(posstr1)}, 't': {'pos':eval(posstr2)}})
```

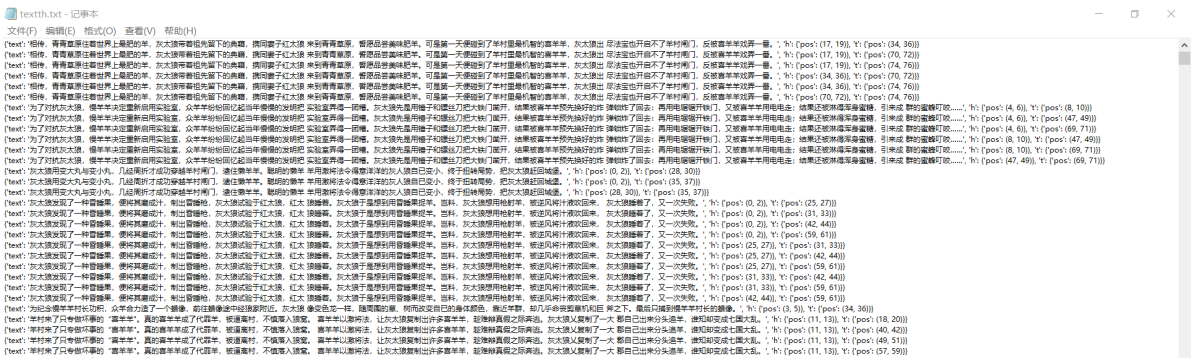
第一次运行后，结果输出如下，发现存在处理异常的数据。

```
[(12, 13), (21, 22), (46, 47), (61, 62), (85, 86)]
[(14, 15), (23, 24), (48, 49), (63, 64), (77, 78), (87, 88)]
新年到，众羊要交换礼物，懒羊羊把口水垫给了暖羊羊，可是他又后悔了，要想尽办 法拿回来，结果被灰太狼捉住了，还当成礼物送给了狼表哥，但很快又后悔了，于是与表哥 争抢起来，让喜羊羊他们有机可乘，把他们弹飞了。
```

根据输出信息，对实体识别结果`result.txt`再次进行检查，发现实体识别将“表哥”标注为”I-Person E-Person"，导致进行上述处理时无法匹配到“表哥”实体的开始。因为此处“表哥”代指“狼表哥”,且已经出现在前面语句中，因此手动将“表哥”实体标注更改为“O O”。然后再次运行上述代码，并将结果写入`喜羊羊与灰太狼\关系抽取/textth.txt`文件中。

```
f = open("textth.txt", "w",errors='ignore')
for line in new_data:
    f.write(str(line))
    f.write('\n')
f.close()
```

打开`textth.txt`文件，数据处理结果如下：



打印其中一条处理结果：

```
{'text': '相传，青青草原住着世界上最肥的羊，灰太狼带着祖先留下的典藉，携同妻子红太狼 来到青青草原，誓愿品尝美味肥羊。可是第一天便碰到了羊村里最机智的喜羊羊，灰太狼出 尽法宝也开启不了羊村闸门，反被喜羊羊戏弄一番。',
 'h': {'pos': (17, 19)},
 't': {'pos': (70, 72)}}
```

3.2 基于opennre的关系抽取

处理完关系抽取模型的输入数据后，接下来通过opennre工具对`wiki80_cnn_softmax`、`wiki80_bert_softmax`、`wiki80_bertentity_softmax`、`tacred_bert_softmax`以及`tacred_bertentity_softmax`，这5种关系抽取模型进行预训练，然后对实验数据进行推理，得到最后的关系抽取结果。

首先，在windows10上安装opennre工具。打开cmd，运行以下指令，安装相关库以及包：

```
git clone https://github.com/thunlp/OpenNRE.git --depth 1
pip install -r requirements.txt
python setup.py install
```

安装完成后，尝试运行测试代码：

```
import opennre
model = opennre.get_model('wiki80_cnn_softmax').cuda()
```

代码报错，显示无法找到`pretrain/glove/glove.6B.50d_word2id.json`文件。这是因为，在代码自动运行`download_glove.sh`文件时，`wget glove.6B.50d_word2id.json`文件链接显示"连接失败"，需要手动查找对应文件的有效下载地址：

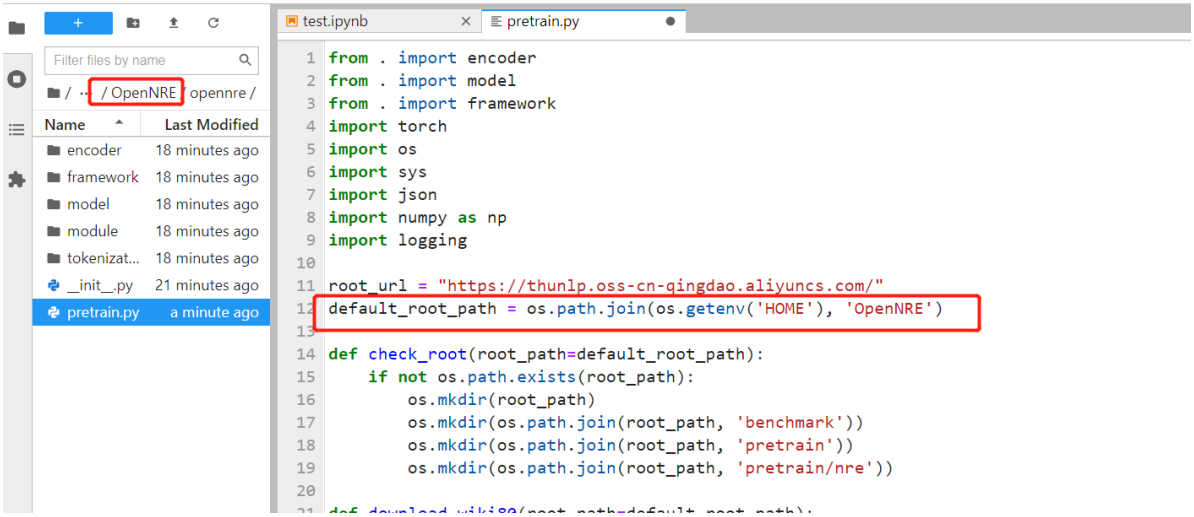
```
https://thunlp.oss-cn-
qingdao.aliyuncs.com/opennre/pretrain/glove/glove.6B.50d_word2id.json
```

保存`glove.6B.50d_word2id.json`文件至`opennre\\pretrain/glove`目录下，同理保存`glove.6B.50d_mat.npy`文件。再次运行，发现仍然找不到对应文件。debug进入`./opennre/pretrain.py`文件，发现引用文件路径需要配置环境变量。

将`pretrain.py`文件中：

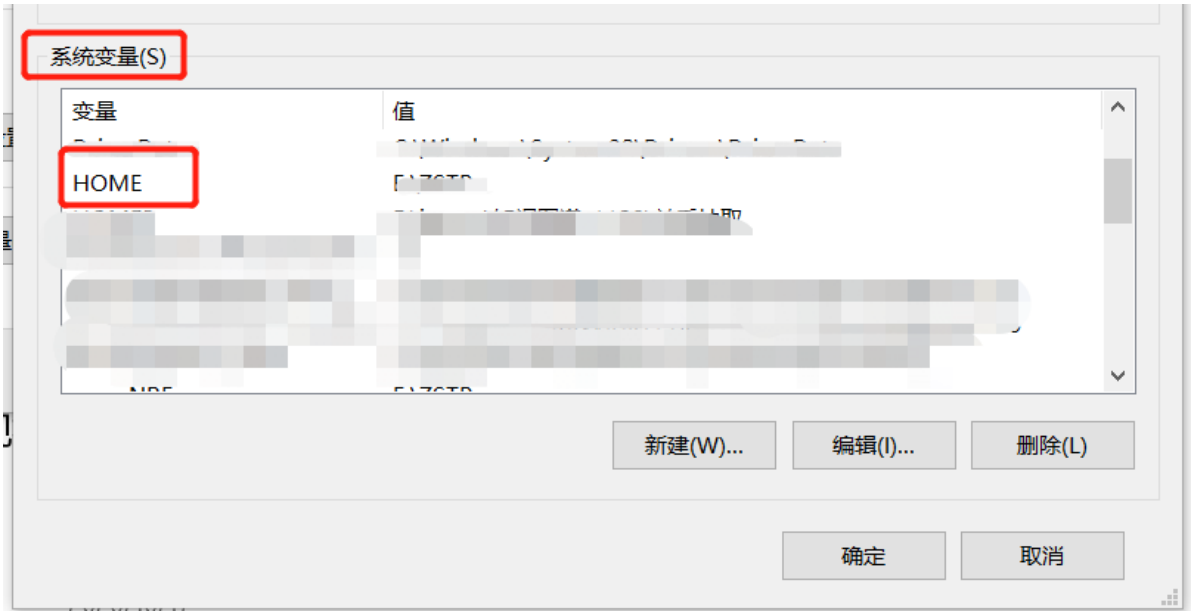
```
default_root_path = os.path.join(os.getenv('HOME'), '.opennre')
```

根据项目名称更改为：




```
default_root_path = os.path.join(os.getenv('HOME'), 'OpenNRE')
```

同时，将`'HOME'`以及对应项目所在路径添加进环境变量。



再次运行测试代码，发现缺少`benchmark/wiki80/wiki80_rel2id.json`、`benchmark/wiki80/wiki80_train.txt`以及`pretrain/nre/wiki80_cnn_softmax.pth.tar`文件，同理找到下载链接，对缺失文件进行下载。

```
https://thunlp.oss-cn-
qingdao.aliyuncs.com/opennre/benchmark/wiki80/wiki80_rel2id.json
https://thunlp.oss-cn-
qingdao.aliyuncs.com/opennre/benchmark/wiki80/wiki80_train.txt
https://thunlp.oss-cn-
qingdao.aliyuncs.com/opennre/pretrain/nre/wiki80_cnn_softmax.pth.tar
```

再次运行测试代码，代码运行成功，输出相应信息：

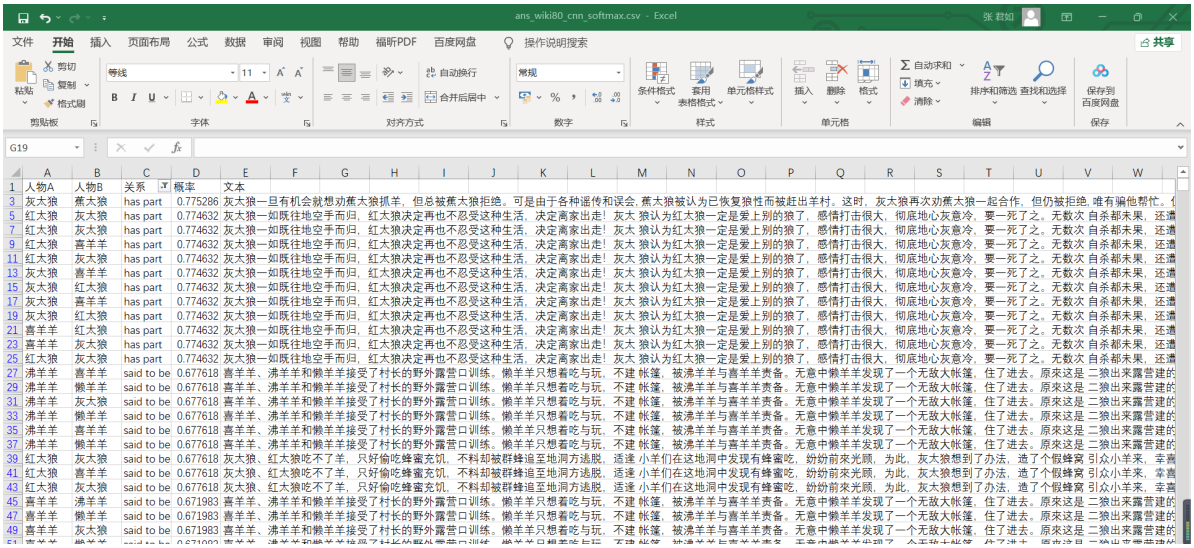
```
2021-11-29 23:59:49,889 - root - INFO - Initializing word embedding with
word2vec.
```

opennre库测试成功后，首先，将刚才调用成功的`wiki80_cnn_softmax`模型，代入`喜羊羊与灰太狼\关系抽取\OpenNRE/relation_train.py`：

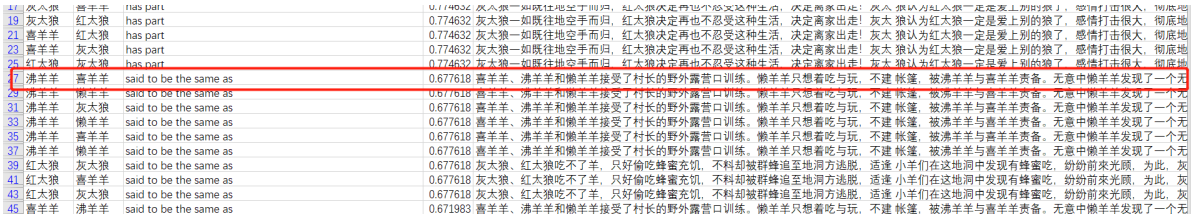
```
import csv
fw=open('ans_wiki80_cnn_softmax.csv',"w")
csv_writer = csv.writer(fw)
csv_writer.writerow(["人物A","人物B","关系","概率","文本"])
with open('textth.txt', 'r', encoding='gb18030',errors='ignore') as f :
    for line in f :
        if line[0].isdigit() :
            continue
        elif len(line) != 0 :
            ss=eval(line)
            #print(ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1], ss['text']
[ss['t']['pos'][0]:ss['t']['pos'][1]+1])
            result = model.infer(ss)
            relation=result[0].encode('gb18030').decode('utf-8')
            #print(relation)
            a=ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1]
            b=ss['text'][ss['t']['pos'][0]:ss['t']['pos'][1]+1]
            if a==b:
                continue
```

```
            else:
                csv_writer.writerow([a, b, relation,result[1],ss['text']])
            print("training")
fw.close()
```

输出`ans_wiki80_cnn_softmax.csv`文件：



从模型运行结果上看，发现关系抽取结果并不理想，例如将“沸羊羊”与“喜羊羊”识别为同一个人物：



出现这样的结果，考虑是由于`wiki80_cnn_softmax`模型是基于wiki80数据集，使用 CNN 编码器对数据集进行训练，而wiki80数据集主要是针对英文语句的词向量间关系进行抽取的，因此并不太适用于中文文本，不能很好地解析中文文本的语义信息。另外，由于CNN模型很难根据训练的结果去针对性的调整具体的特征，大多只能根据词向量之间的距离对实体关系进行推断，因此导致对于本实验场景的关系抽取结果并不理想。

接着，尝试调用`wiki80_bert_softmax`模型：

```
model = opennre.get_model('wiki80_bert_softmax').cuda()
```

报错缺少相关配置，下载`bert`模型，并配置至`pretrain/bert-base-uncased`。

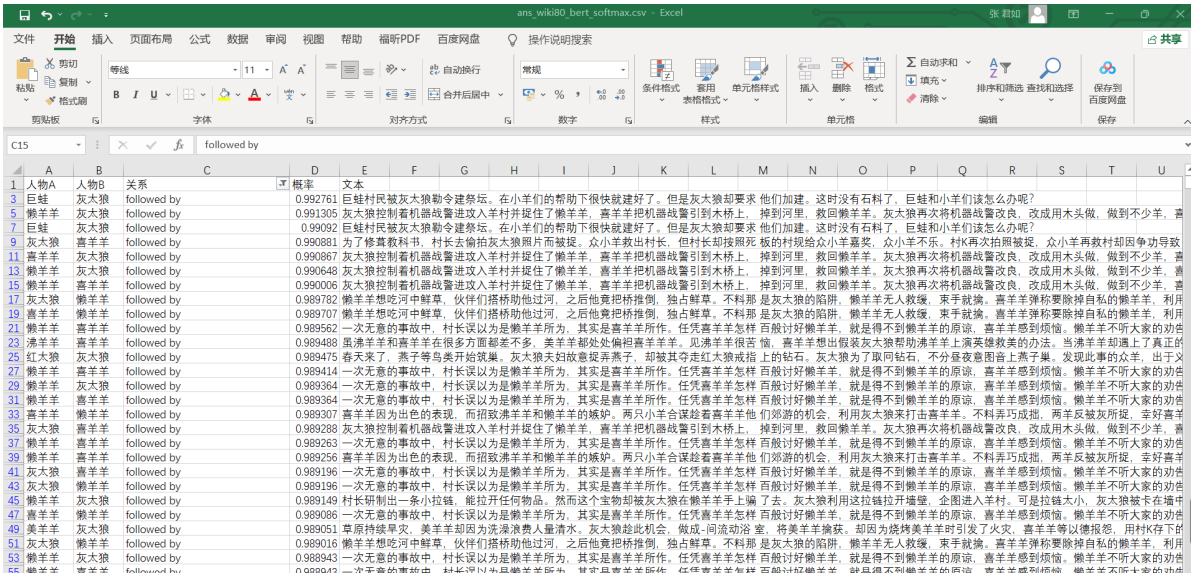
```
https://thunlp.oss-cn-qingdao.aliyuncs.com/opennre/pretrain/bert-base-uncased/config.json
https://thunlp.oss-cn-qingdao.aliyuncs.com/opennre/pretrain/bert-base-uncased/pytorch_model.bin
https://thunlp.oss-cn-qingdao.aliyuncs.com/opennre/pretrain/bert-base-uncased/vocab.txt
```

再次调用`wiki80_bert_softmax`模型，对实验数据进行关系抽取：

```
import opennre
model = opennre.get_model('wiki80_bert_softmax').cuda()
import csv
fw=open('ans_wiki80_bert_softmax.csv',"w")
```

```
csv_writer = csv.writer(fw)
csv_writer.writerow(["人物A","人物B","关系","概率","文本"])
with open('textth.txt', 'r', encoding='gb18030',errors='ignore') as f :
    for line in f :
        if line[0].isdigit() :
            continue
        elif len(line) != 0 :
            ss=eval(line)
            #print(ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1], ss['text']
[ss['t']['pos'][0]:ss['t']['pos'][1]+1])
            result = model.infer(ss)
            relation=result[0].encode('gb18030').decode('utf-8')
            #print(relation)
            a=ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1]
            b=ss['text'][ss['t']['pos'][0]:ss['t']['pos'][1]+1]
            if a==b:
                continue
            else:
                csv_writer.writerow([a, b, relation,result[1],ss['text']])
            print("training")
fw.close()
```

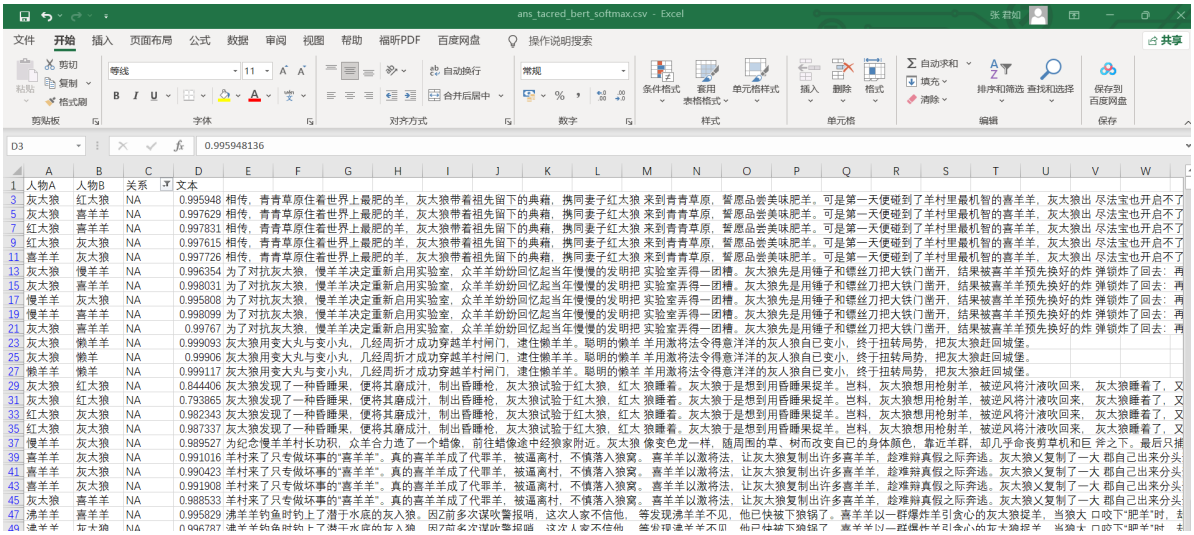
输出`ans_wiki80_bert_softmax.csv`文件：



人物A	人物B	关系	概率	文本
巨蛙	灰太狼	followed by	0.992761	巨蛙村民被灰太狼勒令建祭坛。在小羊们的帮助下很快就建好了。但是灰太狼却要求 他们加建。这时没有石料了，巨蛙和小羊们该怎么办呢？
懒羊羊	灰太狼	followed by	0.991305	灰太狼控制着机器战警进攻入羊村并捉住了懒羊羊，喜羊羊把机器战警引到木桥上， 掉到河里，救回懒羊羊。灰太狼再次将机器战警改良，改成用木头做，做到不少羊，喜
巨蛙	灰太狼	followed by	0.99092	巨蛙村民被灰太狼勒令建祭坛。在小羊们的帮助下很快就建好了。但是灰太狼却要求 他们加建。这时没有石料了，巨蛙和小羊们该怎么办呢？
灰太狼	喜羊羊	followed by	0.990881	为了修真教科书，村长去偷拍灰太狼照片而被捉。众小羊救出村长，但村长却按照死 板的村规给众小羊嘉奖，众小羊不乐。村K再次拍照被捉，众小羊再救村却因争功导致
喜羊羊	灰太狼	followed by	0.990867	灰太狼控制着机器战警进攻入羊村并捉住了懒羊羊，喜羊羊把机器战警引到木桥上， 掉到河里，救回懒羊羊。灰太狼再次将机器战警改良，改成用木头做，做到不少羊，喜
懒羊羊	灰太狼	followed by	0.990648	灰太狼控制着机器战警进攻入羊村并捉住了懒羊羊，喜羊羊把机器战警引到木桥上， 掉到河里，救回懒羊羊。灰太狼再次将机器战警改良，改成用木头做，做到不少羊，喜
懒羊羊	喜羊羊	followed by	0.990006	灰太狼控制着机器战警进攻入羊村并捉住了懒羊羊，喜羊羊把机器战警引到木桥上， 掉到河里，救回懒羊羊。灰太狼再次将机器战警改良，改成用木头做，做到不少羊，喜
灰太狼	懒羊羊	followed by	0.989782	懒羊羊想吃河中鲜草，伙伴们搭桥助他过河，之后他竟把桥推倒，独占鲜草。不料那 是灰太狼的陷阱，懒羊羊无人救援，束手就擒。喜羊羊弹称要除掉自私的懒羊羊，利用
喜羊羊	懒羊羊	followed by	0.989707	懒羊羊想吃河中鲜草，伙伴们搭桥助他过河，之后他竟把桥推倒，独占鲜草。不料那 是灰太狼的陷阱，懒羊羊无人救援，束手就擒。喜羊羊弹称要除掉自私的懒羊羊，利用
懒羊羊	喜羊羊	followed by	0.989562	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
沸羊羊	喜羊羊	followed by	0.989488	虽沸羊羊和喜羊羊在很多方面都差不多，美羊羊都处处偏袒喜羊羊羊。见沸羊羊很苦 恼，喜羊羊想出假装灰太狼帮助沸羊羊上演英雄救美的办法。当沸羊羊却遇上了真正的
红太狼	灰太狼	followed by	0.989475	春天来了，燕子等鸟类开始筑巢。灰太狼夫妇故意捉弄燕子，却被其夺走红太狼戒指 上的钻石。灰太狼为了取回钻石，不分昼夜意图音上燕子巢。发现此事的众羊，出于义
懒羊羊	喜羊羊	followed by	0.989414	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
懒羊羊	灰太狼	followed by	0.989364	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
懒羊羊	喜羊羊	followed by	0.989364	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
喜羊羊	懒羊羊	followed by	0.989307	喜羊羊因为出色的表现，而招致沸羊羊和懒羊羊的嫉妒。两只小羊合谋趁着喜羊羊他 们郊游的机会，利用灰太狼来打击喜羊羊。不料弄巧成拙，两羊反被灰所捉，幸好喜羊
灰太狼	喜羊羊	followed by	0.989288	灰太狼控制着机器战警进攻入羊村并捉住了懒羊羊，喜羊羊把机器战警引到木桥上， 掉到河里，救回懒羊羊。灰太狼再次将机器战警改良，改成用木头做，做到不少羊，喜
懒羊羊	喜羊羊	followed by	0.989263	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
懒羊羊	喜羊羊	followed by	0.989256	喜羊羊因为出色的表现，而招致沸羊羊和懒羊羊的嫉妒。两只小羊合谋趁着喜羊羊他 们郊游的机会，利用灰太狼来打击喜羊羊。不料弄巧成拙，两羊反被灰所捉，幸好喜羊
灰太狼	喜羊羊	followed by	0.989196	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
灰太狼	懒羊羊	followed by	0.989196	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
懒羊羊	灰太狼	followed by	0.989149	村长研制出一条小拉链，能拉开任何物品。然而这个宝物却被灰太狼在懒羊羊手上骗 了去。灰太狼利用这拉链拉开墙壁，企图进入羊村。可是拉链太小，灰太狼被卡在墙中
喜羊羊	懒羊羊	followed by	0.989086	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告
美羊羊	灰太狼	followed by	0.989051	草原持续旱灾，美羊羊却因为洗澡浪费人量清水。灰太狼趁此机会，做成-间流动浴 室，将美羊羊擒获。却因为烧烤美羊羊时引发了火灾，喜羊羊等以德报怨，用村K存下的
灰太狼	懒羊羊	followed by	0.989016	懒羊羊想吃河中鲜草，伙伴们搭桥助他过河，之后他竟把桥推倒，独占鲜草。不料那 是灰太狼的陷阱，懒羊羊无人救援，束手就擒。喜羊羊弹称要除掉自私的懒羊羊，利用
懒羊羊	灰太狼	followed by	0.988943	一次无意的事故中，村长误以为是懒羊羊所为，其实是喜羊羊所作。任凭喜羊羊怎样 百般讨好懒羊羊，就是得不到懒羊羊的原谅，喜羊羊感到烦恼。懒羊羊不听大家的劝告

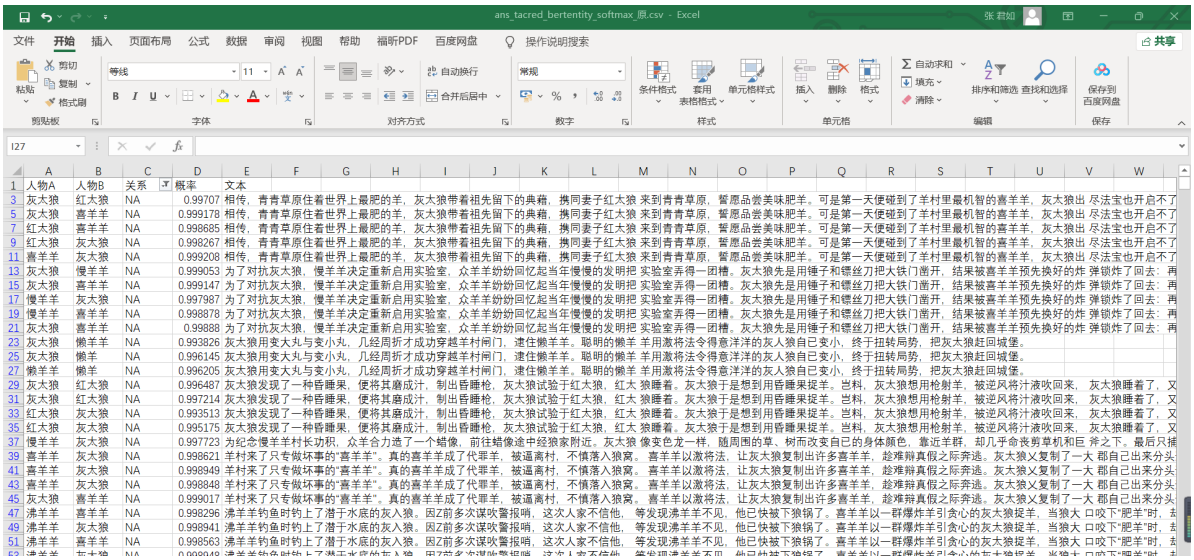
从抽取结果上看，由于`wiki80_bert_softmax`模型是基于wiki80数据集，使用 BERT 编码器对数据集进行训练，相对于`wiki80_cnn_softmax`模型有所提升，但是对于人物关系的抽取并不是很理想，考虑原因仍是和训练数据集的数据特点有关。

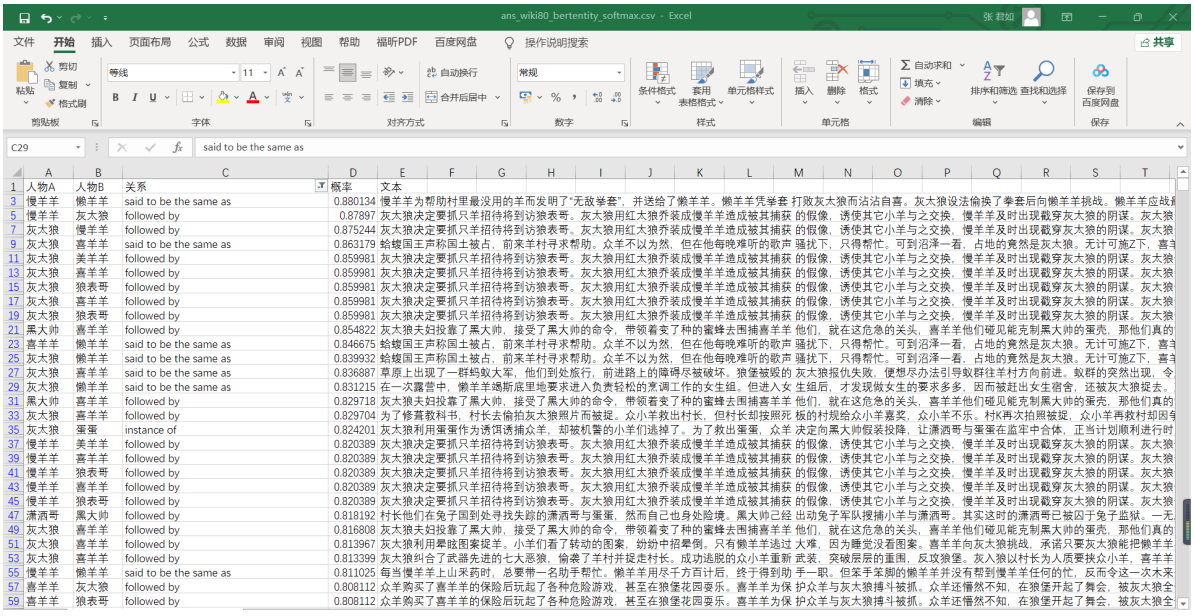
同理，调用`tacred_bert_softmax`模型，并下载相关配置文件，输出`ans_tacred_bert_softmax.csv`文件结果如下：



从抽取结果上看，`tacred_bert_softmax`模型是基于TACRED数据集，使用 BERT 编码器对数据集进行训练。对于本实验场景，关系抽取的输出结果大多是`'NA'`。考虑到这是因为TACRED数据集主要涉及的是英语新闻通讯社和网络文本，所以无法对中文的人物关系以及动画场景做出识别。

同理，调用`tacred_bertentity_softmax`模型以及`wiki80_bertentity_softmax`模型，并下载相关配置文件，输出`ans_tacred_bert_softmax.csv`以及`ans_wiki80_bertentity_softmax.csv`文件结果分别如下：





通过比较发现，从训练数据集上来看，wiki80数据集相比于TACRED数据集训练效果更好，从训练模型上看，使用 BERT 编码器相比于CNN编码器，在人物关系抽取上的训练效果更好。但是，通过对这5中模型结果的筛选和比较，一方面发现关系抽取的结果并没有最开始爬虫的关系抽取结果好，另一方面从关系抽取以及关系置信度上看，抽取的结果中并没有筛选到可以补充添加的人物关系。

3.3 基于chinese-bert-wwm的关系抽取

考虑到英文数据集对模型训练效果的影响，接下来采用20w中文人物关系数据对`chinese-bert-wwm`模型进行训练，对应项目代码放置在`喜羊羊与灰太狼\关系抽取/OpenNRE-master`中。

首先，下载中文`bert`模型：

```
git clone https://github.com/taorui-plus/OpenNRE.git
```

接着，在`OpenNRE-master/pretrain`目录下放置`chinese_wwm_pytorch`模型，其下载地址为https://github.com/ymcui/Chinese-BERT-wwm，打开`example/train_people_chinese_bert_softmax.py`，将代码中`'test_chinese'`替换为`'people-relation'`，并将文件打开方式中添加`encoding='gb18030',errors='ignore'`，修改训练参数后运行，最后保存模型参数至`ckpt/test_chinese_bert_softmax4.pth.tar`。生成中文bert预训练模型参数代码部分如下：

```
parser = argparse.ArgumentParser()
parser.add_argument('--mask_entity', action='store_true', help='Mask entity
mentions')
args = parser.parse_args()

# Some basic settings
root_path = '.'
sys.path.append(root_path)
if not os.path.exists('ckpt'):
    os.mkdir('ckpt')
ckpt = 'ckpt/test_chinese_bert_softmax.pth.tar'

# Check data
rel2id = json.load(open('E:\ZSTP\OpenNRE-
master/benchmark/test_chinese/test_chinese_rel2id.json',encoding= 'gb18030',
errors= 'ignore'))

#Define the sentence encoder
sentence_encoder = opennre.encoder.BERTEncoder(
    max_length=80,

    #pretrain_path=os.path.join(root_path, 'pretrain/chinese-bert-www'),
    pretrain_path='E:\ZSTP\OpenNRE-master\pretrain\chinese-bert-wwm',
    mask_entity=args.mask_entity
)

# Define the model
model = opennre.model.SoftmaxNN(sentence_encoder, len(rel2id), rel2id)


# Define the whole training framework
framework = opennre.framework.SentenceRE(
    train_path='E:\ZSTP\OpenNRE-master/benchmark/test_chinese/people-
relation_train.txt',
```

```
    val_path='E:\ZSTP\OpenNRE-master/benchmark/test_chinese/people-
relation_val.txt',
    test_path='E:\ZSTP\OpenNRE-master/benchmark/test_chinese/people-
relation_val.txt',
    model=model,
    ckpt=ckpt,
    batch_size=16, # Modify the batch size w.r.t. your device
    max_epoch=4,
    lr=2e-5,
    opt='adamw'
)

# Train the model
framework.train_model()

# Test the model
framework.load_state_dict(torch.load(ckpt)['state_dict'])
result = framework.eval_model(framework.test_loader)

# Print the result
print('Accuracy on test set: {}'.format(result['acc']))
```

训练结果如下：

```
=== Epoch 0 train ===
100%|██████████| 12375/12375 [1:29:22<00:00,  2.31it/s, acc=0.559, loss=1.33]
=== Epoch 0 val ===
100%|██████████| 63/63 [00:10<00:00,  5.88it/s, acc=0.704]
Best ckpt and saved.
=== Epoch 1 train ===
100%|██████████| 12375/12375 [1:20:33<00:00,  2.56it/s, acc=0.704, loss=0.87]
=== Epoch 1 val ===
100%|██████████| 63/63 [00:10<00:00,  5.89it/s, acc=0.823]
Best ckpt and saved.
=== Epoch 2 train ===
100%|██████████| 12375/12375 [1:20:30<00:00,  2.56it/s, acc=0.791, loss=0.607]
=== Epoch 2 val ===
100%|██████████| 63/63 [00:10<00:00,  5.86it/s, acc=0.886]
Best ckpt and saved.
=== Epoch 3 train ===
100%|██████████| 12375/12375 [1:20:29<00:00,  2.56it/s, acc=0.847, loss=0.442]
=== Epoch 3 val ===
100%|██████████| 63/63 [00:10<00:00,  5.87it/s, acc=0.919]
Best ckpt and saved.
Best acc on val set: 0.919000
100%|██████████| 63/63 [00:10<00:00,  5.88it/s, acc=0.919]
Accuracy on test set: 0.919

Process finished with exit code 0
```

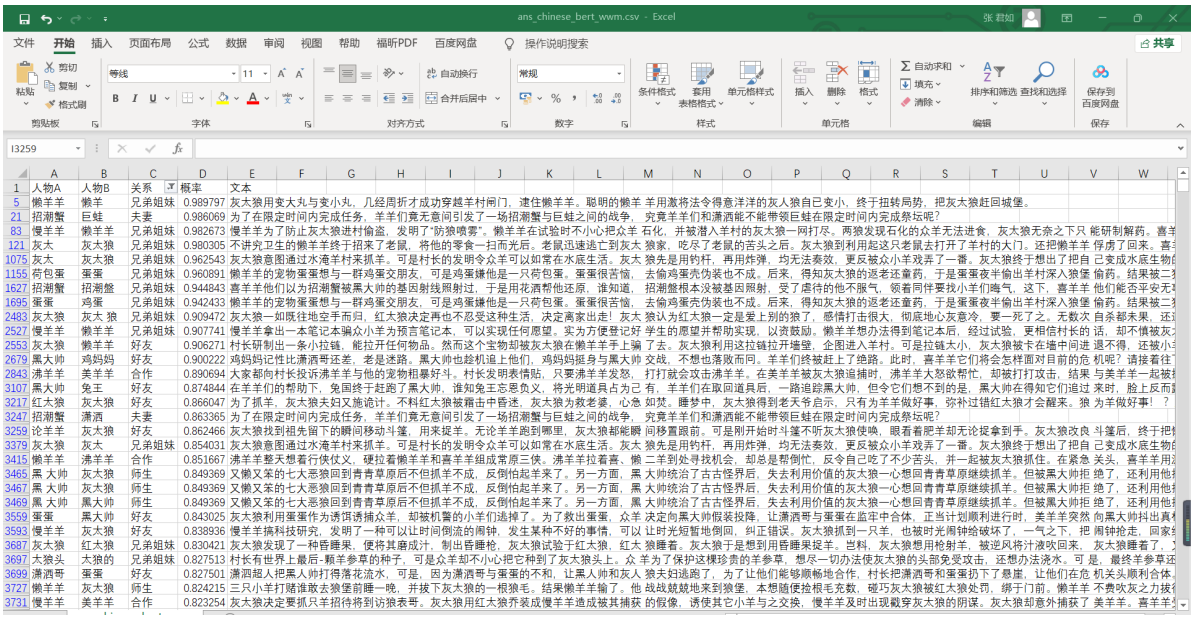
最终`chinese-bert-wwm`模型ACC达到0.919。

将模型预训练参数`test_chinese_bert_softmax.pth.tar`加载至`chinese-bert-wwm`的`framework.model`中，对实验数据进行关系抽取，并将预测结果按照`["人物A","人物B","关系","概率","文本"]`格式输出至`ans_chinese_bert_wwm.csv`中。对应代码在`喜羊羊与灰太狼\关系抽取\OpenNRE-master\example/predict_xyy_chinese_bert.py`文件中。

```
# encoding:gbk

parser = argparse.ArgumentParser()
parser.add_argument('--mask_entity', action='store_true', help='Mask entity
mentions')
args = parser.parse_args()

# Some basic settings
root_path = '.'
sys.path.append(root_path)
if not os.path.exists('ckpt'):
    os.mkdir('ckpt')
ckpt = 'ckpt/test_chinese_bert_softmax3.pth.tar'

# Check data
rel2id = json.load(open('OpenNRE-
master/benchmark/test_chinese/test_chinese_rel2id.json',encoding= 'gb18030',
errors= 'ignore'))

#Define the sentence encoder
sentence_encoder = opennre.encoder.BERTEncoder(
    max_length=80,

    #pretrain_path=os.path.join(root_path, 'pretrain/chinese-bert-www'),
    pretrain_path='OpenNRE-master\pretrain\chinese-bert-wwm',
    mask_entity=args.mask_entity
)

# Define the model
model = opennre.model.SoftmaxNN(sentence_encoder, len(rel2id), rel2id).cuda()


framework = opennre.framework.SentenceRE(
    train_path='OpenNRE-master/benchmark/test_chinese/test_chinese_train.txt',
    val_path='OpenNRE-master/benchmark/test_chinese/test_chinese_val.txt',
    test_path='OpenNRE-master/benchmark/test_chinese/test_chinese_val.txt',
    model=model,
    ckpt=ckpt,
    batch_size=16, # Modify the batch size w.r.t. your device
    max_epoch=7,
    lr=2e-5,
    opt='adamw'
)
framework.load_state_dict(torch.load(ckpt)['state_dict'])
import csv
fw=open('ans_chinese_bert_wwm.csv',"w")
csv_writer = csv.writer(fw)
csv_writer.writerow(["人物A","人物B","关系","概率","文本"])
with open('textth.txt', 'r', encoding='gb18030',errors='ignore') as f :
    for line in f :
        if line[0].isdigit() :
            continue
```

```
        elif len(line) != 0 :
            ss=eval(line)
            #print(ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1], ss['text']
[ss['t']['pos'][0]:ss['t']['pos'][1]+1])
            result = framework.model.infer(ss)
            relation=result[0].encode('gb18030').decode('utf-8')
            #print(relation)
            a=ss['text'][ss['h']['pos'][0]:ss['h']['pos'][1]+1]
            b=ss['text'][ss['t']['pos'][0]:ss['t']['pos'][1]+1]
            if a==b:
                continue
            else:
                csv_writer.writerow([a, b, relation,result[1],ss['text']])
fw.close()
```

`ans_chinese_bert_wwm.csv`输出结果如下：



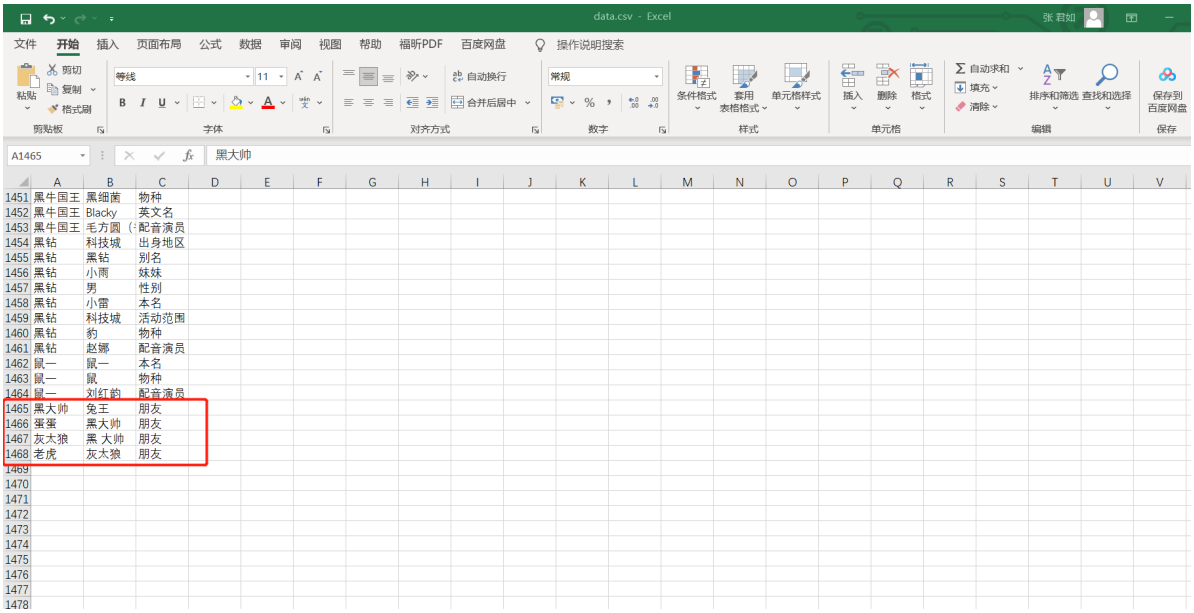
从结果上看，关系抽取结果存在以下两个问题：

第一，由于文本本身数据清洗不全面，导致“灰太”,"招潮盘"等词语滞留在输出文件中。

第二，由于模型训练数据的关系是：

```
{"父母":0, "夫妻":1, "师生":2, "兄弟姐妹":3, "合作":4, "情侣":5, "祖孙":6 ,"好友":7,
"亲戚":8, "同门":9, "上下级":10, "unknown":11}
```

即模型训练时未涉及“敌人“关系，导致将一些敌对关系，例如”黑大帅“与”兔王“，”老虎“与”灰太狼“的关系识别为”好友"关系。在这里，为了更全面展现人物之间的关系连接，本实验场景将“敌人"关系划为”好友"关系。接着，将有效关系的关系置信区间设置为0.8，发现符合的输出结果有38个，然后对原本爬虫得到的`data.csv`中没有的三元组关系进行人为补充，并将”好友“关系的描述统一为"朋友"。共添加了4条关系三元组，补充结果如下：



	A	B	C
1451	黑牛国王	黑细菌	物种
1452	黑牛国王	Blacky	英文名
1453	黑牛国王	毛方圆	配音演员
1454	黑钻	科技城	出身地区
1455	黑钻	黑钻	别名
1456	黑钻	小雨	妹妹
1457	黑钻	男	性别
1458	黑钻	小雷	本名
1459	黑钻	科技城	活动范围
1460	黑钻	豹	物种
1461	黑钻	赵娜	配音演员
1462	鼠一	鼠一	本名
1463	鼠一	鼠	物种
1464	鼠一	刘红韵	配音演员
1465	黑大帅	兔王	朋友
1466	蛋蛋	黑大帅	朋友
1467	灰太狼	黑 大帅	朋友
1468	老虎	灰太狼	朋友

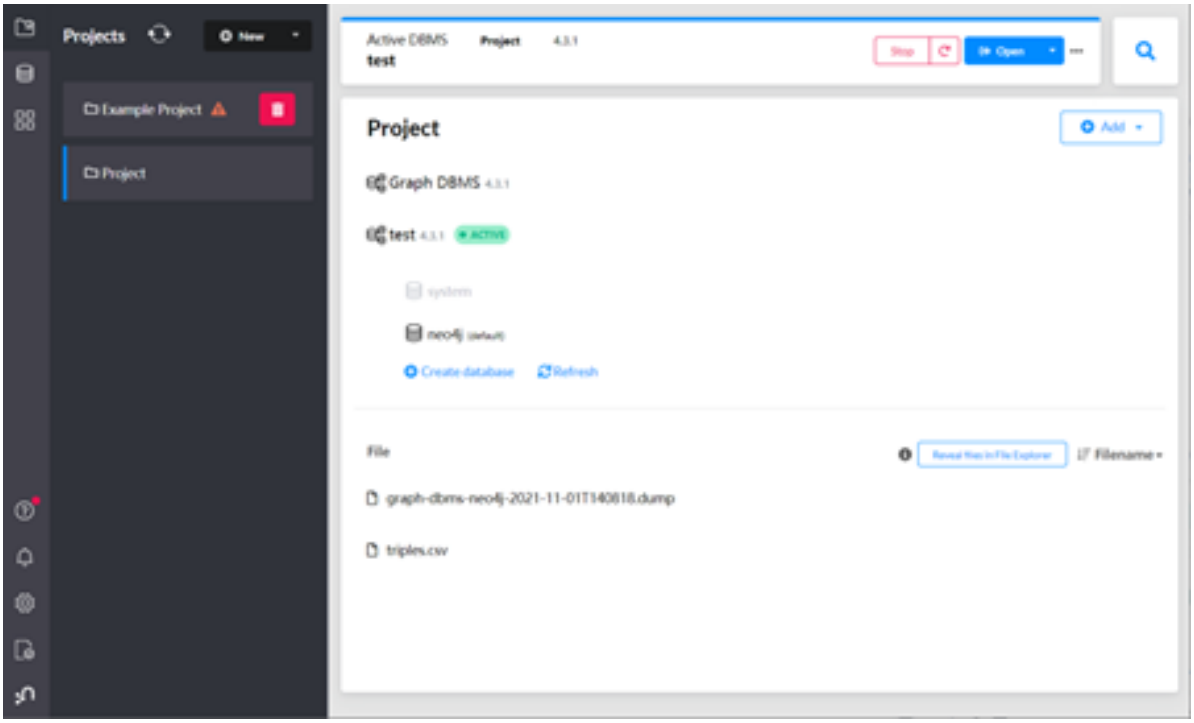
最后，基于6种模型的关系抽取训练与推理完毕。从实验结果上看，基于`chinese-bert-www`模型的关系抽取相比于另外5种模型，准确率有较为明显的提升，同时抽取结果更符合实验文本的语义场景。

4 可视化与知识问答

通过 Neo4j Deskto，对《喜羊羊与灰太狼》的关系抽取结果进行可视化呈现，并实现QA。对应项目代码在`喜羊羊与灰太狼\可视化与知识问答`目录下。

4.1 可视化

首先，下载 Neo4j Deskto ，解压安装，创建 test 数据库并激活运行：



接着，将`relation.txt`三元组导入 Neo4j Browser，进行可视化，具体代码在`喜羊羊与灰太狼\可视化与知识问答/neo.py`目录下。

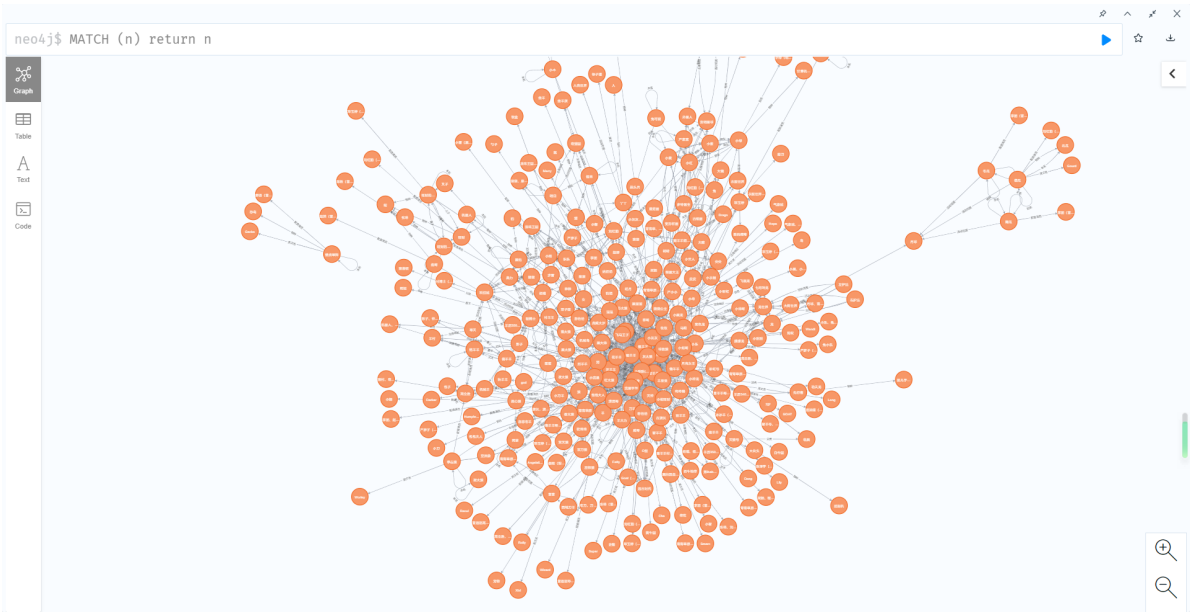
```
class NeoDB(object):
    def __init__(self):
```

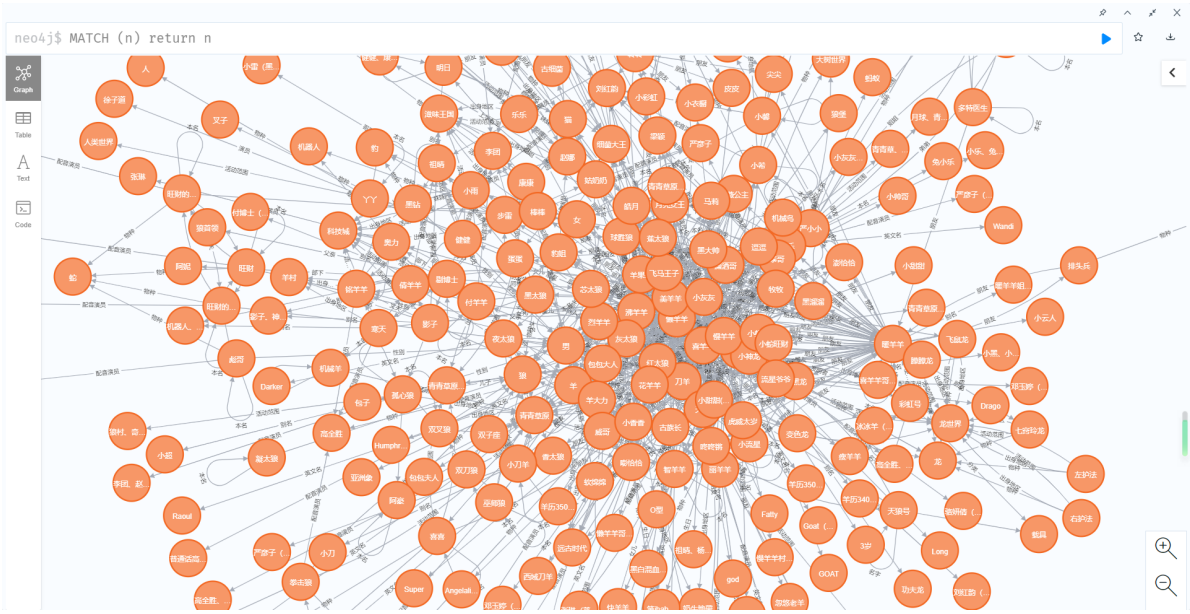
```
        self.graph = Graph("http://localhost:7474",
                           user="neo4j", password="test")
        self.CA_LIST = {"青青草原": 0, "青青草原": 1,
                        "远古时代": 2, "龙世界": 3, "海底": 4, "其他": 5, "冰极": 6}
        self.similar_words = {
            "爸爸": "父亲", "妈妈": "母亲", "爸": "父亲", "妈": "母亲", "朋友": "好朋
友",
            "哥": "哥哥"
        }

    def create_graph(self, relation_filename="data/relation.txt"):
        g = Graph('http://localhost:7474',user='neo4j',password='test')
        with open(relation_filename,'r',encoding='utf-8') as f:
            for line in f.readlines():
                rela_array = line.strip("\n").split(",")
                print(rela_array)
                self.graph.run("MERGE(p: Person{cate:'%s',Name: '%s'})" % (
                    rela_array[3], rela_array[0]))
                self.graph.run("MERGE(p: Person{cate:'%s',Name: '%s'})" % (
                    rela_array[4], rela_array[1]))
                self.graph.run(
                    "MATCH(e: Person), (cc: Person) \
                    WHERE e.Name='%s' AND cc.Name='%s'\
                    CREATE(e)-[r:%s{relation: '%s'}]->(cc)\
                    RETURN r" % (rela_array[0], rela_array[1], rela_array[2],
rela_array[2])
                )
        return

if __name__ == "__main__":
    neo_db = NeoDB()
    neo_db.create_graph("./data/relation.txt")
```

最后，（1）可视化整张图谱。在 Neo4j Browser 端输入：

```
MATCH (n) retu
```

完整图谱展现结果如下：

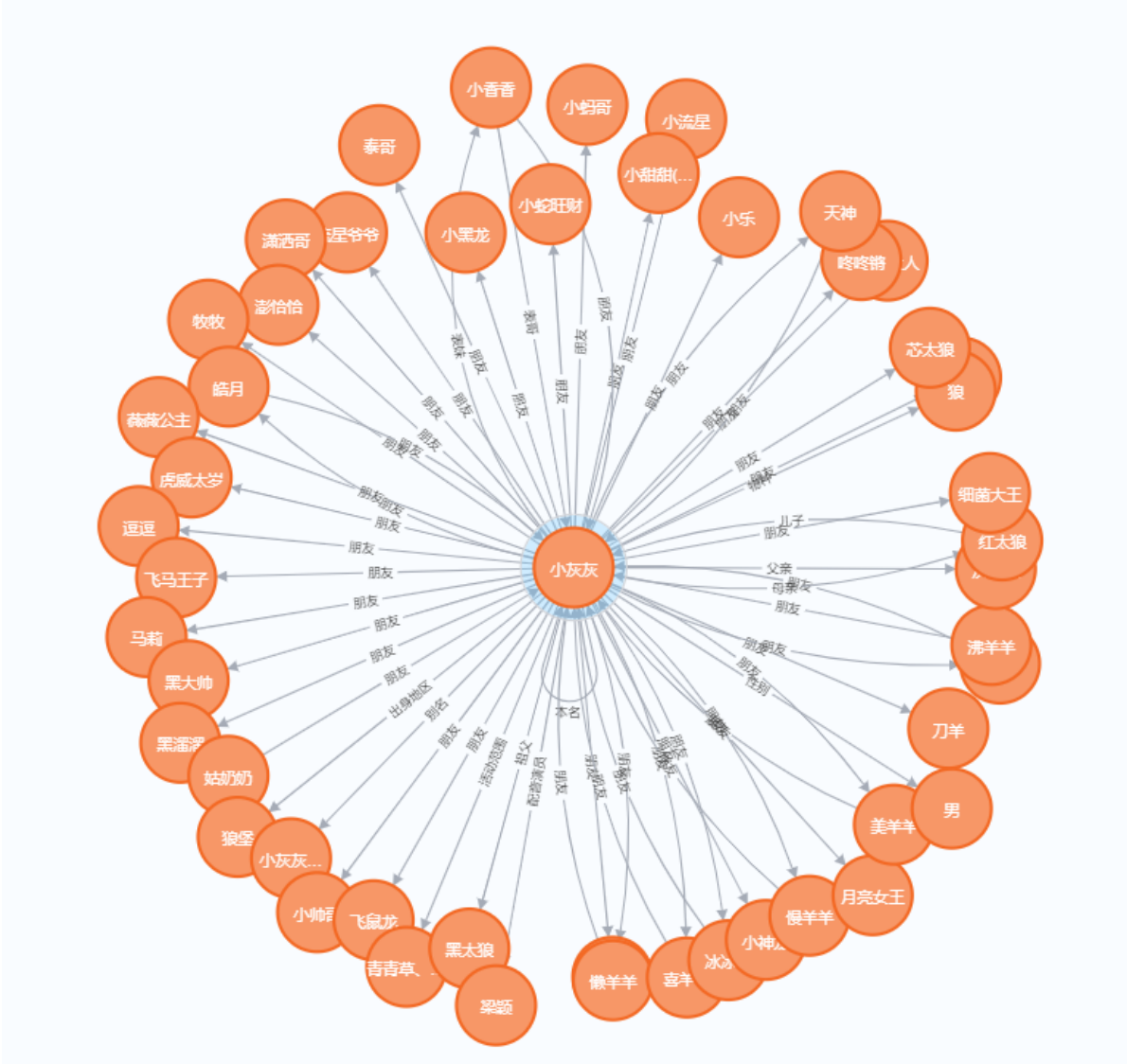




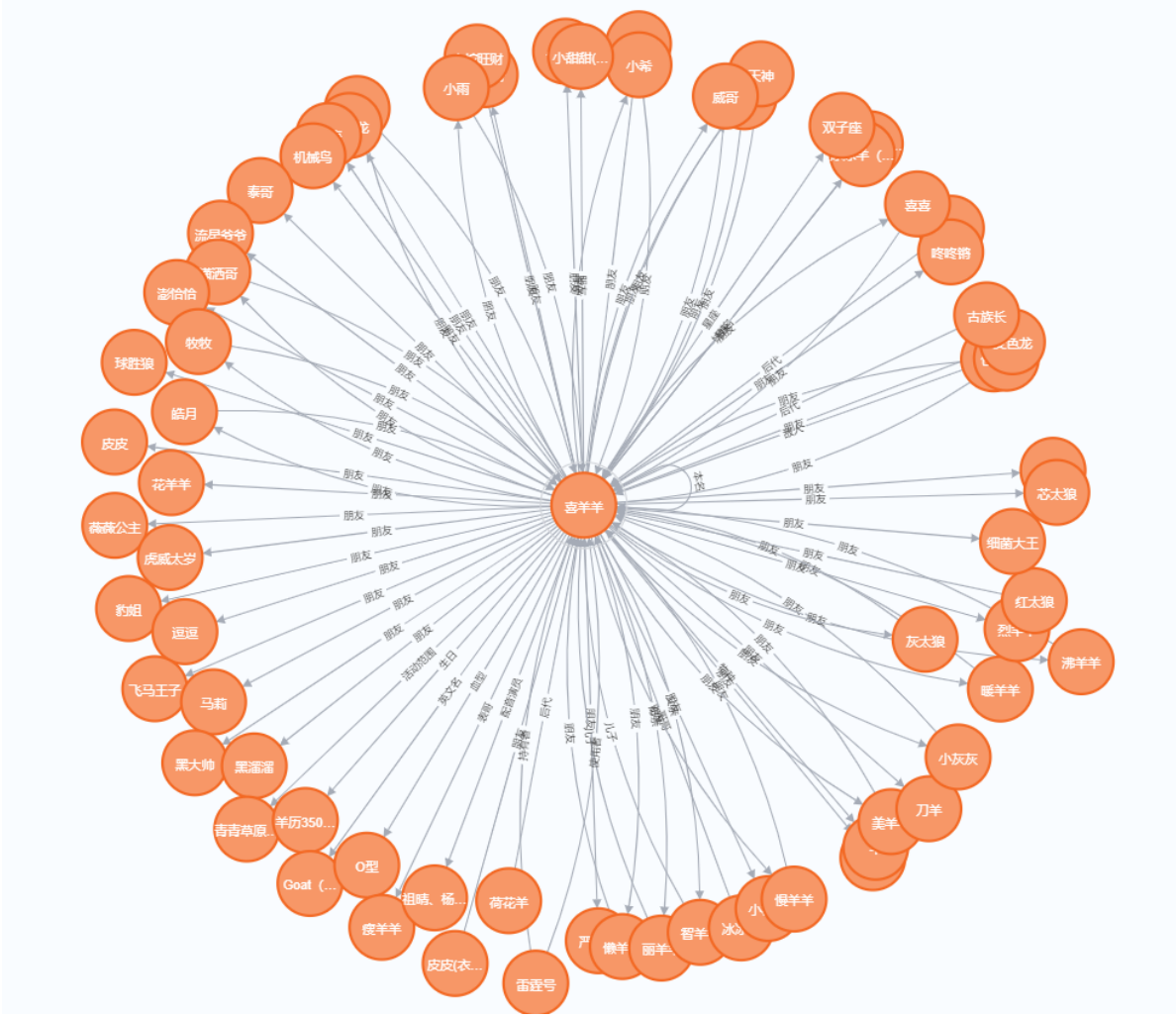
（2）知识查询。例如，在 Neo4j Brows 终端输入：

```
MATCH p=(n:Person{name:" 小灰灰"})-[r:朋友]->() RETURN p
```

小灰灰的图谱展现结果如下：



同理，喜羊羊知识图谱展现如下：



4.2 知识问答

首先，采用jieba工具库对中文自然语言问题进行词性标注，取词性为名称的词语作为问题关键词，并假设问句中的第一个名称词语为实体名称，第二个名称词语为关系名称，调用`answer()`函数实现问题解析。具体实现代码在`喜羊羊与灰太狼\可视化与知识问答\KGQA.py`目录下。

```
def answer(self, sentence):
    """
        :param sentence:str 用户输入的问句，如"喜羊羊的好朋友是谁"
        :return result: str 如: ""
        """
    try:
        sentence = re.sub("[A-Za-z0-9\!\%\[\]\,\。]", "", sentence)
        sentence = re.sub('\W+', '', sentence).replace("_", '')
        person, words = self.cut_words(sentence)

        words_ = [0 for i in range(len(words))]
        for i in range(len(words)):
            words_[i] = '-[r'+str(i) + ':'+words[len(words)-i-1]+']'
            if i != len(words)-1:
                words_[i] += '->(n'+str(i)+':Person)'
                quary = "match(p)" + ''.join(words_) + \
                "->(n:Person{Name:'"+person + \
                "'}) return  p.Name,n.Name,p.cate,n.cate"
                logging.debug(str(quary))
                except Exception as e:
```

```
                return '问题有误'
            try:
                data = self.neo.graph.run(quary)
                data = list(data)[0]
                logging.debug(str(data))

                result = data['n.cate']+'的'+data['n.Name']
                for item in words:
                    result += '的'
                    result += item
                    result += '是'
                    result += data['p.cate']+'的'+data['p.Name']
                    return result
                except Exception as e:
                    return '没有答案'
```

例如，输入问题：

用户：*小灰灰的父亲是谁？*

解析得到，`小灰灰`和`父亲`两个词语。

接着，采用Neo4j查询语句查询知识图谱，返回问题答案。
例如，输入问题：`小灰灰的爸爸是？`，返回回答为：`灰太狼`。

用户：*小灰灰的爸爸是？*
回答： 灰太狼

实践其他问答结果如下：

```
用户：小灰灰的爸爸是？
回答： 青青草原的小灰灰的爸爸是狼堡的灰太狼

用户：小灰灰的妈妈是？
回答： 青青草原的小灰灰的爸爸是狼堡的红太狼

用户：红太狼的妈妈是？
回答： 狼堡的的红太狼的妈妈是狼堡的粉红太狼

用户：喜羊羊的爸爸是？
回答： 青青草原的的喜羊羊的爸爸是青青草原的智羊羊

用户：喜羊羊的妈妈是？
回答： 青青草原的的喜羊羊的爸爸是青青草原的丽羊羊

用户：智羊羊的妻子是？
回答： 青青草原的的智羊羊的妻子是青青草原的丽羊羊

用户：灰太狼的爸爸是？
回答： 狼堡的的灰太狼的爸爸是狼堡的黑太狼
```