实验二报告

小组成员：常鑫鑫-PB17111649，杜艺帆-PB17111594

1. 实验目的

实验要求用给定的英文文本数据为基础，实现一个信息抽取系统，能够对测试数据中给出的句子，预测出句子所包含的实体之间的关系类型，即实现实体关系抽取。其中，实验数据为如下格式：

```
1 "The system as described above has its greatest application in an arrayed
configuration of antenna elements ."
Component-Whole(elements,configuration)
2 "The child was carefully wrapped and bound into the cradle by means of a
cord."
Other(child,cradle)
```

测试数据为如下格式：

```
6401 "The body of her nephew was in a suitcase under the bed."
6402 "The drama unfolded shortly after 7pm last Tuesday (December 22), when
Glyn saw that smoke was coming from a bonfire ."
```

数据中包含的所有实体关系类型为：

```
Cause-Effect、Component-Whole、Entity-Destination、Product-Producer、Entity-
Origin、
Member-Collection、Message-Topic、Content-Container、Instrument-Agency、Other
```

其中，other表示不满足前面九种关系的剩余所有关系类型。

2. 实验步骤与实现

本次实验小组成员分别实现了一种算法，`2.1`为常鑫鑫同学的实现方法，`2.2`为杜艺帆同学的实现方法，以下对两种方法分别进行说明：

2.1 根据关系类型分析语句的tf-idf方法

由于属于不同类别的句子自身存在特点，如：出现的词语词频不同。根据对数据集的统计，属于类型`Cause-Effect`的句子中，`cause`出现的频率约为0.7，而属于其他关系类型的句子几乎不含有该词。受到实验一中使用的tf-idf检索方法启发，对于本实验的一个实现方法为：采用类似tf-idf的思路，对idf的计算进行调整，然后构造描述关系类型特征的向量，对测试数据中的句子按此方法，比较得出最可能的关系类型。具体函数实现描述如下：

首先从`train.txt`中读出训练数据，从句子后一行，分离出实体1、实体2和关系类型，并构建字典，字典内容包括实体、关系和句子：

```
with open(train_data_path, 'r') as f:
  while True:
    data = {}
    a = f.readline()

    if not a:
      break
      data['content'] = a

      b = f.readline()
      relationship = b[:b.find('(')]
      entity = []
      entity_1 = b[b.find('(') + 1: b.find(',')]
      entity_2 = b[b.find(',') + 1: -2]
      entity.append(entity_1)
      entity.append(entity_2)

      data['relationship'] = relationship
      data['entity'] = entity

      train_data.append(data)
```

之后统计数据中各种单词出现的频率，对句子进行分词、词根化、去停用词的预处理，其中对于去停用词进行优化：由于对句子类型进行分析，部分类型的句子中，能够表示其特征的词语在正常的英文停用词表里，按照正常的方法去停用词，会导致此部分重要词语被删除，影响最终分类结果，故设置`useful_stopwords_list`存储有意义的停用词，保留此部分词语：

```
# 统计每条数据中各单词出现的次数
posting_list = {}
for data in current_data:
  content = data['content']
  # 分词
  word_tokens = word_tokenize(content)
  # 去停用词
  stop_words = set(stopwords.words('english'))
  for ele in useful_stopwords_list: # 存储有意义的停用词
    stop_words.remove(ele)
  filtered_tokens1 = [w for w in word_tokens if w.lower() not in stop_words]
  # 词根化
  ps = PorterStemmer()
  stemmed_tokens = [ps.stem(w) for w in word_tokens if w.isalpha()]
  # 去停用词
  filtered_tokens2 = [w for w in stemmed_tokens if w.lower() not in stop_words]
  for token in set(filtered_tokens2):  # 只考虑在该条数据中是否出现，不考虑数量
    if token not in posting_list:
      posting_list[token] = 1
    else:
      posting_list[token] += 1
```

然后，根据每种关系训练出反应该关系特征的向量：

```
feature_vec = {}
for relationship in relationship_list:
  # 得到符合当前关系的数据
  current_data = []
  for data in train_data:
    if data['relationship'] == relationship:
      current_data.append(data)
  ...   # 统计每条数据中各单词出现的次数
  feature_vec[relationship] = {}
  for i in range(1650):
    if posting_list:
      a = max(posting_list.items(), key=lambda x: x[1])
      feature_vec[relationship][a[0]] = a[1]
      del posting_list[a[0]]
    else:
      print('list is already empty')
      break

for relationship_1 in relationship_list:
  for token in feature_vec[relationship_1]:
    freq = 0
    for relationship_2 in relationship_list:
      if token in feature_vec[relationship_2]:
        freq += 1
        feature_vec[relationship_1][token] *= (len(relationship_list) - freq)
```

最后，读入测试数据，并调用classify函数，实现对测试数据的关系类型预测，即按关系分类。classify函数先对测试句子进行分词、词根化、去停用词的预处理，之后根据前述过程中计算的各种关系类型的特征向量，通过比较，得出当前测试数据最可能的关系类型并返回：

```
def classify(sentence, feature_vec):
    max_score = 0
    max_score_relationship = ''
    for relationship in relationship_list:
        ... # 此部分内容进行分词、词根化、去停用词
        score = 0
        for token in set(filtered_tokens1):
            if token in feature_vec[relationship]:
                score += feature_vec[relationship][token]
        if score > max_score:
            max_score = score
            max_score_relationship = relationship
    if not max_score_relationship:
        return 'Other'
    return max_score_relationship
```

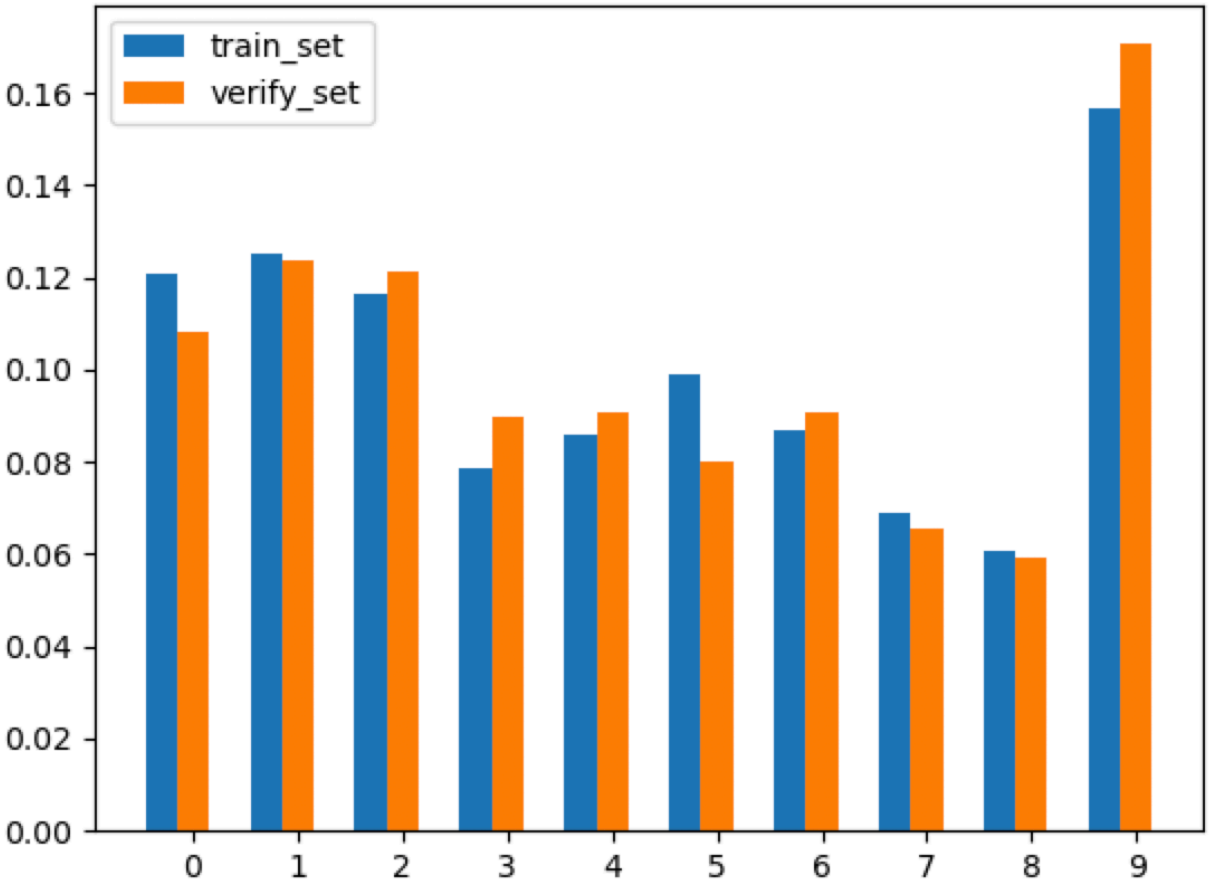
2.2 基于BiLSTM模型的方法

此方法主要为基于BiLSTM模型构造网络，由训练集给出的关系类型作为标签，对应的句子作为数据，采用word2vec方法，将所有句子转换成向量，进行训练，设置检查点和提前终止条件，保存验证集准确率最高的模型，学习率设置为当准确率上升时，逐渐下降的形式，训练出模型后，对测试数据进行分类。具体函数描述如下：

在训练程序`lab2.py`中，由于没有验证集，故在最初先对实验数据进行预处理。读入`train.txt`文件，将前4800条数据作为训练集，之后1600条数据作为验证集，分割后分别写入训练集数据文件和验证集数据文件，之后的训练以新文件进行：（注：由于在测试文件中并未指出实体，为防止过拟合，此处直接将实体相关的数据丢弃，不参加模型训练。）

```
def generate_data():
  # 生成数据文件：包括训练和测试数据
  sentence1, sentence2 = [], [] # 句子
  relation = [] # 关系
  with open(train_file_path, 'r', errors='ignore') as f:
    lines = f.readlines()
    for text in lines:
      # print(text)
      if text[0].isdigit():
        match_sentence = re.search(r'"(.*)"\n', text, re.M|re.I)
        # print(match_sentence)
        sentence1.append(match_sentence.group(1))
      else:
        match_relation = re.search(r'(.*)[(].*', text, re.M|re.I)
        relation.append(match_relation.group(1))
```

由于实验数据的统计数据未知，因此在数据预处理时，需要对训练集、验证集的数据进行统计，根据统计结果，other类型（即下图中第9类）句子最多：



且训练集最长的句子有86个词、验证集最长的句子有61个词；最终使用80作为句子最大单词数，将超过限制的句子截断。然后使用sklearn.preprocessing中的LabelEncoder对标签进行编码；之后对句子进行分词，由于对句子进行词根化并去停用词后，句子的单词减少明显，且对最终的结果影响很差，此处只将句子的所有词语按空格划分开。

使用word2vec对句子进行向量化，此处使用的优化为：引入`GoogleNews-vectors-negative300.bin`数据包（此数据包的内容为他人按照大量数据训练的每个词语特征的向量表示），并对训练集中所有词语进行查找，若在数据包中找到对应的词语，则在embedding_layer的初始化时，使用此数据包中的向量。使用数据包的向量显然优于随机指定embedding_layer中矩阵的数值：

```
# 增加embedding层初始值，利用以训练的包
embedding_size = 300
VECTOR_DIR = 'GoogleNews-vectors-negative300.bin'
print('加载VECTOR_DIR作为embedding初始值中...')
w2v_model = keyedvectors.Word2VecKeyedVectors.load_word2vec_format(VECTOR_DIR,
binary=True)
#w2v_model.save_word2vec_format('GoogleNews-vectors-negative300.txt',
binary=False)
embedding_matrix = np.zeros((len(dict.items()), embedding_size))
not_in_model = 0
in_model = 0
for word, i in dict.items():
  if word in w2v_model:
    in_model += 1
    embedding_matrix[i] = np.asarray(w2v_model[word], dtype='float32')
```

```
  else:
    not_in_model += 1
print(str(in_model) + ' in word2vec model')
print(str(not_in_model) + ' not in word2vec model')
```

然后构建网络：

```
## 模型构建，使用BiLSTM
# 配置网络结构
# model = krs.Sequential()
print('len dict: ', len(dict.items()))
embedding_layer = krs.layers.Embedding(len(dict.items()), embedding_size,
  weights=[embedding_matrix], input_length=max_sequence_length)
#trainable=True)
sentence_input = krs.layers.Input(shape=(max_sequence_length, ), dtype='int64')
embedded_sent = embedding_layer(sentence_input)
out_lstm = krs.layers.Bidirectional(krs.layers.LSTM(100,
return_sequences=False, dropout=0.2))(embedded_sent)
pred = krs.layers.Dense(10, activation='softmax')(out_lstm)
model = krs.models.Model(sentence_input, pred)
# 查看模型结构
model.summary()
```

对模型进行训练，损失函数使用`categorical_crossentropy`，优化器使用`adam`，评价指标为`accuracy`；设置BATCH_SIZE为128，EPOCH为10；再创建检查点和提前终止条件：

```
## 模型训练
# 设置优化器、损失函数和评价指标
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=
["accuracy"])
# 运行参数
BATCH_SIZE = 128
EPOCH = 10
## 定义回调函数
# 创建检查点和提前终止条件
ckpt = krs.callbacks.ModelCheckpoint('../temp/ckpt.h5', monitor='val_acc',
verbose=1,
  save_best_only=True, save_weights_only=True, period=1)
earlystop = krs.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.0001,
patience=5,
  verbose=True)
```

训练集训练结束后，再对验证集进行分类，并计算正确率accuracy：

```
metrics = model.evaluate(x=text_processed_verify, y=dummy_target_verify,
verbose=1)
print("verify_loss: ", metrics[0], "verify_accuracy: ", metrics[1])
```

在测试程序`lab2_test.py`中，先按照训练程序训练最佳的模型参数，重新构建网络：

```
embedding_layer = krs.layers.Embedding(6595, embedding_size,
input_length=max_sequence_length)#, trainable=True)
sentence_input = krs.layers.Input(shape=(max_sequence_length, ), dtype='int64')
embedded_sent = embedding_layer(sentence_input)
out_lstm = krs.layers.Bidirectional(krs.layers.LSTM(100,
 return_sequences=False, dropout=0.2))(embedded_sent)
pred = krs.layers.Dense(10, activation='softmax')(out_lstm)
model = krs.models.Model(sentence_input, pred)
model.load_weights('../temp/ckpt.h5') # 最佳模型存于此文件
```

然后对测试数据同样向量化表示后，进行测试，并将结果输出到文件：

```
with open('../result/results.txt', 'w', encoding='utf-8') as f:
  for i in y_pre:
    f.write(labels_index[int(y_pre[i])] + '\n')
```

3. 实验结果与分析

2中的方法`2.1`在平台的运行结果如下：

Filename	ACC-Relation	ACC-NER
常鑫鑫-PB17111649-5.txt	0.43625	0.0

方法`2.2`在平台的运行结果如下：

Filename	ACC-Relation	ACC-NER
杜艺帆-PB17111594-6.txt	0.18125	0.0

显然采用第一种方法效果更好。分析第二种方法效果差的原因可能为：数据集标签仅有实体之间的关系，并未指出实体，且数据集数量较小，无法达到很好的效果。

4. 总结

本实验我们小组通过实现了两种方式，得出结论：在数据集数目较小的情况下，只采用有监督训练模型的方式效果会受到较大影响，在此情况下，对数据本身进行分析，并在此基础上加以训练，结果会显著提升。