

# 哈 尔 滨 工 业 大 学

## <<信息检索>>

### 实验报告

(2019 年度春季学期)

姓名：	董雄
学号：	1160300415
学院：	计算机学院
教师：	张宇老师

## 目录

实验二： 问答系统设计与实现.....	1
一、实验目的（摘抄自实验指导） .....	1
二、实验内容（大部分摘抄自实验指导） .....	1
三、实验过程和结果.....	2
1.文本集合进行处理、建立索引 .....	2
A.使用倒排索引 .....	2
B.使用 Whoosh 建立索引 .....	3
C.两种索引方式的比较.....	3
2. 问题分类 .....	4
A 特征的提取： .....	4
B 特征的编码： .....	5
C 使用机器学习方法进行训练 .....	6
D 使用测试集进行测试 .....	8
3. 候选答案句排序 .....	8
4. 答案抽取 .....	10
四、实验心得 .....	12

## 实验二： 问答系统设计与实现

### 一、实验目的（摘抄自实验指导）

本次实验目的是对问答系统的设计与实现过程有一个全面的了解。实验主要内容包括：对给定的文本集合进行处理、建立索引；找出问题的候选答案句并排序；答案抽取，逐步调优

### 二、实验内容（大部分摘抄自实验指导）

#### 3.1 文本集合进行处理、建立索引

任务描述：本实验提供的文本集合保存在 `passage.json` 中，每行是一个标准格式的 json 文件，包含文档内容和 id。同学们需要对所有文档分词、分句（已使用上个实验中介绍的 LTP 进行分词、分句操作），并建立索引，作为问答系统的检索语料。检索系统的实现没有具体要求，可以使用开源库（例如 Whoosh），也可以自己实现（自己实现的检索系统有加分）。同学们可以根据后面的任务，自己设计合适的建立索引的方法。另外，同学们可以使用有标注的 `train.json` 数据，来检验自己检索系统的准确性。

#### 3.2 问题分类

任务描述：问题分类是问题理解中的重要部分，问题类别信息对答案抽取有很大帮助。该小节实验期望同学们能够训练一个问题分类模型，得到问题类别信息，然后将其融入到候选答案句排序和答案抽取的任务中，以取得更好的效果（实际上不一定会提高效果，这个取决于问题类别信息的准确性，以及使用问题类别信息的方法）。同学们可以通过对照实验来分析问题类别信息带来的收益。该部分实验必须使用机器学习的方法实现。

问题分类使用的数据是 `train_questions.txt` 和 `test_questions.txt`，分类介绍见“哈工大 IR 研究室问题分类体系 ver2.doc”，由于使用的问答数据集上没有问题类别的标注，因此无法用于训练问题分类任务。本实验提供了额外的问题分类数据集（`train_questions.txt` 和 `test_questions.txt`，介绍见“哈工大 IR 研究室问题分类体系 ver2.doc”），用于训练问题分类任务，训练得到的模型可用来获取问答数据集中的问题类别信息。关于如何将问题类别信息使用到后续任务中，需要同学们自己设计

#### 3.3 候选答案句排序

任务描述：在该实验中，同学们需要从 `test.json` 中读取所有的问题，并通过检索系统来得到每个问题的相关文档（可以是一个或多个，至多不超过 3 个），相关文档中所有的句子称为候选答案句。然后同学们需要对所有候选答案句按照其包含正确的可能性进行排序，可能性越大的越靠前。将排序后的结果保存，用于后面答案抽取的任务。同样的，同学们可以根据有标注的 `train.json` 数据来检验自己候选答案句排序系统的有效性。

候选答案句排序（也叫候选答案句抽取）任务在问答系统中是一个经典任务，主流方法是采用机器学习、深度学习的方法进行文本分类。本实验要求同学们必须使用机器学习的方法完成，通过结合老师上课讲过的文本特征，以及查阅相关文献，设计模型并完成实验。

### 3.4 答案抽取

终于到了最关键的一步，前面我们所有的实验都是为这次实验做数据准备。在前述实验中，我们已经得到了 test.json 中问题的相关文档，已排序的候选答案句，本次实验的目的是从候选答案句中抽取精简的答案片段。例如对于问题“姚明出生于哪一年？”，其最相关的候选答案句为“姚明，前中国男篮国家队队员，生于 1980 年 9 月 12 日”，那么我们期望的答案片段为“1980 年”。答案抽取任务有一定难度，本实验要求同学们使用机器学习的方法完成，通过提取候选答案句的特征，例如 word2vec 词向量、词性标注、命名实体识别、句法树等，来构建分类器，对候选答案句中每个词进行分类。这只是一最最简单的思路，同学们可以通过查阅文献来找到更好的解决方案

## 三、实验过程和结果

### 1.文本集合进行处理、建立索引

#### A.使用倒排索引

针对文本集合进行处理，首先想到的是使用建立倒排索引的方法。倒排索引是把关键字作为键值，并通过关键字来查询带有此关键词的文档 ID 列表的方法。

所以，我们根据以下概念建立倒排索引：

单词词典(Lexicon)：搜索引擎的通常索引单位是单词，单词词典是由文档集合中出现过的所有单词构成的字符串集合，单词词典内每条索引项记载单词本身的一些信息以及指向“倒排列表”的指针。

倒排列表(PostingList)：倒排列表记载了出现过某个单词的所有文档的文档列表及单词在该文档中出现的位置信息，每条记录称为一个倒排项(Posting)。根据倒排列表，即可获知哪些文档包含某个单词。

倒排文件(Inverted File)：所有单词的倒排列表往往顺序地存储在磁盘的某个文件里，这个文件即被称之为倒排文件，倒排文件是存储倒排索引的物理文件。

本次实验中，建立倒排索引的步骤如下：

**Step1** 首先构建单词词典，对文档进行分词，去停用词处理，然后将当前文档的 pid 存储在当前单词的倒排列表中，同时还要记录该单词出现的频率信息(TF)，方便之后进行查询时的答案相似度计算。“文档频率信息”代表了在文档集合中有多少个文档包含某个单词，之所以要记录这个信息，其原因与单词频率信息一样，这个信息在搜索结果排序计算中是非常重要的一个因子。

建立倒排索引的关键代码如下：

```
def create_new_index(filename):
    file_index = {}
    dic_list = read_json_data(filename)
    for dic in dic_list:
        pid = dic['pid']
        for sentence in dic['document']:
            for word in sentence:
                if word not in file_index:
                    file_index[word] = {pid: 1}
                else:
                    if pid not in file_index[word]:
                        file_index[word][pid] = 1
                    else:
                        file_index[word][pid] += 1
    return file_index
```

Step2 接下来处理之后文档的过程中，如果发现单词已经在之前出现过，就将此文档编号、单词出现频率等信息添加在之前对应单词的文档列表中。一直执行直到处理到最后一篇文档。

使用倒排索引进行查询的过程如下：输入一个句子，先对其进行分词，然后对查询的每一个词，获得一个倒排列表。对每个文档进行打分，取得分最高的作为结果返回。

## B.使用 Whoosh 建立索引

在建立倒排索引之后，由于对于其性能的好坏没有统一的评价标准，就基于 Whoosh 建立了另一个索引。

Step1 定义索引的数据类型：

```
schema = Schema(pid=NUMERIC(stored=True),document=TEXT(stored=True, analyzer=analyzer))
```

其中 TEXT 中存储的就是文档内容，analyzer 为传入的中文分词器。

Step2 向建立的索引中添加文档信息，在这里,文档将自动被分词。

```
for dic in distros_dict:
    writer.add_document(
        pid=dic['pid'],
        document=dic['document']
    )
```

进行查询的大致流程如下：先对问题进行分词处理，然后对每一个词进行查询，如果在某篇文档中查到了一个词，就对该文档加分 1，然后处理下一个词，所有的词都查询完后，返回得分最高的文档编号。

## C.两种索引方式的比较

在训练集上对 Top1 和 Top3 进行比较，得到以下结果：

倒排索引：

Top 3: 0.8124065769805681

Top 1: 0.7180493273542601

Whoosh 索引：

Top 3: 0.9124568221678321

Top 1: 0.8546854679532548

虽然使用 Whoosh 建立索引比较简单而且准确率比较高。但是 Whoosh 索引建立时耗费时间更多，查询时也需要更多的时间(因为 Whoosh 相当于是正向索引)。考虑到 Whoosh

的准确率更高，而且多耗费的时间也在可以接受的范围之内，所以在之后的系统中使用 Whoosh 进行索引的建立。

## 2. 问题分类

问题分类采用的是基于机器学习的方法。

### A 特征的提取：

词袋模型：简单来说，词袋模型就是把句子中的每个词当做这个句子的特征。

对词袋特征进行提取，首先要建立词典，并对每个词进行编号，之后的过程将基于这个编号进行。设词典的长度为  $N$ ，然后对于每个问题句，都表示为一个基数为  $N$  的向量，如果某个词出现了，就在词典中查询得到其标号  $M$ ，然后在该词向量的第  $M$  个分量上将值设为 1，最后，在某些机器学习的包中，要求将值为 0 的分量省略。至此，词袋模型提取完成。

```
while line:
    document = re.split('\t',line)
    cutword_result = jieba.lcut(document[1])
    label = document[0]
    feature_str = {}
    for word in cutword_result:
        if word != '\n':
            if word_dic.get(word.strip()):
                feature_str[word_dic[word.strip()]] = word.strip()
    L = sorted(feature_str.items(),key=lambda item:item[0])
    feature = ''
    for l in L:
        feature += str(l[0]) + ":" + str(1)+" "
    label_value = 0

    ...

    if label == "DES_OTHER":
        wf.write("+1" + " " +feature+"\n")
    else:
        wf.write("-1" + " " +feature+"\n")
    ...

    if label_dic.get(label):
        label_value = label_dic[label]
        wf.write(str(label_value) + " " +feature+"\n")
    else:
        n = n+1
        label_value = n
        label_dic[label] = n
        wf.write(str(label_value) + " " +feature+"\n")
    line = f.readline()
```

以上为提取词袋模型的代码。

词性标注特征：

词性特征属于词的语法范畴。在英语中，一般分为十类：即名词、动词、形容词、代词、数词、冠词、副词、介词、连词、感叹词。由于英语是形合语言，其词性是根据词义、句法作用和形式特征来划分的，因此，词性在英语句子的理解中具有非常重要的作用。汉语是意合语言，相对于英语，汉语中的词性作用不是那么明显，一个词的词性并不是与其词义、句

法作用和形式特征严格地对应。这也是为什么我们平常说话造句的时候不能简单地依靠其词性，而需要确切地知道所用词意思的一个非常重要的原因。

Ltp 对问题进行分词、词性标注，然后和建立词袋模型的过程相似，建立关于词性的特征向量。

#### 词袋绑定特征：

实际上，在一个句子中，词性、词义等特征都是对应于某个特定的词。因此，可以将词性、词义和命名实体等特征分别看作问句中特定词的某个属性，并与对应的词绑定在一起形成一类新的问句特征。将上面例句中的词性、词义和命名实体等特征分别绑定到特定的词以后，所得的新特征如图所示：



由于词性特征本来就是对应于问句中每个特定词的，因此可以将它看作问句中所对应词的重要属性，并通过与该词进行绑定以形成一类新的问句特征即。例如，利用 ltp 平台对上面例句进行词性标注以后，所得结果如下：

{黑龙江/ns, 的/u, 省会/n, 在/p, 哪个/r, 城市/n, ? /wp}

将词袋绑定特征作为训练材料进行模型的训练、对结果进行预测，得到的结果并不理想。主要原因是没有充分利用所获取的词袋绑定特征。在分别获取问句中的词袋、词性、词义、命名实体、依存关系以及核心词等基本特征及其词袋绑定特征以后，通过将基本特征和词袋绑定特征进行融合，以形成更加丰富和有效的问句特征集合。

具体的融合过程如下：

**步骤 1.** 选择若干基本特征或词袋绑定特征，取分类精度较高的基本特征集合和词袋绑定特征集合；

**步骤 2.** 将所得基本特征集合和词袋绑定特征集合中的所有特征放在一起（相同的特征只取一个）构成新的问句特征集合；

**步骤 3.** 从该特征集合中每次去掉一个特征，如果分类精度上升，则说明去掉的那个特征在原来的特征集合中起抑制作用；如果分类精度下降，则说明去掉的那个特征在原来的特征集合中起促进作用。

**步骤 4.** 从新的问句特征集合中去掉所有起抑制作用的特征，如果分类精度上升，则对应的特征集合即为最终的问句特征集合；如果分类精度下降，则将步骤中去掉单个特征以后分类精度最大的那个特征集合作为最终问句特征集合。

## B 特征的编码：

本次实验使用的机器学习算法是 LibSvm，所以要根据其要求的数据格式对特征进行编码。

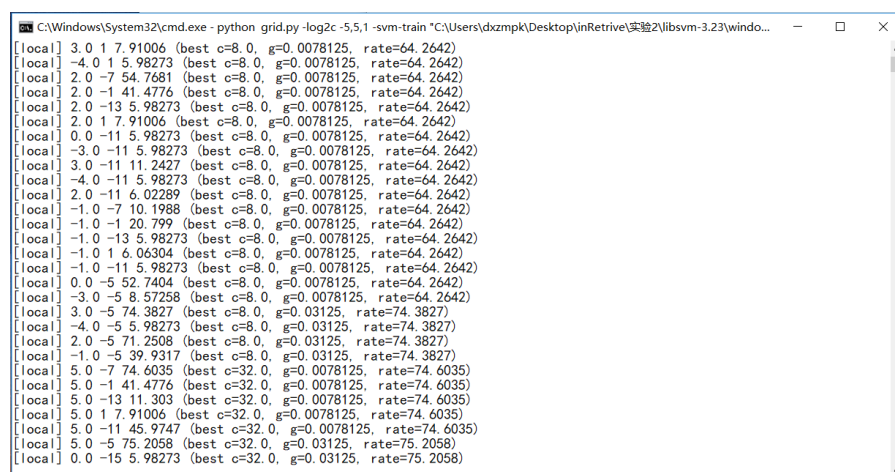
对特征进行编码就是将经过分词后的问题中的特征表示成分类器可以接受的数据形式。本文首先将分词后所得到的所有词进行排列并剔除重复词组成待用词库。本文是基于分类器进行中文问题分类的，所接受的特征向量格式为：

<label> <index1>:<value1> <index2>:<value2> ... <index n>:<value n>

这里的为问题类别序号，为特征项的编号，为特征项的对应取值。实际构建特征向量时，首先根据某个特征的所有取值构造对应该特征的一个特征词典，该词典中每一行对应一个特征项及其取值，行号表示该特征项的编号，对于特征项的取值采用布尔编码的方式，即该特征项出现则为，否则为。例如，对于词袋特征，通过扫描它在对应特征词典中的位置作为其在特征向量中的并将其值赋为，就可以将一个问题转化为一个特征向量的形式。对于多个特征的编码，每个特征的特征项的取值方式完全相同，即均为布尔编码，但在对每个特征的特征项进行标号时要使得第二个特征的第一个特征项的编号要紧接着第一个特征的最后一个特征项的编号依次进行，第三个特征的第一个特征项的编号要紧接着第二个特征的最后一个特征项的编号依次进行，以此类推。例如，若词的特征词典有  $m$  个特征项，词性的特征词典有  $n$  个特征项，则第一个特征（词袋特征）的每个特征项的编号依次为  $1, 2, \dots, m$ ，而第二个特征（词性特征）的每个特征项的编号依次为  $m+1, m+2, \dots, m+n$ 。

## C 使用机器学习方法进行训练

首先对模型进行训练，为了获得 LibSvm 的参数，参照官网给出的方法，对模型进行迭代训练， $C$  是惩罚系数，就是你对误差的宽容度这个值越高，说明你越不能容忍出现误差  $\gamma$  是选择径向基函数作为 kernel 后，该函数自带的一个参数。隐含地决定了数据映射到新的特征空间后的分布。过程如图所示，得到最优参数  $c=32.0, \gamma=0.03125$ ，此时交叉验证的正确率达到了 75.2058。



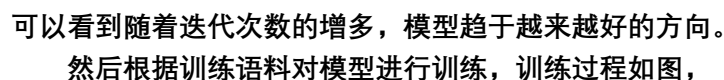
```

[local] 3.0 1 7.91006 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -4.0 1 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 2.0 -7 54.7681 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 2.0 -1 41.4776 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 2.0 -13 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 2.0 1 7.91006 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 0.0 -11 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -3.0 -11 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 3.0 -11 11.2427 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -4.0 -11 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 2.0 -11 6.02289 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -1.0 -7 10.1988 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -1.0 -1 20.799 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -1.0 -13 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -1.0 1 6.06304 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -1.0 -11 5.98273 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 0.0 -5 52.7404 (best c=8.0, g=0.0078125, rate=64.2642)
[local] -3.0 -5 8.57258 (best c=8.0, g=0.0078125, rate=64.2642)
[local] 3.0 -5 74.3827 (best c=8.0, g=0.03125, rate=74.3827)
[local] -4.0 -5 5.98273 (best c=8.0, g=0.03125, rate=74.3827)
[local] 2.0 -5 71.2508 (best c=8.0, g=0.03125, rate=74.3827)
[local] -1.0 -5 39.9317 (best c=8.0, g=0.03125, rate=74.3827)
[local] 5.0 -7 74.6035 (best c=32.0, g=0.0078125, rate=74.6035)
[local] 5.0 -1 41.4776 (best c=32.0, g=0.0078125, rate=74.6035)
[local] 5.0 -13 11.303 (best c=32.0, g=0.0078125, rate=74.6035)
[local] 5.0 1 7.91006 (best c=32.0, g=0.0078125, rate=74.6035)
[local] 5.0 -11 45.9747 (best c=32.0, g=0.0078125, rate=74.6035)
[local] 5.0 -5 75.2058 (best c=32.0, g=0.03125, rate=75.2058)
[local] 0.0 -15 5.98273 (best c=32.0, g=0.03125, rate=75.2058)

```

对参数进行优化的过程中，使用 Gnuplot Graph 对两个参数  $c$  和  $\gamma$  对结果的影响进行绘图，得到





其中，#iter 为迭代次数，nu 与前面的操作参数-nv 相同，obj 为 SVM 文件转换后的二次规划求解得到的最小值，rho 为判决函数的常数项 b，nSV 为支持向量个数，nBSV 为边界上的支持向量个数，Total nSV 为支持向量总个数。

模型文件：

- 7 -

## D 使用测试集进行测试

```
C:\Users\dxzmpk\Desktop\inRetrive\实验2\libsvm-3.23\windows>svm-predict test_features.txt great.model testClassification
.result
Accuracy = 70.9506% (933/1315) (classification)
```

得最后问题分类正确率为 70.95%，基本满足问答系统对于问题分类的精度要求。

## 3. 候选答案句排序

这里使用 BM25 算法对问句和答案句的相关度进行打分。BM25 算法是一种常见用来做相关度打分的公式，思路比较简单，主要就是计算一个 query 里面所有词和文档的相关度，然后在把分数做累加操作，而每个词的相关度分数主要还是受到 tf/idf 的影响。公式如下：

$$Score(Q, d) = \sum_i^n W_i \cdot R(q_i, d)$$

$R(q_i, d)$  是每个词和文档的相关度值，其中  $q_i$  代表每个词， $d$  代表相关的文档， $W_i$  是这个词的权重，然后所有词的乘积再做累加。

$W_i$  是第  $i$  个词的权重，默认的话是 idf 值，公式如下， $N$  是文档总数， $n(q_i)$  是包含该词的文档数，0.5 是调教系数，避免  $n(q_i)$  为 0 的情况，从这个公式可以看出  $N$  越大， $n(q_i)$  越小的花 idf 值越大，这也符合了“词的重要程度和其出现在总文档集合里的频率成反比”的思想，取个 log 是为了让 idf 的值受  $N$  和  $n(q_i)$  的影响更加平滑。

$$IDF(q_i) = \log \frac{N + 0.5}{n(q_i) + 0.5}$$

第二项的计算公式为：

$$R(q_i, d) = \frac{f_i(k_1 + 1)}{f_i + K} \cdot \frac{qf_i(k_2 + 1)}{qf_i + k_2}$$

其中， $k_1, k_2, b$  都是调节因子，和上面的 0.5 是一个作用

$qf_i$  表示  $q_i$  在查询 query 中出现的频率， $f_i$  表示  $q_i$  在文档  $d$  中出现的频率，因为在一般的情况下， $q_i$  在查询 query 中只会出现一次，因此把  $qf_i=1$  和  $k_2=1$  代入上述公式中，后面一项就等于 1，最终可以得到：

$$R(q_i, d) = \frac{f_i(k_1 + 1)}{f_i + K}$$

$K$  也是一个公式的缩写，即：

$$K = k_1 \cdot (1 - b + b \cdot \frac{dl}{avg(dl)})$$

在这里， $dl$  代表文档的长度， $avg(dl)$  是文档的平均长度， $b$  是调节因子

在文章长度比平均文章长度固定的情况下，调节因子  $b$  越大，文章长度占有的影响权重就越大，反之则越小。在调节因子  $b$  固定的时候，当文章的长度比文章的平均长度越大，则  $K$  越大， $R(q_i, d)$  就越小。我们把  $K$  的展开式带入到 bm25 计算公式中去，得到最终的 BM25 计算

公式：

$$Score(Q, d) = \sum_i^n W_i \cdot R(q_i, d) = \sum_i^n W_i \cdot \frac{f_i(k_1 + 1)}{f_i + K}$$

$$= \sum_i^n IDF(q_i) \cdot \frac{f_i(k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \frac{dl}{avg(dl)})} = \sum_i^n (\log \frac{N + 0.5}{n(q_i) + 0.5}) \cdot \frac{f_i(k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \frac{dl}{avg(dl)})}$$

影响 BM25 公式的因数有

idf, idf 越高分数越高

tf tf 越高分数越高

dl/avgdl 如果该文档长度在文档水平中越高则分数越低。

所以，最终得到打分最高的句子就是我们需要的答案句。

**BM25 算法的实现：**

首先，计算 tf,idf 等参数，定义文本长度，文本平均长度

```
def inition(docs):
    D = len(docs)
    avgdl = sum([len(doc) + 0.0 for doc in docs]) / D
    for doc in docs:
        tmp = {}
        for word in doc:
            tmp[word] = tmp.get(word, 0) + 1 # 存储每个文档中每个词的出现次数
        f.append(tmp)
        for k in tmp.keys():
            tf[k] = tf.get(k, 0) + 1
    for k, v in tf.items():
        idf[k] = math.log(D - v + 0.5) - math.log(v + 0.5)
    return D, avgdl
```

计算相似度等方法，将上面的公式直接代入，得

```
def sim(query, index):
    score = 0.0
    for word in doc:
        if word not in f[index]:
            continue
        d = len(document[index])
        score += (idf[word] * f[index][word] * (k1 + 1) / (f[index][word] + k1 * (1 - b + b * d / avgdl)))
    return score
```

对所有的文本都进行相同的操作，得：

```
def simall(query):
    scores = []
    for index in range(D):
        score = sim(query, index)
        scores.append(score)
    return scores
```

然后，对 multi\_sentences 文件中的文本进行去停用词操作，计算所需要的文本长度和平均文本长度，tfidf 也在这里计算出来，得到的分词结果存储在 query 中，然后代入到 simall

( ) 方法中，得到 bm25 算法的结果，相似度计算完成。返回得分最高的几个结果，最终的结果使用 Mrr 来衡量，MRR 算法：

```

iterate = 0
while len(scores):
    iterate = iterate+1
    max_index = scores.index(max(scores))
    if answer_list[max_index]==answer_sentence[0]:
        correct=correct+(1/iterate)
        break
    else:
        scores.remove(scores[max_index])
        print('预测答案句：'+answer_list[idx])
        print('标准答案句：'+answer_sentence[0])

```

即对得到的每个答案组根据正确答案所在的位置求出相应的权重，如果正确答案在第一个，即加上 1/1，如果在第 n 个，就加上 1/n，相加并求平均值就得到最终的结果。

MRR 值为 0.5699901844201791

经过助教和同学的提示，发现这里结果并不好的原因是参数设置问题。即上面提到的 k1,k2,b 设置有误，经过不断测试和调整的迭代，得到最优的 k1,k2,b 参数，最优时的 MRR 为：

0.7131214342323421

## 4. 答案抽取

根据之前的三个步骤，

我们先对文档建立索引，所以给定问句，我们可以通过分词得到关键词，找到和问句最相关的文档。

然后在文档中找到和我们需要的答案相关度最高的句子，将答案定位精确到句子的级别。

最后，对句子进行分词，找到和答案最相关的词组，即我们需要的最终答案。

在对答案进行抽取时，有基于机器学习和基于规则的方法。

**基于机器学习方法的思想**是将训练预料中的问句和答案进行分词，然后进行词性标注，将词性信息作为特征进行训练，得到对应的机器学习模型。最后根据模型进行预测。

机器学习方法的原理：

### 基于规则的方法

先对问题进行分类，这里面需要用到机器学习算法，然后根据对应的问题类别编写不同的算法。

例如，对于实体类问题，问句中往往会问到…是什么？那在此之前的实体就是源实体，我们对此进行查询，在对应的答案句中找到目标实体，就是问题对应的最佳答案。这里需要根据 ltp 的实体标注结果对实体进行提取，提取算法如下：

```
#将实体提取出来
def extract_entity(words,postags):
    entity = ''
    for word,postag in zip(words,postags):
        #是单个实体, 直接返回
        if 'S' in postag:
            return word
        if 'B' in postag:
            entity = entity + word
        if 'I' in postag:
            entity = entity + word
            continue
        if 'E' in postag:
            entity = entity + word
    return entity
```

其中, B 为实体的起始位置, I 为实体的中间部分, E 为实体的结束位置。然后, 利用实体标注结果对句子进行依存语法分析, 这在之后答案成分的提取有作用。

LTP 使用的词性标注集:

Tag	Description	Example	Tag	Description	Example
a	adjective	美丽	ni	organization name	保险公司
b	other noun-modifier	大型, 西式	nl	location noun	城郊
c	conjunction	和, 虽然	ns	geographical name	北京
d	adverb	很	nt	temporal noun	近日, 明代
e	exclamation	哎	nz	other proper noun	诺贝尔奖
g	morpheme	茨, 甥	o	onomatopoeia	哗啦
h	prefix	啊阿, 伪	p	preposition	在, 把
i	idiom	百花齐放	q	quantity	个
j	abbreviation	公检法	r	pronoun	我, 我们
k	suffix	界, 率	u	auxiliary	的, 地

上图是 ltp 所用的词性标注集, 比较方便的是, 这里面已经将外文、日期、地名、朝代等信息全部提出

**对于日期类问题**, 则寻找答案句中中和日期有关的信息, 注意这里面有干扰项。比如所题目中问道“最早是什么时候?”, 就不能只是简单的提取日期信息, 而应该对齐进行相应的处理。

**对于人名类问题**, 若题目中问道“...是谁”, 则答案应该为一个人名, 所以从答案句中提取出该人名即可。

通过测试, 基于规则的方法可以得到更加精确的答案。但是由于基于规则的方法只能解决特定的问题, 所以很容易造成过拟合问题。

**对于外文类问题**, 处理策略是检测题目中是否提到英文或者其他语言, 然后提取答案句中的外文作为返回结果。这里只针对常见的语言, 如韩文英文德文法文等等。

**对于实体类问题**, 即“...是什么”, 先对问题进行实体的抽取, 然后在答案句中根据依存语法分析谓词和次级实体, 即最终的答案。

所以, 为了让答案具有更好的表现, 这里选择先用基于机器学习的方法进行扫描。然后对答案进行检查, 如果发现与正确答案的相似度太低, 就进行相应的基于规则的修改, 这样

得到的结果表现会更好。

结果：在 train.json 上进行测试，基于规则的方法

bleu1 value: 0.36157720264817518

exact match rank: 0.15678082191780822

## 四、实验心得

通过本次实验，对机器学习算法，基于规则的算法，检索系统的实现过程有了更深刻的理解。同时，自己做的系统还有很多的不足，离达到实际应用的水平还有很大的差距。在之后的学习过程中，也应该加强这方面的了解，提高自己的专业知识和技能。