人工智能大作业报告

--冬奥会对话系统

组员1 李昊宸 学号2017K8009929044

组员2 李颖彦 学号2017K8009929025

1. 任务描述
2. 赛题背景

智能问答系统（对话系统）的应用是非常普遍的，比如说客服，前台机器人，讲解机器

人等很多场景都可能会用到FAQ问答系统。所谓的FAQ就是 frequently asked questions，

也就是说在某个场景下，算法可以回答一些比较常见的问题。

1. 任务要求

对数据方提供的冬奥会知识数据进行集成，输入端接受以自然语言输入的问题（比如使

用中文询问“中国在索契冬奥会获得金牌数目”），输出端输出该问题对应的回答。

1. 数据描述

数据包有两个：标注数据1和标注数据2。标注数据1中的数据为单层问题，标注数据2中的数据为叠加问题。

1）单层问题

问句：哪一届奥运会的金牌总数最多？

分词：哪 一届 奥运会 的 金牌 总数 最 多 ？

#1问句类型：Which多选一

#1领域类型：Competition比赛

#1语义类型：Calculation计算

#2问句类型：NA

#2领域类型：NA

#2语义类型：NA

2）叠加（双层）问题

问句：中国奥运第一人值得尊敬吗？

分词：中国 奥运 第一 人 值得 尊敬 吗 ？

#1问句类型：Who

#1领域类型：Competition

#1语义类型：Factoid

#2问句类型：Whether

#2领域类型：Competition

#2语义类型：Opinion

1. 实施方案

对于一个完整的对话系统FAQ的构建，第一步要做的是对输入的问题进行预处理。预

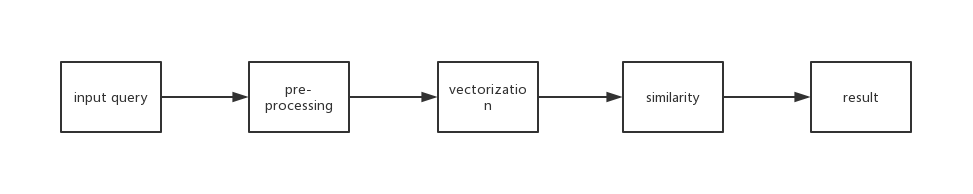
处理需要做的事情主要为删除无用文字，去除停用词，并将问题切分成一个个中文词语。

第二部就是将处理之后的语料进行向量化。常见的向量化方法有词频向量化、word2vec、

tf-idf 等方法。向量化之后，每一个问题对应的都为一个高维向量，当有询问问题输入

时，先将问题预处理、向量化，然后和数据集中的数据进行比对，输出相似度最高的问

题的答案，这就是检索式对话系统的大致框架。

1. 数据预处理

查看测试集，发现有的问题没有答案，所以预处理的第一步就是将没有答案的问题删除。预处理的第二步就是将不属于中文的文本删除（包括各种标点符号）。预处理的第三步是将修正后的文本进行词语的切分，从而将一整段话切分为一个个词语。

1. 向量化

使用CountVectorizer对每一条语料进行词频矩阵的生成，从而完成语料的空间向量化。

1. 相似度比对

将数据库中的语料与输入问题的向量化后的向量进行相似度比对。这里我们采用的是余弦相似度比对算法。比对完成之后，将得分最高的语料的答案返回。

1. 具体实现

以下是整个处理过程的具体实现：

1. 数据清理

对非中文无用数据的清理，需要将以下类别的数据从训练集和测试集中清除：

html  = re.compile('<.\*?>')

    http  = re.compile(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\(\),]|(?:%[0-9a-fA-F][0-9a-f

A-F]))+')

    src   = re.compile(r'\b(?!src|href)\w+=[\'\"].\*?[\'\"](?=[\s\>])')

    space = re.compile(r'[\\][n]')

    ids   = re.compile('[(]["微信"]?id:(.\*?)[)]')

    wopen = re.compile('window.open[(](.\*?)[)]')

    english = re.compile('[a-zA-Z]')

    others= re.compile(u'[^\u4e00-\u9fa5\u0041-\u005A\u0061-\u007A\u0030-\u0039\u3002\uFF1F\uFF01\uFF0C\u3001\uFF1B\uFF1A\u300C\u300D\u300E\u300F\u2018\u2019\u201C\u201D\uFF08\uFF09\u3014\u3015\u3010\u3011\u2014\u2026\u2013\uFF0E\u300A\u300B\u3008\u3009\!\@\#\$\%\^\&\\*\(\)\-\=\[\]\{\}\\\|\;\'\:\"\,\.\/\<\>\?\/\\*\+\\_"\u0020]+')

需要将html链接、数据来源、用户名、英语字符和其他单个非中文字符清除，采用以

上正则表达式描述需要删除的类别，配合sub命令将其从训练集和数据集中删除。由于

本次的数据来源质量较好，该步对源数据的处理很少。

1. 数据划分

数据划分有两种方式：保留停用词和去除停用词。数据划分使用的库为jieba分词，具体的操作如下：

Question.txt和Answer.txt分别存放了问题和答案，预处理后输出到QuestionSeg.txt和AnswerSeg.txt。

inputQ = open('Question.txt', 'r', encoding='gbk')

outputQ = open('QuestionSeg.txt', 'w', encoding='gbk')

inputA = open('Answer.txt', 'r', encoding='gbk')

outputA = open('AnswerSeg.txt', 'w', encoding='gbk')

先来看保留停用词的划分方式：

def segmentation(sentence):

sentence\_seg = jieba.cut(sentence.strip())

out\_string = ''

for word in sentence\_seg:

out\_string += word

out\_string += " "

return out\_string

for line in inputQ:

line\_seg = segmentation(line)

outputQ.write(line\_seg + '\n')

outputQ.close()

inputQ.close()

for line in inputA:

line\_seg = segmentation(line)

outputA.write(line\_seg + '\n')

outputA.close()

inputA.close()

接下来看不保留停用词的划分方式，测试结果表明保留停用词划分效果更好，查阅资料

显示原因为CountVectorizer目前版本对停用词的优化更佳。

def stopword\_list():

stopwords = [line.strip() for line in open('stopword.txt', encoding='utf-8').readlines()]

return stopwords

def seg\_with\_stop(sentence):

sentence\_seg = jieba.cut(sentence.strip())

stopwords = stopword\_list()

out\_string = ''

for word in sentence\_seg:

if word not in stopwords:

if word != '\t':

out\_string += word

out\_string += " "

return out\_string

for line in inputQ:

line\_seg = seg\_with\_stop(line)

outputQ.write(line\_seg + '\n')

outputQ.close()

inputQ.close()

for line in inputA:

line\_seg = seg\_with\_stop(line)

outputA.write(line\_seg + '\n')

outputA.close()

inputA.close()

1. 构建词频矩阵

经过以上两步，我们已经将文本成功分割为独立的中文词语，接下来需要统计每个词出现的频率及分布。

stopwords = [line.strip() for line in open('stopword.txt', encoding='utf-8').readlines()]

首先需要获得停用词表。这里我们使用的是百度停用词表、哈工大停用词表、中文停用词表等多个词表的综合结果。

CountVectorizer是通过fit\_transform函数将文本中的词语转换为词频矩阵，矩阵元素a[i][j] 表示j词在第i个文本下的词频。即各个词语出现的次数，通过get\_feature\_names()可看到所有文本的关键字，通过toarray()可看到词频矩阵的结果。

count\_vec = CountVectorizer()

先对文本内容进行词频统计。需要说明的是，由于代码第一版优化不佳，没有进行对向量化数据的保存，在每次输入问题之后都需要将全体语料进行向量化的重新演算，导致查询时间较长。

1. 余弦相似度比对

余弦相似度量：计算个体间的相似度。

相似度越小，距离越大。相似度越大，距离越小。

假设有3个物品，item1，item2和item3，用向量表示分别为：

item1[1,1,0,0,1]，

item2[0,0,1,2,1]，

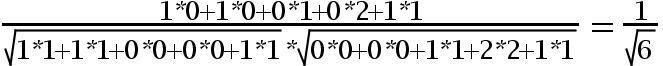
item3[0,0,1,2,0]，

即五维空间中的3个点。用欧式距离公式计算item1、itme2之间的距离，以及item2和item3之间的距离，分别是：

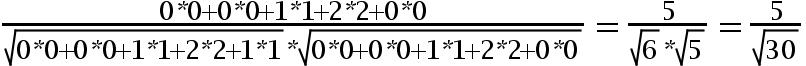
item1-item2=IMG_256

item2-item3=IMG_257

用余弦函数计算item1和item2夹角间的余弦值为：



用余弦函数计算item2和item3夹角间的余弦值为：



由此可得出item1和item2相似度小，两个之间的距离大(距离为7)，item2和itme3相似度大，两者之间的距离小(距离为1)。

余弦相似度算法：一个向量空间中两个向量夹角间的余弦值作为衡量两个个体之间差异的大小，余弦值接近1，夹角趋于0，表明两个向量越相似，余弦值接近于0，夹角趋于90度，表明两个向量越不相似。

基于余弦相似度算法，我们将输入的问题的向量与数据库中的语料的向量进行一一比对，输出相似度最高（最接近1）的语料的答案。

余弦相似度计算函数：

def count\_cos\_similarity(vec\_1, vec\_2):

if len(vec\_1) != len(vec\_2):

return 0

s = sum(vec\_1[i] \* vec\_2[i] for i in range(len(vec\_2)))

den1 = math.sqrt(sum([pow(number, 2) for number in vec\_1]))

den2 = math.sqrt(sum([pow(number, 2) for number in vec\_2]))

return s / (den1 \* den2)

计算两个语句的余弦相似度

def cos\_sim(sentence1, sentence2):

sentences = [sentence1, sentence2]

vec\_1 = count\_vec.fit\_transform(sentences).toarray()[0] #输入问题向量化

vec\_2 = count\_vec.fit\_transform(sentences).toarray()[1] #语料库向量化

return count\_cos\_similarity(vec\_1, vec\_2)

1. 交互界面的实现

def get\_answer(sentence1):

sentence1 = segmentation(sentence1)

score = []

for idx, sentence2 in enumerate(open('QuestionSeg.txt', 'r')):

# print('idx: {}, sentence2: {}'.format(idx, sentence2))

# print('idx: {}, cos\_sim: {}'.format(idx, cos\_sim(sentence1, sentence2)))

score.append(cos\_sim(sentence1, sentence2))

if len(set(score)) == 1:

print('暂时无法找到您想要的答案。')

else:

index = score.index(max(score))

file = open('Answer.txt', 'r').readlines()

print(file[index])

while True:

sentence1 = input('请输入您需要问的问题(输入q退出)：\n')

if sentence1 == 'q':

break

else:

get\_answer(sentence1)

1. 实验结果及分析
2. 数据集划分方式

为了提高问答系统的性能，这次实验中我们没有对数据集进行分割，而使将全部的问题都用来训练模型。

1. 测试结果

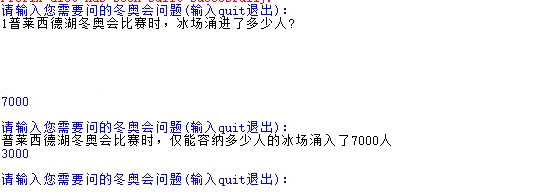
我们针对几个角度刁钻的问题进行测试，结果如下：

针对高相似度问题的测试：

原问题及答案及测试结果截图：

1932普莱西德湖冬奥会比赛时，仅能容纳3000人的冰场，一下子涌进了多少人?

答案：7000



由此可见，检索系统对于相似度很高的问题之间仍然可以做出明确的区分。

部分缺失问题的提问：

原问题及答案及测试结果截图：

1932普莱西德湖冬奥会美、加两队比赛时，第一场美国队以l：多少败北?

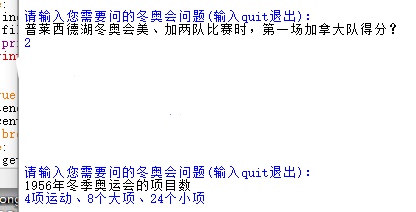
答案：2

1956年科蒂纳丹佩佐冬季奥运会的项目数是什么?

答案：4项运动、8个大项、24个小项

1964年因斯布鲁克冬奥会第一次参赛的有什么?

答案：朝鲜民主主义人民共和国、印度和蒙古

屏幕截图(11)

以上样例分别从原问题中去除了部分不同部位的主体信息，检索器的表现良好。由此可见，检索系统对于部分主体信息缺失的提问仍然可以做出准确的回答。

1. 性能优化

经测试，由于余弦相似度算法的局限性，每一次提问的问题都需要和全体语料库进行向量化，这导致单个问题的提问会有较长时间的预处理时间。为了优化性能，我们进行了第二版处理方式，即一次向对话系统询问大量的问题（我们用到的测试用例为1000个记录在txt文件中的问题，用换行符隔开），发现总计回答时间与询问单个问题的时间相近，原因在于询问1000个问题可以只进行1次全体向量化，避免了单次询问每次都要做类似的工作。优化之后，问题回答时间可以满足题目要求（甚至平均值远低于500ms）。

1. 总结展望

总的来说，基于余弦相似度比对的算法能够达到较高的分词精度，对多重问题的识别和

相似问题的区分也达到了比较好的效果。

但是由于检索式对话系统自身的缺陷，对向量化结果的查询必然遍历全体语料库，故有

着先天的速度劣势。

其次，检索式与生成式相比的弊端在于回答问题的能力较为单一，只能回答预先设计好

的答案，而不能主动生成答案。

再之，基于word2Vec思路下的语义比对能实现的分类

精度大致不会有更大的提高，查阅相关资料表明这与word2Vec不基于上下文的词义分

类有关。如果使用Bert进行词语的训练或许会有更大的突破。