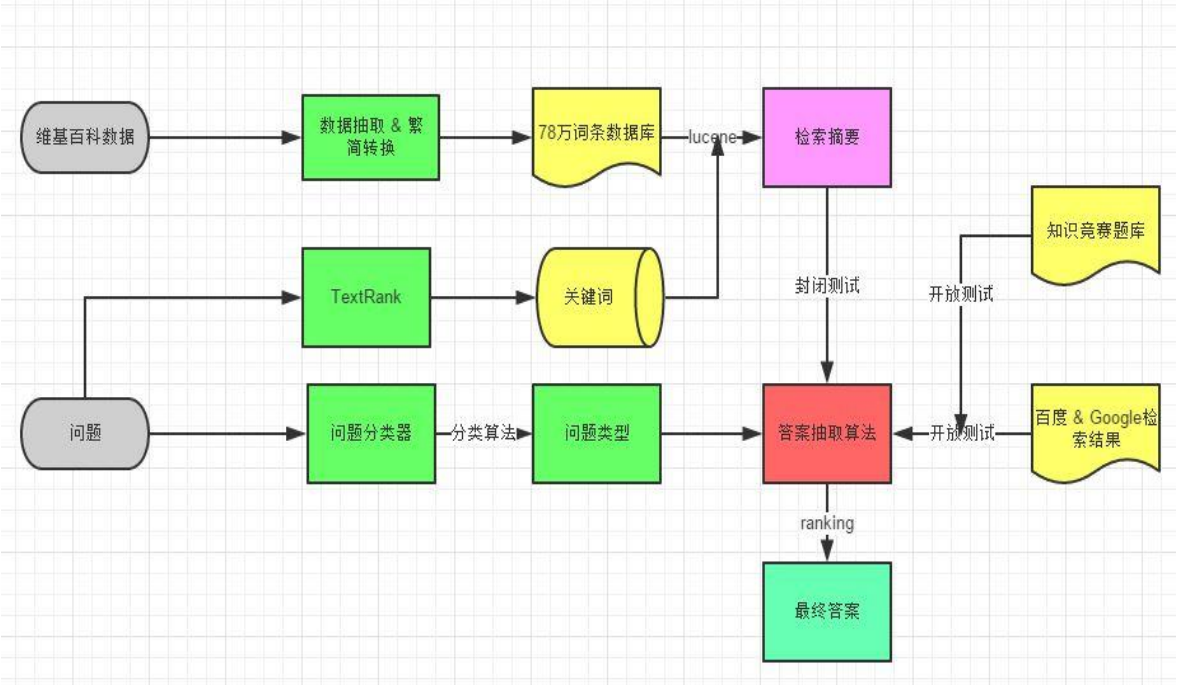
中文问答系统大作业报告

王亮 1401214305　　赵磊 1401111368
李琦 1401214286　　贾延延 1401111363

一、系统整体架构

系统主要包括四个部分：答案分析模块、文档检索模块、答案抽取模块、开放问答模块，其中答案分析模块主要负责问题分类和关键词抽取；文档检索模块负责建立维基百科数据的索引，并快速给出某个查询串的检索结果；答案抽取模块负责抽取一系列的候选答案，并进行排序，得到最佳候选答案；开放问答部分则负责收集尽可能多的相关资源；

各个部分形成一个流水线，系统的架构图如下所示：



开发过程中我们利用 bitbucket[1]做私有项目托管，所有代码均可通过访问小组项目主页 https://bitbucket.org/sckr14/pkuqa/src 获得。

二、问题分析模块

问题分析模块，负责对原始问题进行关键词提取得到后续检索系统的查询串，并训练一个问题分类器，给出该问题的答案类型。

[1] https://bitbucket.org

2.1 编译/运行环境

程序采用python 2.7编写；需要依赖sklearn[2]，nltk[3]，jieba[4]三个自然语言处理相关的库。

2.2 系统架构& 关键技术

本部分的工作包括三部分，首先进行关键词提取，去掉停用词，然后训练分类器，我们采用尝试了多种分类器进行训练，根据结果好坏选择支持向量机进行训练，预测答案类型，在随机选择的验证数据集上达到了87%的准确率。最后我们将关键词和答案类型保存在XML中为后续步骤做准备。

2.2.1. 分词和关键词提取

首先我们调用 jieba 分词进行关键词提取，采用基于图的 TextRank 算法，topK 为返回几个 TF/IDF 权重最大的关键词，默认值为 20，可以进行调节，由于问题文本相对较短，我们选择了抽取 5 个关键词。

2.2.2. 训练分类器

训练问题分类器是一个典型的有监督机器学习问题，人为将问题的答案类型划分成了四个类别：Person，Location，Number，Other。这一步的最大困难在于训练数据的缺失，我们采用了两个办法来解决这个问题：

1. 从FudanNLP[5]的数据集中抽取事实性问题，得到大约4000条训练数据。
2. 从网络上爬取一系列的问题答案对，对答案进行命名实体识别标注，得到大约6000条训练数据。

以得到的数据为基础，尝试对比了一系列的分类器模型，在预留出来的20%数据上进行了效果验证，各个分类器的效果数据如下表格所示：

分类器	**NB**	**Random Forest**	**L1 SVM**	**L2 SVM**	**KNN**
F值	0.82	0.79	0.85	0.87	0.81

经过性能对比，最终选择了L2 SVM作为分类器模型。

2.2.3. 保存关键词提取结果和答案类型

将测试集中的问题进行关键词提取，然后用训练好的分类器处理生成答案类型。采用python中的ElementTree构建XML树保存测试集中问题的关键词和预测的答案类型。

[2] https://github.com/scikit-learn/scikit-learn
[3] https://github.com/nltk/nltk
[4] https://github.com/fxsjy/jieba
[5] code.google.com/p/fudannlp/

三、文档检索模块

文档检索的任务，就是根据问题分析给出的检索词列表进行检索，找出 top N 的相关文档片段，我们就认为答案就在这些片段里面。具体包括提取 wiki 正文内容，繁简转换，Lucene[6] 建立索引，查询关键词及摘要提取。

3.1 数据抽取

由于原始的 wiki 文本包含很多无用信息，包括 web 标记，历史编辑信息、编辑人。这些信息对于提取关键内容摘要会造成干扰，因此在数据预处理阶段需要对原始的 wiki 文本进行过滤，去除无用信息。

在这一过程，我们使用 Wikipedia Extractor[7] 抽取正文文本。Wikipedia Extractor 是意大利人用 Python 写的一个维基百科抽取器，使用非常方便。下载之后直接使用相应命令即可完成抽取，运行了大约半小时的时间。

3.2 繁简转换

维基百科的中文数据是繁简混杂的，里面包含大陆简体、台湾繁体、港澳繁体等多种不同的数据。有时候在一篇文章的不同段落间也会使用不同的繁简字。

为了方便起见，我们直接使用了开源项目 opencc[8]。参照安装说明的方法，安装完成之后，使用下面的命令进行繁简转换，整个过程大约需要 1 分钟。

```
opencc -i wiki_00 -o wiki_chs -c zht2zhs.ini
```

命令中的 wiki_00 这个文件是此前使用 Wikipedia Extractor 得到的。

到此为止，已经完成了大部分繁简转换工作。实际上，维基百科使用的繁简转换方法是以词表为准，外加人工修正。

在完成文本的抽取和繁简转换后，为了避免一个文本文件包含不同词条会对摘要提取和答案抽取过程造成干扰。我们将每一个词条写入单独的文本，最后共得到 786985 个文本文件用于后续的建立索引过程。

3.3 建立索引

这一过程我们使用 Lucene 全文检索引擎。Lucene 是 apache 软件基金会 4 jakarta 项目组的一个子项目，是一个开放源代码的全文检索引擎工具包， Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

Lucene 采用倒排索引方式，倒排索引一般表示为一个关键词，然后是它的频度（出现的次数），位置（出现在哪一篇文档中，及有关的日期，作者等信息）。建立索引过程首先要进行分词，Lucene 提供了中文分词器，但该自带的分词器只能将中文文本分割成一个个

[6] lucene.apache.org/
[7] https://github.com/bwbaugh/wikipedia-extractor
[8] https://github.com/BYVoid/OpenCC

分立的字，不能实现真正意义上的分词。为此我们采用 IKAnalyzer[9]分词器，IKAnalyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包，可以得到较好的中文分词效果。

分词示例如下：

1971 年作为总统特使访华，为中美关系开启了历史性的大门的美国国务卿是谁

分词结果：

(body:中美关系 body:中美 body:关系) body:特使 (body:国务卿 body:国务) (body:历史性 body:历史)

实现分词后，就可利用 Lucene 自带的 API 建立索引。

3.4 查询和摘要提取

建立了文档的索引之后，下一步我们根据问题关键词在索引中查询相关文档并提取内容摘要。

实现过程为：

在索引中进行关键词查询，得到与关键词相关度最高的 10 篇文档。对于每篇文档，我们提取出关键词前后共 100 个字符作为摘要，以 XML 格式输出，用于后续答案抽取。

示例摘要输出如下：

```
<question id="15">
  <q>芭蕾起源于哪个国家？</q>
  <category>Location</category>
   <summary>芭蕾起源于哪个国家,兴盛于哪个国家,并已形成了几个学派?...芭蕾艺术孕育在意大利,诞生在十七世纪后期路易十四的法国宫廷,</summary>
   <summary>芭蕾舞起源於文艺复兴时代的意大利贵族,它主要是在皇帝和皇后面前表演的宫廷舞蹈,其后流传到法国，这些舞蹈员后来离开皇宫在剧院演出</summary>
   <summary>芭蕾起源于哪个国家 芭蕾最初是欧洲的一种群众自娱或广场表演的舞蹈,在发展进程中形成了严格的规范和解构形式、其主要特征是女演员要穿上特制的足尖鞋立起脚尖舞.</summary>
</question>
```

四、答案抽取模块

4.1 编译/运行环境

Java JDK6 及更高版本,另需斯坦福中文分词程序[10]以及词性标注程序[11]。

4.2 系统架构& 关键技术

在对可能包含答案的文档片段进行分词和词性标注后，主要分为候选答案提取、文档片段评分、候选答案评分三个部分。

4.2.1.候选答案提取

AnswerSelector.java 处理候选答案提取。我们将问题分为人物(Person)、地点(Location)、数字(Number)、其他(Other)类型。不同类型的问题限制候选答案的可能词性，

[9] code.google.com/p/ik-analyzer/

[10] http://nlp.stanford.edu/software/segmenter.shtml

[11] http://nlp.stanford.edu/software/tagger.shtml

该模块负责将根据问题的类型选择出词性符合该类问题的答案词性的词。在性能调优过程中，我们还发现，通过将单字构成的候选答案过滤掉将可较大程度提高答案正确率，同时各类问题也有出现频率较高的错误答案，我们也对其进行过滤。

4.2.2.文档片段评分

edu.pku.QRanking.summaryscore 包中包含对文档片段评分的所有类。对文档片段评分也就是判断该文档片段存在正确答案的可能性。对文档打分的类都是对 SummaryScorer 接口的实现。CombinationSummaryScorer.java 中综合了所有的具体给文档片段打分的类，给每个打分类一个权值，加权求和得到每个文档片段的得分。具体的打分类包括以下几个：TermEmergeScorer.java 根据问题中实词在文档片段的出现频率对文档片段打分，BigramEmergeScorer.java 根据问题中实词的二元词在文档片段中的出现频率打分,SkipBigramEmergeScorer.java 根据问题中实词中间间隔一个词的两个词以同样的形式在文档片段中的出现频率打分。

4.2.3.候选答案评分

edu.pku.QRanking.answerscore 包中包含候选答案评分的所有类。对候选答案打分的类是对 AnswerScorer 接口的实现。

CombinationAnswerScorer.java 中按照一定的权重综合所有具体给候选答案打分的类给答案评分。由于候选答案是从各个文档片段中提取的，我们先将其看做不同的答案分别进行打分，将答案的评分与其所在文档片段的得分相加，然后再将不同文档片段中提取出的相同候选答案合并，也就是将分数累加，最后按照合并后的候选答案的总分对候选答案进行排序，选出最高分候选答案作为最终答案。在细节上通过一些可信度很高的规则提高答案正确率：将问题中出现的词过滤掉，将可能经常出现的一些词过滤掉，比如开放领域检索结果中的“百度”“知道”“答案”等；对人物类型的问题，考虑到有些位置出现的国外人物的姓名是完整的，有些位置只出现 Family Name，可以根据如果一个人物姓名类型答案的字符串包含另一个答案的姓名，也将其合并。

具体的打分类如下：TfScorer.java 根据候选答案的出现频率打分。TermDistanceScorer.java 根据候选答案在候选答案中的位置与问题中的实词在文档片段中的位置的距离总和来打分，距离越小打分越高。TermMiniDistanceScorer.java 根据候选答案在候选答案中的位置与问题中的实词在文档片段中的位置的最小距离来打分，同样是距离越小打分越高。TermAlignmentScorer.java 基于文本对齐的想法，将候选答案依次替换问题中的每个词，再在每两个词之间增加一定数量的任意匹配构成正则表达式在文档片段中根据匹配情况打分。LooseTermAlignmentScorer.java 是更不精确的文本对齐，它将问题中的单字词不加考虑，再按照前面一种打分类的方式根据匹配进行打分。

在性能调优中，我们发现基于文本对齐的评分赋予较高权重会得到更高的正确率，其次是基于词项距离的评分，基于词频的评分给予较低权重，其影响过大容易造成某些常见类型错误，比如在人物类型问题中，可能最高分答案为其国籍，这是因为斯坦福词性标注程序的词性标签来自 LDC Chinese Treebank POS tag set[12]，此集合在词性上不将两者区分。

[12] https://catalog.**ldc**.upenn.edu/**LDC**2013T21

4.2.4 计算公式

文档片段评分计算公式：

文档片段总得分 = Σ文档片段的各项评分结果

文档片段词项出现得分 = 词项出现得分所占权重 *（问题中的所有实词出现的次数 / 文档片段长度）

文档片段二元实词出现得分 = 文档片段二元实词出现得分所占权重 *（问题中的所有二元实词出现的次数 * 2 / 文档片段长度）

文档片段间隔二元实词出现得分 = 文档片段间隔二元实词出现得分所占权重 *（问题中的所有间隔二元实词出现的次数 * 2 / 文档片段长度）

候选答案评分计算公式：

候选答案的最终得分 = Σ候选答案在所有文档片段中得分

候选答案在文档片段中的得分 = 候选答案的各项评分求和 + 文档片段的得分

候选答案文本对齐得分 = 候选答案文本对齐得分所占权重 * 文本对齐匹配次数 * (问题长度 / 平均对齐文本长度)

候选答案不精确文本对齐得分 = 候选答案文本对齐得分所占权重 * 不精确的文本对齐匹配次数 * (问题长度 / 平均对齐文本长度)

候选答案距离得分 = 候选答案距离得分所占权重 * 1 / [问题中所有实词的位置与候选答案所有位置的距离总和 /（候选答案在问题中出现的次数 * 问题的长度）]

候选答案最小距离得分 = 候选答案最小距离得分所占权重 * 1 / 问题中实词位置与候选答案的最小距离

候选答案出现频率得分 = 候选答案出现频率得分所占权重 * 候选答案在文档片段中的出现频率

五、开放问答模块

对于开放问答而言，最重要的是寻找最有效的知识库资源，将基于中文维基百科的检索扩展成基于所有可获得知识的检索，我们采用了两个基本思路：

1. 利用主流搜索引擎返回的搜索结果，从中抽取摘要，作为答案抽取模块的输入；

主流的中文搜索引擎包括百度和谷歌，其中百度并没有公开的 API 接口，于是我们模拟正常客户端发送 http 请求，并利用 jsoup 来解析[13]，由于每发送连续的大约 1000 个 http 请求会因为流量作弊被封禁 ip，为了抓取 15000 个问题的搜索结果，我们以 VPN 的服务器为 http 代理反复进行了十几次抓取；

对于 google，有公开的 ajax API[14]，得到 json 格式的数据[15]，可以利用 json-java[16]函数库来进行解析，提取出其中的摘要部分。后经测试，发现连续抓取几十条结果之后 google 会自动关闭相应的 API，不再返回任何结果，因此对于上万条数据的批量抓取并不合适，但对于分散少量数据爬取还是有意义的。

13 https://github.com/jhy/jsoup/
14 https://developers.google.com/speed/libraries/
15 www.json.org/
16 https://github.com/douglascrockford/JSON-java

2. 爬取主流知识竞赛的题库，抽取出相应的问题答案对，并进行字符串的近似匹配；

搜索引擎返回结果的摘要一大优势就是覆盖的文档范围很广，但是仍然需要在摘要里面运行答案抽取算法，并不能保证一定可以得到一个问题的正确答案。相比之下，现有的知识竞赛题库，覆盖的范围虽然小，但是问题答案对都是精准的。

在开放评测中，我们收集了接近 10000 条半结构化的知识竞赛题目数据，在进行了基于规则的预处理之后，提取出接近 8000 条问题答案对数据，然后利用编辑距离[17]进行了匹配，共匹配到 6000 条左右。

六、小组分工

贾延延 负责问题分析模块；
赵磊 负责文档检索模块；
李琦 负责答案抽取模块；
王亮 负责开放问答部分；

七、总结

小组四个人从 10 月份开始看论文、找思路，每周三晚上都进行例行的讨论，同时在 QQ 上不定期讨论。从一开始毫无思路，到搜索相关资料、学习各种开源工具的使用、逐步建立并完善各个模块，几乎每天都会往 bitbucket 上提交代码，修正 bug、改进实验效果，大家虽然都不是大牛，但都非常靠谱，相互之间都学习到了很多东西。

【参考文献】

[1] Patanaik A. Open Domain Factoid Question Answering System[D]. INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR, 2009.
[2] Murdock J W, Fan J, Lally A, et al. Textual evidence gathering and analysis[J]. IBM Journal of Research and Development, 2012, 56(3.4): 8: 1-8: 14.
[3] ysc. QuestionAnsweringSystem[Z],2014. https://github.com/ysc/QuestionAnsweringSystem
[4] Mihalcea, Rada, and Paul Tarau. "TextRank: Bringing order into texts." Association for Computational Linguistics, 2004.
[5] Iyyer, Mohit, et al. "A neural network for factoid question answering over paragraphs." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.

[17] http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance