Entity relationship visualization

-- based on Bert Pre-trained model & Co-occurrence Matrix

Chen Xumin	19430019
Lei Chuyue	19409281
Dai Zhen	19430078

Abstract:Considering reasons for device environment and data, we chose to use BERT[1] model which perform well in pre-trained models, fit our data through fine-tuning, and then complete the downstream NER part, extracting the named entity. After that, we build the co-occurrence matrix of entities that appear in each sentence to visualize the relationship between these entities.

Keywords: BERT, Pre-trained model, NER, Co-occurrence Matrix, Data Visualization

1 Motivation

Traditional named entity recognition and relationship extraction methods need to be marked by a large amount of data, but it requires a lot of human resources and workload to achieve, meaning that it is time consuming and costly. Therefore, we considered whether there was a mode that satisfied two conditions of less data annotation and good effect of entity extraction. Also, it should not only generate word vectors well but also perform well on NER tasks. In addition, the task of Relation Extraction can be achieved through supervised learning, but it also causes a decrease in efficiency. So we thought about extracting word vectors containing contextual semantic relations through BERT, which use multi-layer transformers[2] encoder part may lead to pay a high attention to other token in the same sentence, and that means the word embeddings can also consider the contextual semantic.

In addition, BERT is a bi-directional pre-trained model, while ELMo[3] also a bi-directional pre-trained model, but sometimes it might repeat and weaken the relationship between each token. And calculating frequency of occurrence between entities. It simplifies calculation complexity of the entire process, and realizes the relationship extraction by constructing a co-occurrence matrix, we try to find an easy way to do this instead of use another language model, such as GloVe[4] and LDA[5].

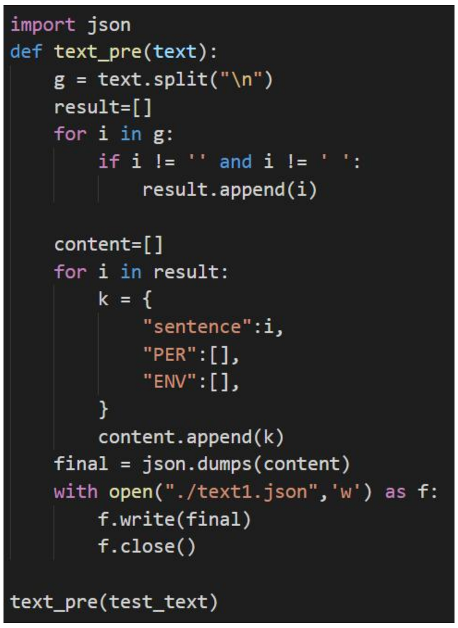
On the other hand, scenes in real life are more complicated. We tried to extract the entity relationship from the news corpus crawled on the Internet, but extraction results were not good due to limitation of the length of each news and highly sparse relationship between the named entities of the characters. Then, we considered the possibility of replacing news with novels. Because the plot of the novel is compact, while the frequency of occurrence between named entities is not low. Fortunately, the text data content we need already appeared on Kaggle. In the end, we decided to use the entire Harry Potter series as our text data.

2 Details of all the steps

2.1 Annotate entities for some texts in the novel

2.1.1 Use python script to divide the novel into sentences and mark names of characters appearing in sentences as entities

Firstly, we use line break symbol("\n") to split the text. If the result is not empty, we will add it to a new list (result); then add three tags to each element in the list: sentence, PER and ENV. Where, "Sentence" means the elements in the list (segmented sentence); "PER" means the name of the person appearing in the sentence; "ENV" means the name of the specific environment that appears in the sentence such as the specific company name, school name, etc.



```python
import json
def text_pre(text):
    g = text.split("\n")
    result=[]
    for i in g:
        if i != '' and i != ' ':
            result.append(i)

    content=[]
    for i in result:
        k = {
            "sentence":i,
            "PER":[],
            "ENV":[],
        }
        content.append(k)
    final = json.dumps(content)
    with open("./text1.json",'w') as f:
        f.write(final)
        f.close()

text_pre(test_text)
```

Figure 1. Sentence segmentation code

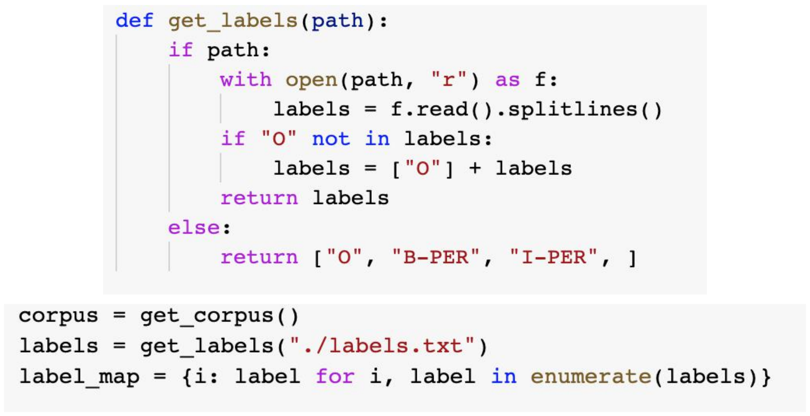
2.1.2 BIO sequence annotation for some chapters of the novel

Sequence labeling refers to labeling each element in a sequence. To be Specified, each word in the sentence is labeled differently so that the artificial intelligence algorithm will recognize and classify the text data by supervised learning.

There are two types of sequence labeling: Raw labeling and Joint segmentation and labeling. In the raw labeling method, each word exists independently and will be labeled. However, there are noun phrases in English which different words are combined to express a unique entity such as using a surname and first name to represent a special name. Therefore, you need to use joint annotations to annotate phrases, rather than labeling each word separately.

BIO labeling is a simple method to simplify the joint labeling problem to the raw labeling problem. Specifically, for the noun phrase, the first word is marked as "B-X" (the word is in the begin of the noun phrase and belongs to the X type), and then the second and last word are marked as "I-X" (the word is in the immediate of noun phrase, and belongs to the X type); in the remaining part, the single word representing the entity is marked as "B-X" while the meaningless entity is marked as "O" (Other).

In our project, we only mark human named entities and others will be marked as "O". For name entities, the person's name is marked as “B-PER” (First name) or “I-PER”(last name). Also, a position is labeled as B-PER with the second name as “I-PER”, etc.



```
def get_labels(path):
    if path:
        with open(path, "r") as f:
            labels = f.read().splitlines()
        if "O" not in labels:
            labels = ["O"] + labels
        return labels
    else:
        return ["O", "B-PER", "I-PER", ]
```

```
corpus = get_corpus()
labels = get_labels("./labels.txt")
label_map = {i: label for i, label in enumerate(labels)}
```

Figure 2. BIO annotation code

2.2 Fine-tune the BERT model, use labeled data with BIO annotation to extract entities

2.2.1 Fine-tuning step

For the NER task[6], a large number of annotations increase the workload for the entire project, so we need to adopt a new method to achieve. According to the paper [1], the author mentions that BERT performs well on 11 natural language processing models, including the NER task we are going to do. When the author was doing the NER experiment, the F1 score based on the CoNLL-2003 data set was higher than that of the ELMo [2] and other models. Therefore, we decided to mark part of the data, and then fit the entire model to our data with the fine-tuning method.

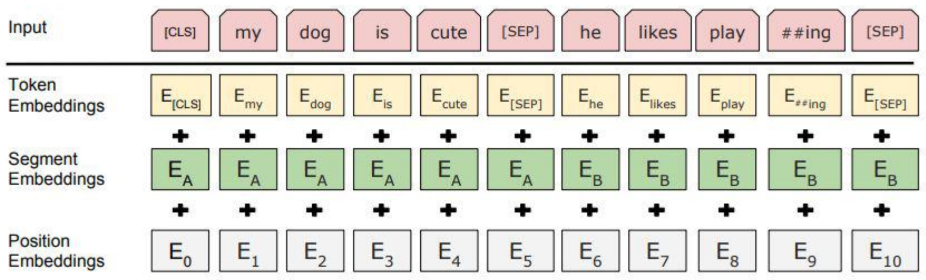

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3. Fine-tuning method

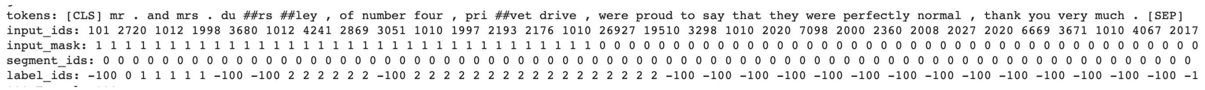
We use the text of the first six chapters of Harry Potter as our training data with the other chapters as prediction data sets. The input format is shown below[1]:

This input method is in accordance with the embedding method mentioned in the paper:

2.2.2 Output Constraints

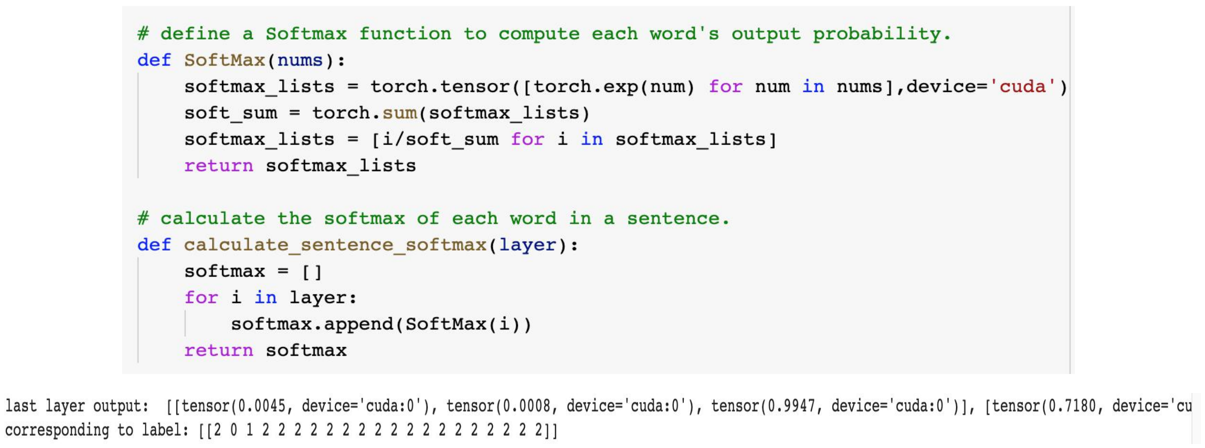
In addition, the paper also mentioned the model structure of BERT when doing NER tasks. Each token had an output label. In fact, after understanding the paper, we learned that the output label was not the real output. The output of the BERT model was the probability distribution of each token, that is, the probability of being a certain label. We need to input this probability into a fully connected layer and made a mapping to the id of the real label.

At the same time, in order to ensure that our output is a probability distribution, that is to say, the value range of the output is between [0,1], we did Softmax on the output result, which ensured that the output result was the probability distribution we wanted.



```
tokens: [CLS] mr . and mrs . du ##rs ##ley , of number four , pri ##vet drive , were proud to say that they were perfectly normal , thank you very much . [SEP]
input_ids: 101 2720 1012 1998 3680 1012 4241 2869 3051 1010 1997 2193 2176 1010 26927 19510 3298 1010 2020 7098 2000 2360 2008 2027 2020 6669 3671 1010 4067 2017
input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
label_ids: -100 0 1 1 1 1 1 -100 -100 2 2 2 2 2 2 -100 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -1
```

Figure 4. Before Softmax



```python
# define a Softmax function to compute each word's output probability.
def SoftMax(nums):
    softmax_lists = torch.tensor([torch.exp(num) for num in nums],device='cuda')
    soft_sum = torch.sum(softmax_lists)
    softmax_lists = [i/soft_sum for i in softmax_lists]
    return softmax_lists

# calculate the softmax of each word in a sentence.
def calculate_sentence_softmax(layer):
    softmax = []
    for i in layer:
        softmax.append(SoftMax(i))
    return softmax
```

```
last layer output:  [[tensor(0.0045, device='cuda:0'), tensor(0.0008, device='cuda:0'), tensor(0.9947, device='cuda:0')], [tensor(0.7180, device='cu
corresponding to label: [[2 0 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]]
```

Figure 5. Softmax code and result

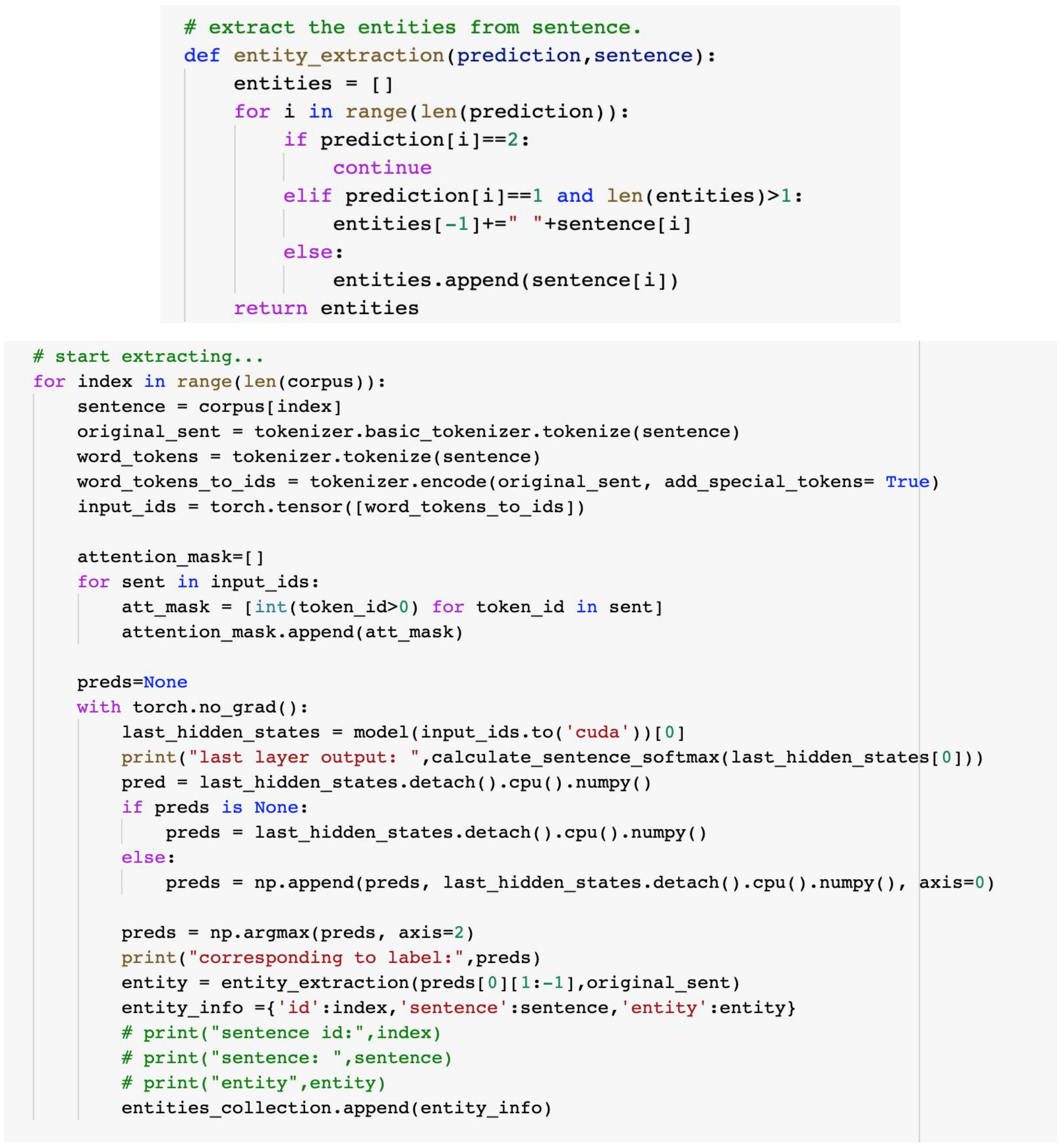
2.2.3 Named Entity Recognition

After fine-tuning our model, the model had been adapted to our data and needs. So we then used this trained model to predict named entities in our corpus.

The output of the model is as follows, which is the softmax in section 3.2.2. That is the probability of which label the output of each token belongs to. Through the argmax () function, we obtained the index of the maximum probability. This index was matched with the label dictionary to get the category to which it belonged.

2.2.4 Entity Extraction

Finally, we need to extract the named entity of the output from the original sentence. We used another tokenize method. According to the bert's BERT Tokenizer module during training and prediction phase, we identified each token in the sentence. If it was in the pre-trained vocabulary set, the token of the word would be returned; if not, it would be divided into several stems or replaced with [UNK]. But in this way, what we extract was not the result we wanted. As a result, we used another kind of tokenize, that is, extracting from the original sentence, without adding any mask and word segmentation operations.



```python
# extract the entities from sentence.
def entity_extraction(prediction,sentence):
    entities = []
    for i in range(len(prediction)):
        if prediction[i]==2:
            continue
        elif prediction[i]==1 and len(entities)>1:
            entities[-1]+=" "+sentence[i]
        else:
            entities.append(sentence[i])
    return entities
```

```python
# start extracting...
for index in range(len(corpus)):
    sentence = corpus[index]
    original_sent = tokenizer.basic_tokenizer.tokenize(sentence)
    word_tokens = tokenizer.tokenize(sentence)
    word_tokens_to_ids = tokenizer.encode(original_sent, add_special_tokens= True)
    input_ids = torch.tensor([word_tokens_to_ids])

    attention_mask=[]
    for sent in input_ids:
        att_mask = [int(token_id>0) for token_id in sent]
        attention_mask.append(att_mask)

    preds=None
    with torch.no_grad():
        last_hidden_states = model(input_ids.to('cuda'))[0]
        print("last layer output: ",calculate_sentence_softmax(last_hidden_states[0]))
        pred = last_hidden_states.detach().cpu().numpy()
        if preds is None:
            preds = last_hidden_states.detach().cpu().numpy()
        else:
            preds = np.append(preds, last_hidden_states.detach().cpu().numpy(), axis=0)

        preds = np.argmax(preds, axis=2)
        print("corresponding to label:",preds)
        entity = entity_extraction(preds[0][1:-1],original_sent)
        entity_info ={'id':index,'sentence':sentence,'entity':entity}
        # print("sentence id:",index)
        # print("sentence: ",sentence)
        # print("entity",entity)
        entities_collection.append(entity_info)
```

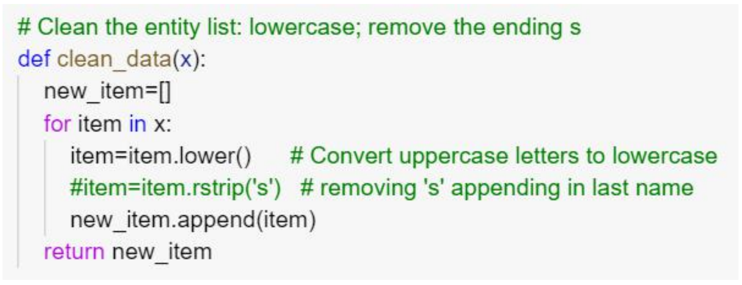
Figure 6. Entity Extraction Code

In addition, it is worth mentioning that considered the purpose of the next layer of tasks which is to extract the relationship of named entities, BERT would act as a Bi-directional pre-trained model, combined with the transformer part of multiple self-attention layers. It also combined semantics in Deep-level regions as well as information extraction between entity relationships.

2.3 Cleaning entity data extracted through the BERT model

After using the NER model to automatically extract named entities, the entity data extracted by the model contains some noise (meaningless words). For example, there are some words like “He”, “it”,”potter delightly”, None and so on are extracted wrongly as entities. So these data need to be cleaned to provide more accurate data for the subsequent word vector model.

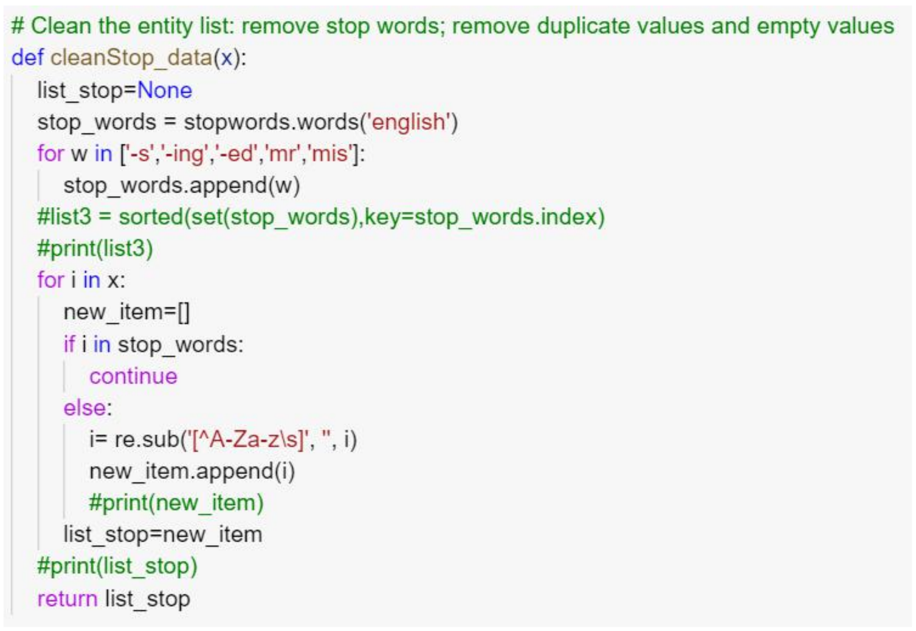
(1) First lowercase the entity data and convert uppercase letters to lowercase to facilitate stopword recognition;



```
# Clean the entity list: lowercase; remove the ending s
def clean_data(x):
    new_item=[]
    for item in x:
        item=item.lower()      # Convert uppercase letters to lowercase
        #item=item.rstrip('s')   # removing 's' appending in last name
        new_item.append(item)
    return new_item
```

Figure 7. Lowercase code

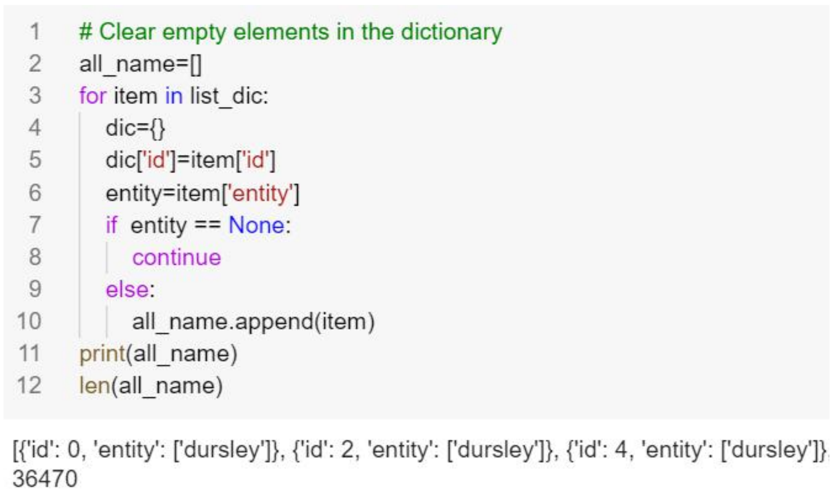
(2) Secondly, use NLTK's English stop word list to remove irrelevant words such as "the", "it", "a", "he" in the entity and only retain meaningful entities. Considering that data also contains digital noise, we also use re.sub to remove the numbers;



```python
# Clean the entity list: remove stop words; remove duplicate values and empty values
def cleanStop_data(x):
    list_stop=None
    stop_words = stopwords.words('english')
    for w in ['-s','-ing','-ed','mr','mis']:
        stop_words.append(w)
    #list3 = sorted(set(stop_words),key=stop_words.index)
    #print(list3)
    for i in x:
        new_item=[]
        if i in stop_words:
            continue
        else:
            i= re.sub('[^A-Za-z\s]', '', i)
            new_item.append(i)
            #print(new_item)
        list_stop=new_item
    #print(list_stop)
    return list_stop
```

Figure 8. Stop list function code

(3) When traversing the dictionary, if the element in the dictionary is None, the traversal will make an error. Therefore, we should know the "None" key value.\



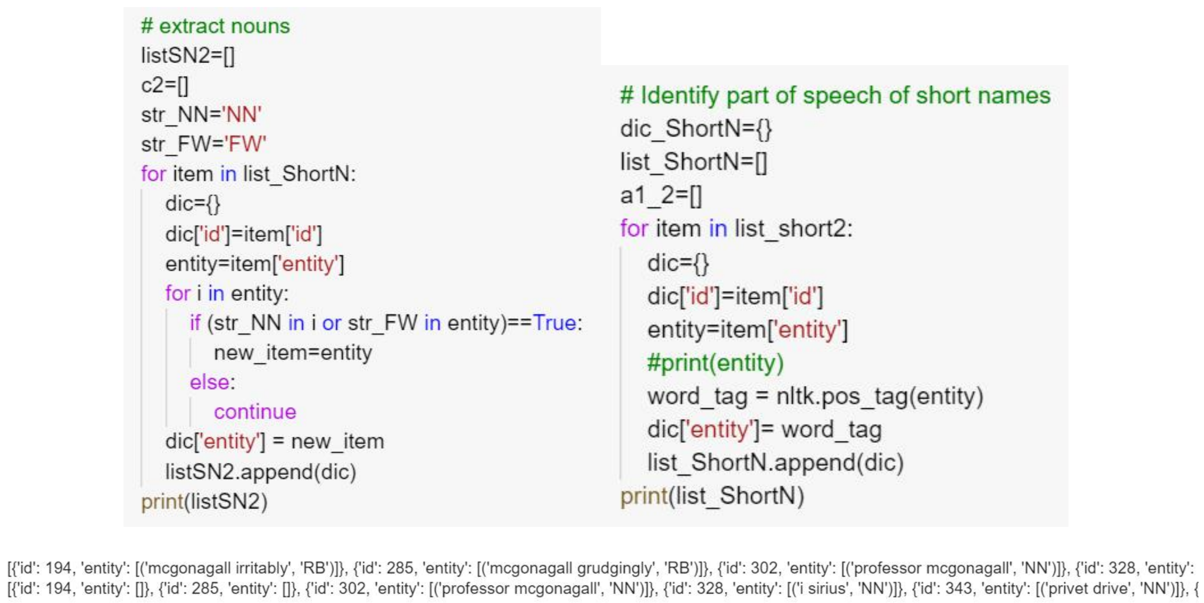
```python
# Clear empty elements in the dictionary
all_name=[]
for item in list_dic:
    dic={}
    dic['id']=item['id']
    entity=item['entity']
    if  entity == None:
        continue
    else:
        all_name.append(item)
print(all_name)
len(all_name)
```

```
[{'id': 0, 'entity': ['dursley']}, {'id': 2, 'entity': ['dursley']}, {'id': 4, 'entity': ['dursley']}
36470
```

Figure 9. Clean empty elements in dictionary code

(4) Import the Lemmatisation module in the NLTK package, using the pos_tag () method[1], recognizing the part of speech so that we can only retain the nouns.

[1] http://www.nltk.org/book/ch05.html



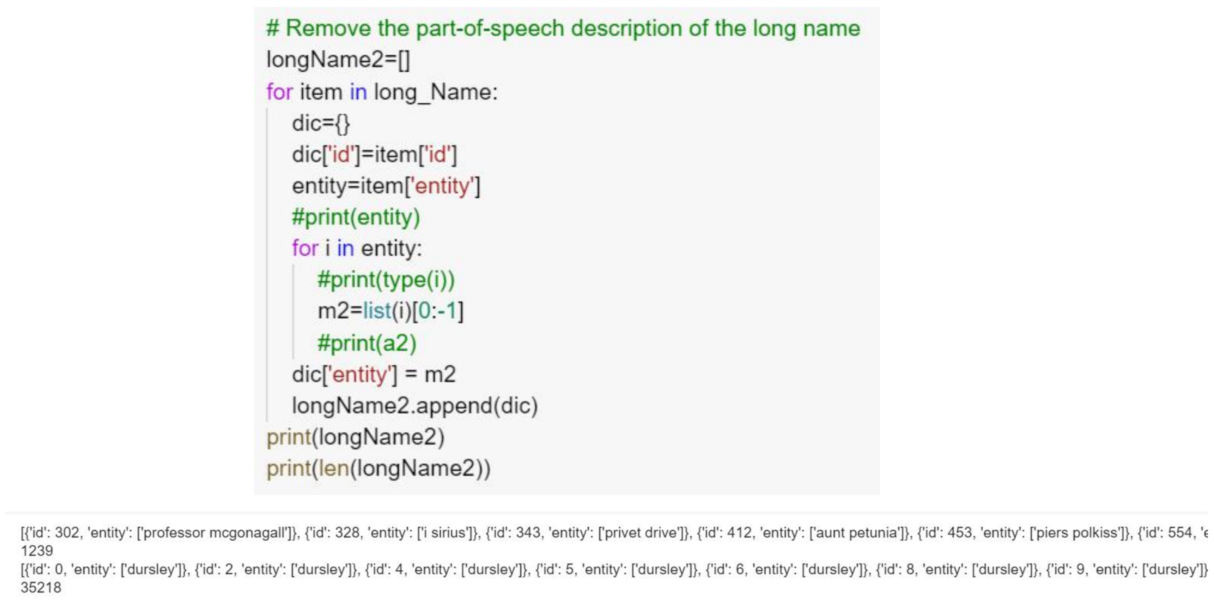
```
# extract nouns
listSN2=[]
c2=[]
str_NN='NN'
str_FW='FW'
for item in list_ShortN:
    dic={}
    dic['id']=item['id']
    entity=item['entity']
    for i in entity:
        if (str_NN in i or str_FW in entity)==True:
            new_item=entity
        else:
            continue
    dic['entity'] = new_item
    listSN2.append(dic)
print(listSN2)
```

```
# Identify part of speech of short names
dic_ShortN={}
list_ShortN=[]
a1_2=[]
for item in list_short2:
    dic={}
    dic['id']=item['id']
    entity=item['entity']
    #print(entity)
    word_tag = nltk.pos_tag(entity)
    dic['entity']= word_tag
    list_ShortN.append(dic)
print(list_ShortN)
```

```
[{'id': 194, 'entity': [('mcgonagall irritably', 'RB')]}, {'id': 285, 'entity': [('mcgonagall grudgingly', 'RB')]}, {'id': 302, 'entity': [('professor mcgonagall', 'NN')]}, {'id': 328, 'entity':
[{'id': 194, 'entity': []}, {'id': 285, 'entity': []}, {'id': 302, 'entity': [('professor mcgonagall', 'NN')]}, {'id': 328, 'entity': [('i sirius', 'NN')]}, {'id': 343, 'entity': [('privet drive', 'NN')]}, {
```

Figure 10. Lemmatisation Code and result

(5) Among them, "NN" means Noun, singular or mass; "FW" means Foreign word; "RB" means Adverb; "JJ" means Adjective. We only need to retain data with "NN" or "FW" labels. So remove part-of-speech descriptions and keep only the names. As the subsequent co-occurrence matrix does not need part of speech.

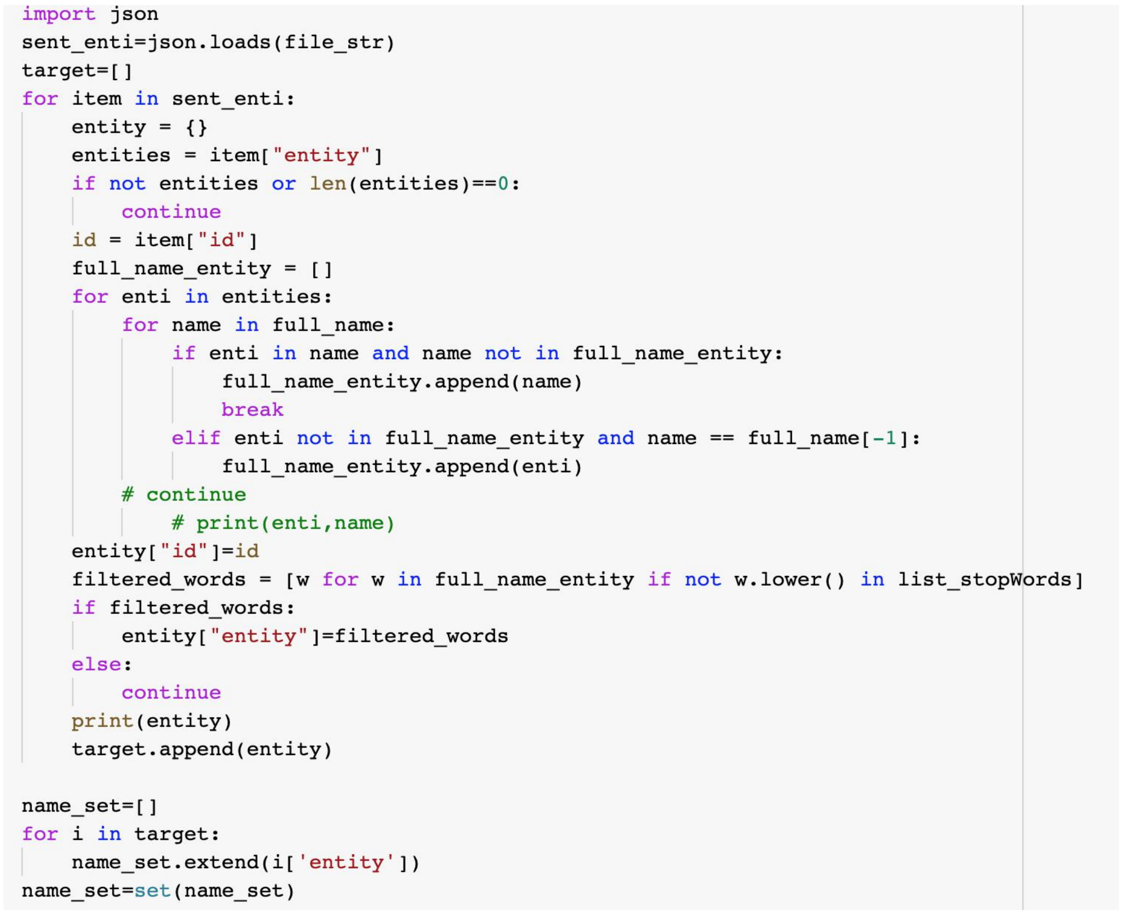


```
# Remove the part-of-speech description of the long name
longName2=[]
for item in long_Name:
    dic={}
    dic['id']=item['id']
    entity=item['entity']
    #print(entity)
    for i in entity:
        #print(type(i))
        m2=list(i)[0:-1]
        #print(a2)
    dic['entity'] = m2
    longName2.append(dic)
print(longName2)
print(len(longName2))
```

```
[{'id': 302, 'entity': ['professor mcgonagall']}, {'id': 328, 'entity': ['i sirius']}, {'id': 343, 'entity': ['privet drive']}, {'id': 412, 'entity': ['aunt petunia']}, {'id': 453, 'entity': ['piers polkiss']}, {'id': 554, 'e
1239
[{'id': 0, 'entity': ['dursley']}, {'id': 2, 'entity': ['dursley']}, {'id': 4, 'entity': ['dursley']}, {'id': 5, 'entity': ['dursley']}, {'id': 6, 'entity': ['dursley']}, {'id': 8, 'entity': ['dursley']}, {'id': 9, 'entity': ['dursley']}
35218
```

Figure 11. Remove part-of-speech descriptions Code and result

(6) Compare naming libraries and replace surnames and first names with full names



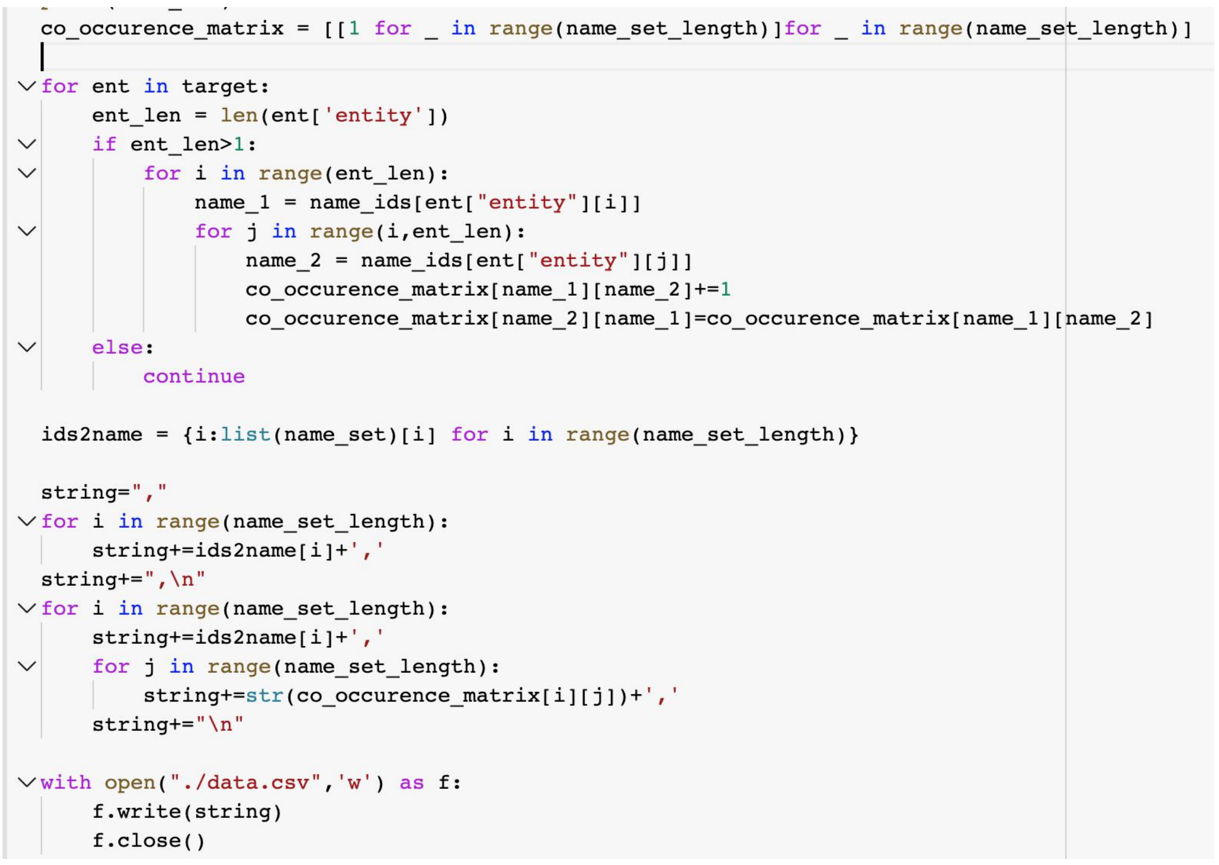
```python
import json
sent_enti=json.loads(file_str)
target=[]
for item in sent_enti:
    entity = {}
    entities = item["entity"]
    if not entities or len(entities)==0:
        continue
    id = item["id"]
    full_name_entity = []
    for enti in entities:
        for name in full_name:
            if enti in name and name not in full_name_entity:
                full_name_entity.append(name)
                break
            elif enti not in full_name_entity and name == full_name[-1]:
                full_name_entity.append(enti)
        # continue
            # print(enti,name)
    entity["id"]=id
    filtered_words = [w for w in full_name_entity if not w.lower() in list_stopWords]
    if filtered_words:
        entity["entity"]=filtered_words
    else:
        continue
    print(entity)
    target.append(entity)

name_set=[]
for i in target:
    name_set.extend(i['entity'])
name_set=set(name_set)
```

Figure 12. Replace surnames and first names with full names Code and result

2.4 Substituting entity data into the co-occurrence model to calculate word vectors

The row and column length of the initial co-occurrence matrix are the length of the set of names in the novel. The content in the matrix is the frequency[7] of the names of the people counted in the above data. We get a frequency matrix of names, but we need to reduce the dimension of the matrix to make it run due to the limitation of the visualization tool Gephi. Therefore, a random matrix is created, the number of rows is the length of the name set in the novel and the number of columns is 1. In this case, after dimension reduction, the matrix size finally input into Gephi is length of the name set * 1.



```
co_occurence_matrix = [[1 for _ in range(name_set_length)]for _ in range(name_set_length)]

for ent in target:
    ent_len = len(ent['entity'])
    if ent_len>1:
        for i in range(ent_len):
            name_1 = name_ids[ent["entity"][i]]
            for j in range(i,ent_len):
                name_2 = name_ids[ent["entity"][j]]
                co_occurence_matrix[name_1][name_2]+=1
                co_occurence_matrix[name_2][name_1]=co_occurence_matrix[name_1][name_2]
    else:
        continue

ids2name = {i:list(name_set)[i] for i in range(name_set_length)}

string=","
for i in range(name_set_length):
    string+=ids2name[i]+','
string+=",\n"
for i in range(name_set_length):
    string+=ids2name[i]+','
    for j in range(name_set_length):
        string+=str(co_occurence_matrix[i][j])+','
    string+="\n"

with open("./data.csv",'w') as f:
    f.write(string)
    f.close()
```

Figure 12. Co-occurrence code

In Gephi, the layout method used is Fruchterman Reingold. It ensurs that each node surrounds the central character , and the distance represents[7] how many times the central character and similarity[8] appears together.

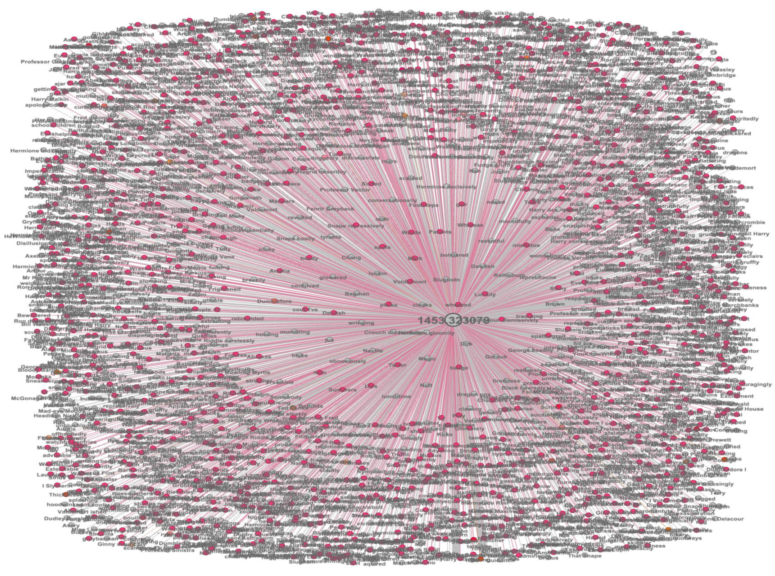


Figure 13. Gephi Graph about entity relationship

3 Discussion of the results obtained and your interpretation

From the result of the picture, it is not ideal enough. After analysis, the possible reasons are as follows:

1. No model can identify the named entity in 100%, so the output of the NER part has noisy data;
2. During the second cleaning, the data was not completely cleaned, which leaded to interference items when constructing the co-occurrence matrix[9];
3. We generated a dictionary for named entities to store the dictionary and the number id, but the length of the dictionary was too large. We tried to reduce the dimension of the co-occurrence proof. However, once the dimension is reduced, the dimension is also reduced, which weakens the entity relationship, resulting in poorly displaying with distance between entities.

4 Conclusion

Based on the BERT model and co-occurrence matrix, we extracted the name entity in the Harry Potter novels and visualized the word vector distance to obtain the relationship graph. However, the result is not satisfactory enough. Even though we simplified the process of relationship extraction and optimized the visualization of the entire relationship graph, the accuracy decreased due to noise data. Besides, the relationship graph was so large that it caused the calculation amount to exceed expectations.

5 References

[1]Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[2]Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.

[3]Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations[J]. arXiv preprint arXiv:1802.05365, 2018.

[4]Pennington J, Socher R, Manning C D. Glove: Global vectors for word representation[C]//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1532-1543.

[5]Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of machine Learning research, 2003, 3(Jan): 993-1022.

[6]Nadeau D, Sekine S. A survey of named entity recognition and classification[J]. Lingvisticae Investigationes, 2007, 30(1): 3-26.

[7]Kusner M, Sun Y, Kolkin N, et al. From word embeddings to document distances[C]//International conference on machine learning. 2015: 957-966.

[8]Ye X, Shen H, Ma X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C]//Proceedings of the 38th international conference on software engineering. 2016: 404-415.

[9]Chen P C , Pavlidis T . Segmentation by texture using a co-occurrence matrix and a split-and-merge algorithm[J]. Computer Graphics and Image Processing, 1979, 10(2):172-182.

Person	Contribution
Chen Xumin **(40%)**	Model fine-tuning and construct NER task as our downstream task; Recognize and extract entities from the original text; Perform secondary cleaning and replacement of the data

Lei Chuyue **(30%)**	Named entity by BIO annotations;
	Text processing;
	Data cleaning

Dai Zhen **(30%)**	Construction of co-occurence matrix; Data visualization