**冬奥会领域问答机器人实验报告**

吴俊亮 贾志杰 孙严顺 陈天宇

一、实验任务

本次实验旨在实现一个冬奥会领域的问答系统，即当输入一个有关冬奥会的问题，系统给出相应的答案。要求：输入输出以json格式；单次响应时间在500ms以内（除去初始加载模型的时间）；我们将数据集进行8/2的划分，划分为训练数据和测试数据并返回正确答案的可接受率。

二、实验设计

（一）实验原理

我们的总体实验设计思路是使用python的gensim库实现TF-IDF算法以判断训练语料和测试语料的相似度，进而选择相似度最高的训练语句的对应答案为相应测试语句的答案。

1. gensim库

gensim 是一个通过衡量词组（或整句、文档）模式来挖掘文档语义结构的工具。gensim

有三大宏观概念：语料→向量→模型：gensim读取一段语料，输出一个词向量来表示文档中的一个词，词向量进而可以用来训练各种分类器模型。

语料：语料是指一组文档的集合。这个集合是gensim的输入，gensim会从这个语料中推断出它的结构，主题等。在本次实验中语料即为训练数据集，收集语料之后，需要做一些预处理，我们这里的预处理比较简单，就是使用jieba库把语句分成词组。

向量：如果要对文档的隐含结构进行推断，就需要使用适当的数学模型：我们在这里使用的方法是doc2bow，也就是将文档转化为bag-of-words（词袋）。在词袋向量中，每篇文档表示被表示成一个向量，代表字典中每个词出现的次数。例如，给定一个包含[‘2020’,‘冬奥会’,‘举办地’, ‘金牌’]的字典，一个包含[‘2020 冬奥会 2020’]字符串的文档可以表示成向量[2,1,0,0]。向量各个元素的位置和字典的顺序相对应，向量的长度就是字典的长度。词袋模型的一个重要特点是，它完全忽略的单词在句子中出现的顺序，这也就是“词袋”这个名字的由来。

模型：现在我们已经向量化了语料，接下来可以使用各种模型了。我们可以认为模型转换是在两个向量空间进行转换。以本次实验为例，我们通过TF-IDF模型把词袋向量转换到另一个向量空间再进行比较，关于TF-IDF模型将在下面详细介绍。

1. TF-IDF模型

TF-IDF是一种针对关键词的统计分析方法，用于评估一个词对一个文件集或者一个语料库的重要程度。一个词的重要程度跟它在文章中出现的次数成正比，跟它在语料库出现的次数成反比。这种计算方式能有效避免常用词对关键词的影响，提高了关键词与文章之间的相关性。

其中TF（Term Frequency）指的是某词在文章中出现的总次数：

TF = 某词在文档中出现的次数/文档的总词量

TF可以反应一个词在文档中出现的频率但是并不能反应出这个词的重要程度，比如以我们的实验举例：“冬奥会”和“的”的频率可能是一样的，但是“冬奥会”的重要程度显然高于“的”。所以还需要一个重要性调整系数，衡量一个词是不是常见词。如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，正是我们所需要的关键词。IDF（Inverse Document Frequency）指的是逆向文档频率，包含某词语的文档越少，IDF值越大，说明该词语具有很强的区分能力：

IDF = log（语料库中文档总数/包含该词的文档数+1）

知道了"词频"（TF）和"逆文档频率"（IDF）以后，将这两个值相乘，就得到了一个词的TF-IDF值。某个词对文章的重要性越高，它的TF-IDF值就越大。

（二）重要模块分析

1、随机划分数据集

# 随机划分数据集

print("> 正在随机划分数据集")

train = []

test = []

with open(data\_filepath, encoding='utf-8') as original\_file:

    data = json.load(original\_file)

    for value in data:

        rand\_num = random.uniform(0, 10)

        if rand\_num < 8.0:

            train.append(value)

        else:

            test.append(value)

print("  训练集：" + str(len(train)) + "  测试集" + str(len(test)))

with open(train\_filepath, 'w', encoding='utf-8') as train\_file:

    train\_file.write(json.dumps(train, ensure\_ascii=False))

with open(test\_filepath, 'w', encoding='utf-8') as test\_file:

    test\_file.write(json.dumps(test, ensure\_ascii=False))

with open(train\_filepath, encoding='utf-8') as train\_file:

    data = json.load(train\_file)

2、生成分词结果

# 生成分词结果

print("> 正在分词")

content = []

for value in data:

    question = value['question']

    content.append(split\_word(question))

with open(splitdata\_filepath, 'w', encoding='utf-8') as f:

    f.write(json.dumps(content, ensure\_ascii=False))

通过split\_word函数调用jieba库，实现了列表结构的分词，再生成分词后的json格式文件存储在splitdata.json中。

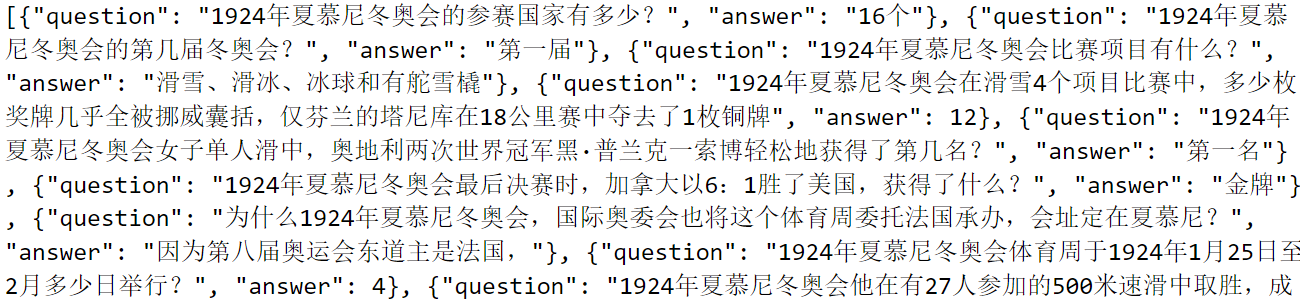


图1 test\_data.json中的测试原始数据

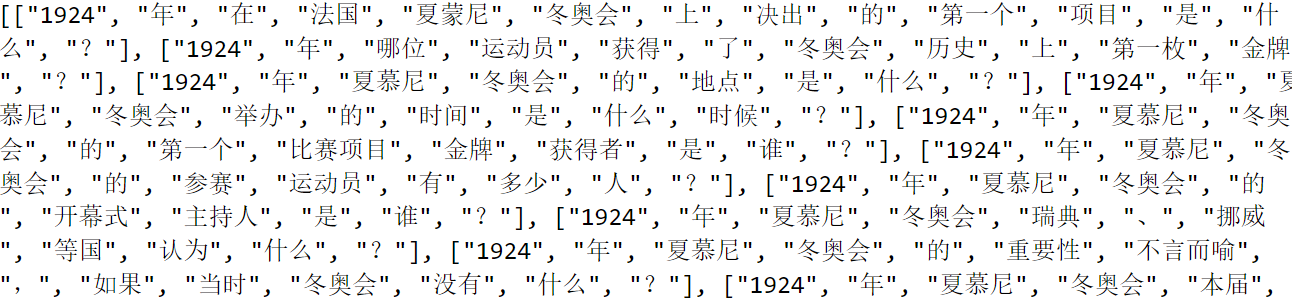


图2 splitdata.json中的原始数据分词后结果

1. 生成gensim字典

# 生成gensim字典

print("> 正在生成gensim字典")

dictionary = gensim.corpora.Dictionary(content)

dictionary.save(dictionary\_filepath)

num\_features = len(dictionary)  # 特征数

使用gensim.corpora.Dictionary函数生成字典并且保存在dictionary\_filepath中，并且在此处设置num\_features变量用于保存字典长度，在后面的相似度比较中会用到。

1. 生成TF-IDF模型

# 生成tfidf模型

print("> 正在生成tfidf模型")

corpus = [dictionary.doc2bow(line) for line in content]  # 语料转词袋表示

tfidf = gensim.models.TfidfModel(corpus)  # 构建tfidf模型

tfidf.save(model\_filepath)  # 保存tfidf模型

通过第三行将语料中每一篇文档转化为对应的向量（这里是bow向量），进而构建并保存TF-IDF模型。

1. 生成TF-IDF相似度比较序列

# 生成tfidf相似度比较序列

print("> 正在生成tfidf相似度比较序列")

index = gensim.similarities.Similarity(index\_filepath, tfidf[corpus], num\_features)  # 文本相似度序列

index.save(index\_filepath)

通过gensim.similarities.Similarity函数生成相似度比较序列，该函数的三个参数分别为文件地址，TF-IDF向量化的语料库，以及字典的文本数量（也就是之前生成的字典长度）。

1. 导入测试集进行测试

# 导入测试集进行测试

with open(test\_filepath, encoding='utf-8') as test:

    test\_data = json.load(test)

print("> 正在使用测试集测试，共有" + str(len(test\_data)) + "个问题")

right = 0

output\_list = []

for value in test\_data:

    question = value['question']

    sentences = split\_word(question)     # 分词

    vec = dictionary.doc2bow(sentences)  # 转词袋表示

    sims = index[tfidf[vec]]             # 相似度比较

    sorted\_sims = sorted(enumerate(sims), key=lambda x: x[1], reverse=True)  # 逆序

    i = sorted\_sims[0][0]

    output\_list.append({'question:':value['question'], 'answer':data[i]['answer']})

    if(str(data[i]['answer']) == str(value['answer'])):

        right = right + 1

with open(output\_filepath, 'w', encoding='utf-8') as output:

    output.write(json.dumps(output\_list, ensure\_ascii=False))

print("  正确率至少为：" + str(float(right/len(test\_data)) \* 100) + "%")

1. 用户继续输入问题

# 允许用户继续输入问题

sentences = input('Question: ')

while True:

    sentences = split\_word(sentences)    # 分词

    vec = dictionary.doc2bow(sentences)  # 转词袋表示

    sims = index[tfidf[vec]]             # 相似度比较

    sorted\_sims = sorted(enumerate(sims), key=lambda x: x[1], reverse=True)  # 逆序

    i = sorted\_sims[0][0]

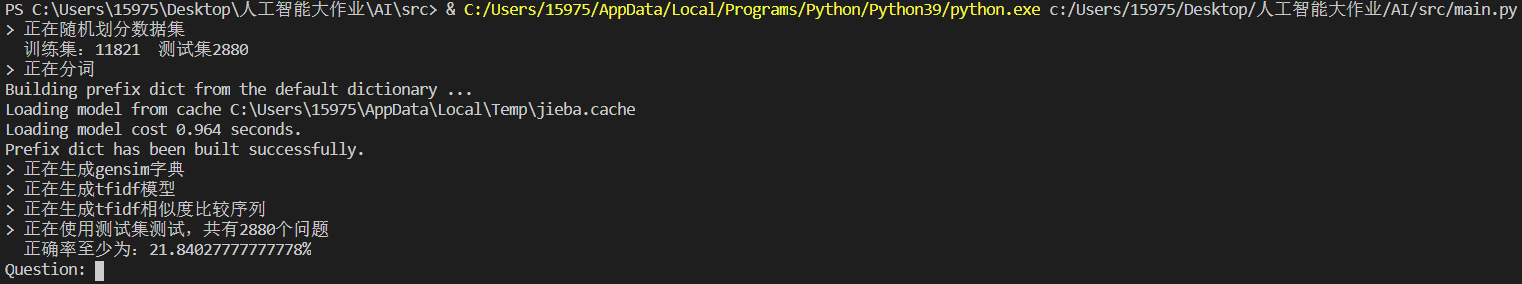
    print(data[i]['answer'])

    sentences = input('Question: ')

首先对用户输入问题进行分词、转词袋表示、相似度比较，得到相似度比较序列，之后将相似度比较序列逆序，逆序后列表的第一位即为测试集中与输入问题相似度最高的问题，系统输出该问题对应答案。

1. 实验结果

在命令行中运行问答机器人的输出如图3所示，完成测试集的测试后，用户可以继续在命令行中向问答机器人提问。



|  |  |  |  |
| --- | --- | --- | --- |
| 次数 | 训练集问答对数量 | 测试集问答对数量 | 测试集中的正确率 |
| 1 | 11821 | 2880 | 21.84% |
| 2 | 11660 | 3041 | 21.28% |
| 3 | 11738  8 | 2963 | 20.59% |
| 4 | 11723 | 2978 | 19.68% |
| 5 | 11831  2870 | 2870 | 21.15% |

1. 组内分工

|  |  |
| --- | --- |
| 成员 | 分工 |
| 吴俊亮 |  |
| 贾志杰 |  |
| 孙严顺 |  |
| 陈天宇 | 实验报告、环境配置、参考资料 |

1. 参考文献

Gensim：

<https://blog.csdn.net/duinodu/article/details/76618638>

TF-IDF：

https://zhuanlan.zhihu.com/p/94446764

## [gensim中similarities.Similarity用法](https://www.cnblogs.com/duoba/p/11337188.html)：

https://www.cnblogs.com/duoba/p/11337188.html