对话系统大作业报告

组员：蔡洛姗 袁宇箭 周子琪 郑宇鹏

分工：蔡洛姗-组长，训练集预处理，提取句子的特征向量，整合代码，debug并运行

袁宇箭-实现交互界面，计算相似度并选择最佳匹配答案输出，改进匹配算法

周子琪-训练集预处理，实现分词，引入自定义词典

郑宇鹏-实现倒排序索引，结果分析

一、交互界面

用于问答交互的测试驱动程序包含两种版本，shell.py文件支持多轮单次的问答，输入一个问题则会返回对应的答案；driver.py实现了对所给测试集所有问题逐个生成回答并保存为json文件。代码如下：

```
def print_answer(question):
    question_seg=segmentation([question], test_Qfile)   #分词
    question_vector=bert_client.encode([question])      #提取特征向量
    found=0
    #读取倒排序索引
    with open('data/index_table.json','r',encoding='utf-8') as fp:
        transfer_list = json.load(fp)
    #看看是否有相同的关键词
    for item in question_seg[0]:
        if item in transfer_list:#.keys()
            found=1
    #如果相同的关键词都没有，则无法找到答案
    if found==0:
        print('暂时寻找不到答案')
        return "NAN",0
    else:
        result_id=get_result_id(question_seg[0],question_vector,transfer_list)
        answer = find_answer(result_id)
        print("答案是：" + answer)
        return answer,result_id

if __name__ == '__main__':
    while True:
        sentence1 = input('请输入您需要问的冬奥会问题(输入quit退出)：\n')
        if sentence1 == 'quit':
            break
        else:
            answer,result_id = print_answer(sentence1)
```

通过将得到的答案与训练集或测试集给出的答案做对比，则可以计算模型的接受率。

二、数据预处理

我们对训练问答对的处理过程为：调用jieba分词库，将训练集的问句进行分词，然后去除停用词并，剩下的作为关键词保存成文件QuestionSeg.txt。随后需要建立一个到排序索引词典，以每一个关键词作为key，以包含这个key的句子下标集合作为value，value是一个元素为句子的索引值的list。此外，我们采用了BERT模型库来提取训练集问句的特征向量，并保存为numpy数组文件。根据多次运行训练集的结果，我们还补充了辅助词典和数词转换。辅助词典加入到jieba分词的词典中可以更好地划分专有名词。量词转换可以完成阿拉伯数字与中文汉字的转化，使得对问题的识别更加准确。具体的模块实现如下：

辅助词典：

```
jieba.load_userdict("our_dict/dict.txt")
```

词典的内容如下图：

拉尔松 25 nr
奥-哈根 25 nr
巴兰格鲁德 25 nr
吕-维代曼 25 nr
马尔-安德森 25 nr
努尔海姆 25 nr
利迪娅斯科布利科娃 25 nr
里查德·尼克松 25 nr
萨科娃 25 nr
格里申 25 nr
克约翰内森 25 nr
库利宁 25 nr
西耶恩伯格 25 nr

数词转换：

训练集和测试集中对于数量的表达可能会出现阿拉伯数字与汉字数字无法匹配的情况，因此我们考虑统一形式，在分词之前将测试集中可能与训练集意思相同但表达不同的数字均转化为汉字数字。

```
terms = {'1届': '一届', '2届': '二届', '3届': '三届', '4届': '四届',
    '5届': '五届', '6届': '六届', '7届': '七届', '8届': '八届',
    '9届': '九届', '10届': '十届', '11届': '十一届',
    '12届': '十二届', '13届': '十三届', '14届': '十四届',
    '15届': '十五届', '16届': '十六届', '17届': '十七届',
    '18届': '十八届', '19届': '十九届', '20届': '二十届',
    '21届': '二十一届', '22届': '二十二届', '23届': '二十三届',
    '24届': '二十四届', '25届': '二十五届'}
for str in strs:
    for term in terms:
        if term in str:
            str.replace(term, terms[term])
```

分词：

首先过滤停用词，（停用词主要是特殊符号和空格）

```
 for str in strs:
    seg_list = jieba.lcut(str, cut_all=True) *#**对一个句子分词*
    for w in seg_list[::-1]:          *#**过滤停用词*
        if w in stopwords:
            seg_list.remove(w)
            continue
```

这里使用了一个小技巧，由于正序索引list变量在使用remove删除某个元素后index指针会跳过当前元素直接指向下一个元素，所以可能会漏删一些停用词，采用倒序索引(seg_list[::-1])可以较好地避免这一问题。

将处理好的句子逐词写入目标txt文件(用来保存分词后的问句训练集)：

```
for w in seg_list:
    outputfile.write("%s " %w)
sentence_list.append(seg_list)                  #形成分词后的句子列表
```

生成句向量：

生成句向量我们采用了三种方式，根据测试的效果最后选用了Bert模型，

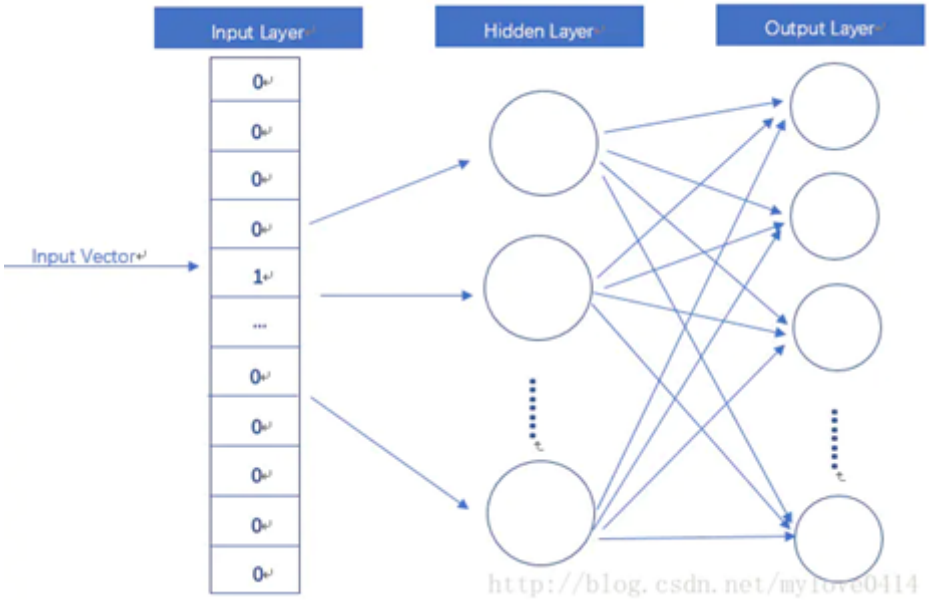
```
'''==================sklearn(词袋模型+TF- IDF)=========================
from sklearn.feature_extraction.text import CountVectorizer
count_vec = CountVectorizer()
Vectors_1 = count_vec.fit_transform(Sentences).toarray()
```

sklearn词袋模型通过单纯地统计词频，得到一个句子向量，非常稀疏，比较的难度较大。不过它可以配合其它算法进行进一步的处理。

```
'''==========================word2vec==============================
from gensim.models import Word2Vec
dim = 600
#预训练词向量模型
model = Word2Vec(Sentences, sg=1, size=dim,  window=5,  min_count=1,  negative=3,
sample=0.001, hs=1, workers=4)
model.save("data/word2vec.model")
#构建文本数据集的特征向量:调用模型索引词向量获取样本中每个词的词向量，直接相加
def Get_Feature(sentences, word_vec, dim):
    m = len(sentences)
    feature = np.zeros((m,dim))
    for i,sentence in enumerate(sentences):
        for w in sentence:
            feature[i,:] += word_vec[w]
    return feature
model = Word2Vec.load("data/word2vec.model")
word_vec = model.wv
Vectors_3 = Get_Feature(Sentences,word_vec,dim)
np.save(Qfeatures_file,Vectors_3)'''
```

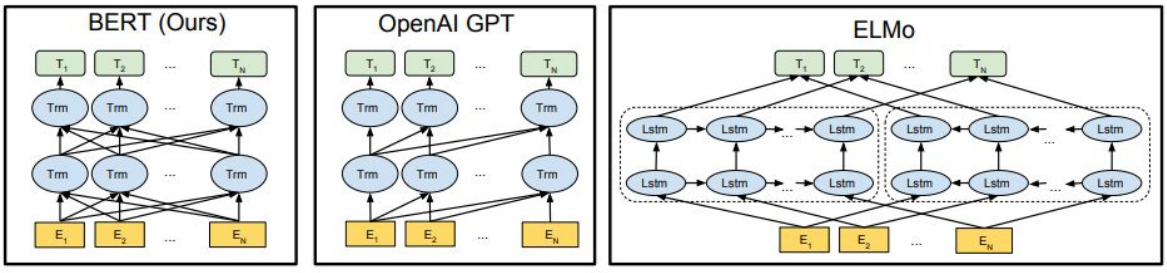
word2vec模型是一个较为成熟的训练模型，它是简化的神经网络。一般采用三层神经网络结构，分为输入层，隐藏层和输出层，如图所示。输入是One-Hot Vector；隐藏层有N个神经元，代表我们想要的词向量的维度，没有激活函数，也就是线性的单元。输出层的神经元个数和输入相同，隐藏层再到输出层时最后需要计算每个位置的概率，使用softmax回归。当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，例如隐藏层的权重矩阵。这个模型是如何定义数据的输入和输出呢？一般分为CBOW(Continuous Bag-of-Words) 与Skip-

Gram两种模型。CBOW模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。Skip-Gram模型输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。




```
"========================Bert模型库=============================="
from bert_serving.client import BertClient
bert_client = BertClient()
Vectors_2 = []
for idx,sentence in enumerate(Sentences):
    Vectors_2.append(bert_client.encode([sentence])) #对每个句子进行特征提取，维度为
768
import numpy as np
features = np.array(Vectors_2)
np.save(Qfeatures_file,features)
```

BERT模型由Google在2018年发布，为NLP领域带来了里程碑式的发展。BERT的全称是Bidirectional Encoder Representation from Transformers，即双向Transformer的Encoder，因为decoder是不能获要预测的信息的。模型的主要创新点都在pre-train方法上，即用了Masked LM和Next Sentence Prediction两种方法分别捕捉词语和句子级别的representation。其模型结构与另外两种著名的模型GPT、ELMO比对如下图所示：



三、模型设计与实现

模型整体使用“检索式对话系统”的思路，以关键词、句子的相似度为指标将测试集的问句与训练集的问句进行匹配，并取与测试集问句“最相似”的问句的答案作为最终的答案来输出。模型设计要点如下：

1. 匹配特殊关键词的数量

由于Bert模型直接将输入的句子进行特征提取，生成句向量，采用的是现有的语料库，偏生活化，所以在冬奥会的专业领域特征提取的结果可能不够准确。为了弥补这一缺陷，我们在匹配含有测试问句中关键词的训练问题库中的问题时加入了优先特殊关键词(包括人名、地名和时间)的策略。

为了减少运算量并减少问题句式的不同对结果造成的影响，我们手动挑选了若干个特殊关键词存在文件里，目前这些特殊关键词为训练集中的：人名、地名和时间。在匹配的最开始，对于问句中的每一个关键词，去tranfer_list中查询包含它的训练集句子，并比对它和问句之间相同的特殊关键词（super_key_num），并维护一个max_super_key_num来表示当前和问句的最大特殊关键词匹配数，同时使用一个list_for_contract存储对应的训练集问句。如果存在多个和问句的特殊关键词匹配数一样的句子，那就会进入下一步的精细选择。具体的实现代码如下：

```
for item in question_sentence:#对于提问的句子中的每一个关键词
        if item not in transfer_list:
            continue
        for idx in transfer_list[item]:#对于关键词对应的句子
            #idx_to_sentence_list是把训练集的idx转换成关键词句子的数组
            seg_sentence = seg_Q[idx].split()
            super_key_num=0
            if item in seg_sentence:
                #specialwords是从存人名、时间、地点的文件里读出来的list
                specialwords = specialword_list()
                if item in specialwords:
                    super_key_num+=1
            if super_key_num>=max_super_key_num:
                if super_key_num>max_super_key_num:
                    max_super_key_num=super_key_num
                    list_for_contract=[]    #存在更大的就清空list
                list_for_contract.append(idx)#存下来人名地名时间匹配数最多的句子的
id
```

首先根据提问句子中的每一个关键词item，去transfer_list内寻找训练集中包含这个关键词item的句子的id（在训练集的Excel内，一个id唯一对应一个训练集的问句）。然后使用seg_sentence这个列表取到idx对应的句子的关键字数组。

之后初始化super_key_num。如果item在对应的训练集问句idx_sentence里，那就继续判断它是否是特殊关键词，如果是特殊关键词的话就将super_key_num加1，在统计完之后，判断当前的super_key_num是否比max_super_key_num大，如果的确更大的话，那就更新max_super_key_num并清空list_for_contract（因为它存储的是super_key_num最大的训练集问句，如果有更大的就需要将其清空）。最后将当前句子的id储存到list_for_contract中。如此得到的list_for_contract便是初次选择的与问句“最像”的句子的集合。

2. 取句向量相似度最大的句子作为结果

在初步选择出了特殊关键词匹配度最高的一些句子后，需要在这些句子中选出和问句“更像”的句子来提供最终的答案。于是我们将问句都转化为特征向量，然后对特征向量采取一些运算并比较后得到最后的结果。具体代码如下：

```
for idx in list_for_contract:
        score=get_score(Qset_vector[idx],q_vector,1)
        #逐个比对得到最大的相似值及其id
        if score>max_similarity:
            max_similarity=score
            result_id=idx
    #匹配度太低就抛弃掉
    if max_similarity < 0.7:
        result_id=-1
```

对于list_for_contract中的每一个句子的id，通过Qset_vector字典得到这id的句子对应的特征向量，然后将提问句的特征向量q_vector一起输入get_score函数得到两个句子的相似度得分，之后循环统计最大的得分max_similarity和对应的训练集句子result_id，之后将result_id返回。在这里我们设计了一个"门槛"来过滤掉无关的句子：如果最大的相似度都低于70%那么认为这个句子不存在答案，也就是返回"暂时没有合适的答案"。

3. 利用余弦函数计算句向量之间的相似度

对于向量来说，最大的相似就是两个向量是平行向量，所以整体来说，两个向量的夹角越小，他们就越相似。所以直接使用两个向量的向量积除以他们的模长积来计算夹角的cos值，并以这个cos值作为两个句向量之间的相似度。具体代码如下：

```
def cosine_distance(v1, v2):
    #计算余弦距离
    up = float(np.sum(v1 * v2))
    down = np.linalg.norm(v1) * np.linalg.norm(v2)
    return try_divide(up, down)
def get_score(vector_1,vector_2,dis_type):
    '''
    计算两个句子匹配度的函数
    ----------
    vector_1和vector_2是两个要比对的句子的特征向量
    dis_type 是选择计算距离的方法（有很多种，可以根据效果调试）
    '''
    if dis_type==1:
        result=cosine_distance(vector_1,vector_2)
    return result
```

四、实验结果与数据分析

首先我们在训练集上进行了测试，经过对错误问题和重复问题的筛选后，两个数据集共有14000个问题。对于训练集的问题，我们程序的准确率接近了99%，只有62个问题回答错误。

对于测试集，我们分析了测试集的内容，对测试集做出了划分。测试集中有86个与训练集不同的问题，其余的问题均与训练集中的相同。因此，我们只要分别算出这两部分的准确率，通过加权就可以知道运行该测试集时，结果的准确度如何。

对于非训练集的86个问题，部分测试结果如下：

```
{
    "question":"第6届冬奥会是在哪个国家举办的",
    "answer":"圣莫里茨\n"
}
{
    "question":"第6届冬奥会是在哪个城市举办的",
    "answer":"加拿大卡尔加里\n"
}
{
    "question":"第6届冬奥会是哪一年举办的",
    "answer":"2014年索契冬奥会\n"
}
{
    "question":"第7届冬奥会是在哪个国家举办的",
    "answer":"科蒂纳丹佩佐\n"
}
{
    "question":"第7届冬奥会是在哪个城市举办的",
    "answer":"加拿大卡尔加里\n"
}
{
    "question":"第7届冬奥会是哪一年举办的",
    "answer":"2014年索契冬奥会\n"
}
{
    "question":"第8届冬奥会是在哪个国家举办的",
    "answer":"圣莫里茨\n"
}
{
    "question":"第8届冬奥会是在哪个城市举办的",
    "answer":"加拿大卡尔加里\n"
}
{
    "question":"第8届冬奥会是哪一年举办的",
    "answer":"2014年索契冬奥会\n"
}
{
    "question":"第9届冬奥会是在哪个国家举办的",
    "answer":"科蒂纳丹佩佐\n"
}
```


可以看出，在对于冬奥会举行的时间、地点（城市、国家）的问题的回答上，我们的模型回答的并不好，这些问题几乎都不对。原因有以下3点：

1. 训练集给出的数词是以汉字给出的，而问题里的量词是以阿拉伯数字形式出现的，这导致我们的模型对冬奥会的届数判断有误；
2. 训练集给的是在哪里举办，而问题问的是国家和城市；
3. 训练集给的是时间，而问题问的是哪一年；

这三点问题导致我们的模型对问题判断失误。这72个问题几乎全错。统计后，我们的模型在86个问题中只对了9个。

根据概率可知，在测试集的1246个问题中，我们的错误大概有12个，而在那86个中，我们错误的有77个。共计错误89个。计算正确率约为95%。

若要改进上述问题，可以通过增加辅助词典和额外数据集针对冬奥会举办的时间、国家、城市进行训练。