1 作者信息：

徐威迪 1401111377

白艺冲 1401214332

杨蕴伦 1401214401

2 分工情况：

徐威迪：问题语料集的预处理、问句分析（类型判定、关键词抽取）

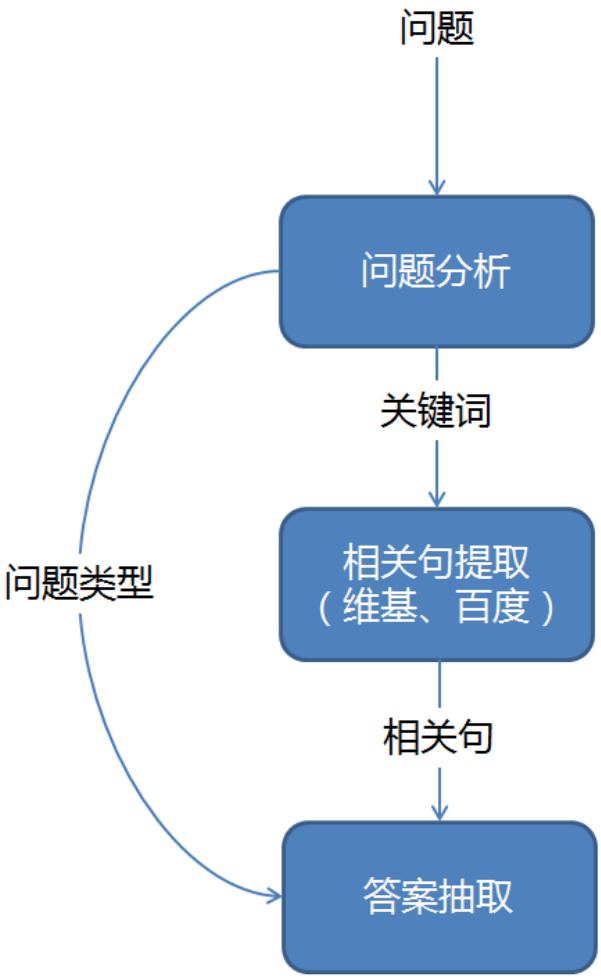
白艺冲：维基语料集的预处理、相关句提取（从维基中提取与问题相关的若干句子）

杨蕴伦：开放测试语料集的采集与预处理，答案抽取（根据问题与相关句获取答案）

3 编译/运行环境：

Python 2.7

4 系统架构：



5 使用的方法/资源

5.1 问题分析

5.1.1 方法概述：

在该系统的问句分析部分，我们主要分析了问句的类型，限定词（Lexical Answer Type），和关键词。问句的类型主要包括如下几个部分：

1. WHICH: 通常指哪个东西，哪个国家等等，需要返回一个指定的名词。
2. WHAT: 通常含有疑问词“什么”，与“WHICH”类似，通常需要返回一个
3. WHO: 询问内容为人名，包括姓，名，需要返回一个与人名相关的词

4. WHEN: 指哪些查询内容为时间的问句。抽取内容还包括时间的范围，比如“年”，“月”，“日”，“星期”，“小时”等，需要返回一个时间。
5. WHERE: 查询地点，返回地点。
6. HOW: 形容“怎么”，“如何”之意，应返回一个原因或者形容过程的答案。
7. HOWMANY: 形容数量多少，包括“多少个”，“几倍”，“多久”等，需要返回一个数量。
8. LIST: 需要返回一个列表。
9. HOWMUCH: 一般指“多＋<形容词>”，不一定是指数量，需要返回一个程度副词或者数量。
10. RANK: 询问内容包括“第几”等特定排序次，需要返回一个名次。
11. YES/NO: 问句是是判别句，需要返回“是”或者“否”。

每种类型的问句，该系统都制定了相应的模板，使用自定义数据结构抽取相关限定词，并返回所有匹配。模版是根据样例和测试集拟定的，包含 167 个匹配模版，全部由人工工作完成。在模版的基础上，对问句进行抽取。该系统使用了 trie 树的数据结构，并且做了特定的修改，使之能够高效地完成问句分析工作。

关键词的抽取则调用 jieba 库，使用其 API。

5.1.2 实现及其细节:

模版是问句分析的基础，可以视为人对问句的先验知识的体现。模版的结构定义如下：

```
==== <问句类型>
<匹配>:<{0,1}>[:<默认词>] […]
<匹配>:<{0,1}>[:<默认词>] […]
```

其中”<A>”表示 A 的集合中的某一个值，”<问句类型>”为上述 11 种，”<匹配>”表示需要匹配的内容，可以是词（包括结束符“$”）也可以是 POS[注 1]或者指定符号[注 2]，”<{0,1}>”表示该词是否需要提取，”<默认词>”是只要匹配到就默认提取的词，主要用于解决“哪年”之类的匹配（需要返回限定词“年”）。一个模版可能有多个匹配词构成。部分模版可以参看下图

注 1：词性分析工具使用 Python jieba 中文分词库
注 2：我们另外定义了三个匹配符号，”＊”，　“＊＊”，“－1”分别指代任何词，任何长度的子串和句子的最后一个词。

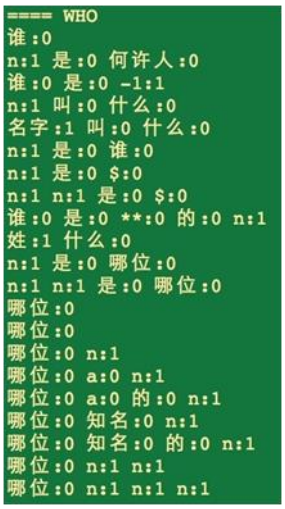


图 1：“WHO”类型的匹配模版示例

限定词的抽取工作是基于 trie 树数据结构完成的。树节点的数据结构定义如下：

```
class Trie:
    %components
    self.val: string            %表示匹配词
    self.extract: boolean       %表示是否抽取
    self.child: dict            %子节点，根据匹配词生成
    self.ltype: string          %问句类型
    self.end: boolean           %是否该模版匹配完成，虽然不一定到达根节点
```

首先，我们要读取模版内容，然后根据模版内容生成 trie 树结构。对每个问句的后缀数组都进行一次扫描，如果匹配就将匹配的问句类型（由 ltype 给出）和匹配词组成的一个匹配添加到一个列表内。具体代码参照 scripts/trie.py。提取结果可以参考图 2。

图 2：部分样例分析结果

关键词抽取则调用 jieba 库的内置 API，使用如下语句即可得到关键词。

```
keywords = jieba.analyse.extract_tags(query, num_query)
```

5.1.3 实验结果和分析：

在样例集合上，问句类型的覆盖率为 98/100，表明模版匹配的覆盖率还是较高的。其中有两句没有得到问句类型，一句是由于分词错误导致模版匹配失败，另一句是比较类型的问句，该系统没有覆盖。

限定词的覆盖率大概在 95％，去除部分问句如“WHERE”和“WHO”某些情况下本身没有限定词。没有覆盖的主要原因是由于词性标注错误导致限定词覆盖率降低。

5.1.4 总结：

问句分析是整个问答系统的基础，我们需要尽可能地提取有效的信息，然后提交给检索系统和答案提取系统。这里我们选择基于人工规则的匹配方式，主要有如下几个原因。首先我们没有足够多的数据支撑基于经验统计模型的训练，另外人工规则在目前限定范围内的问句分析是非常有效的，尤其当问句的形式固定的时候。

结果显示该方法的覆盖率是比较高的，在测试集开放之后，我们针对测试集进行了模版扩展，所以在测试集上的效果也是让人满意的。然而该方法的缺点也是显而易见的。第一点是其受限于固定的模式，不能解决问句的多样性。第二点是模版的覆盖范围是有限的，在模版的覆盖率和准确率之间有一个平衡关系，在保证高准确率的情况下，模版的覆盖范围是有限的。随着模版越来越多，相互重叠的情况就会发生，从而带来噪声。这些都是将来要解决的问题，运用大量数据的统计模型结合少量数据的规则方法，从而达到更好的效果。

5.2 相关句提取（搜索）

5.2.1 方法概述

由于数据量过大，我们使用一种 coarse-to-fine 的过程，先找出相关的篇章，再在这些篇章里面选取句子。对于文本，我们一方面采取字符串比较的方式，另一方面又使用词向量。

在寻找相关文章的时候，我们采取字符串比较。如果某篇章中出现了问题中的关键词，则认为此篇章相关。在寻找相关句子的时候，我们使用句子平均词向量，与问题平均词向量做差，来判断相关性。

5.2.2 算法细节

使用到的算法

1. 对篇章分词
   a) 使用 jieba pseg 进行分词
   b) 去掉标签为 x、eng 的词
2. 计算词向量
   a) 使用 word2vec 计算词向量
   b) 首先根据词语列表训练词向量模型，由于分词中含有噪声，所以选择 10 作为窗口数。维度选择 200.
3. 按照每个词是否出现，找出相关篇章
   a) 使用 jieba 提取问题关键词
   b) 采取倒排索引方式确定关键词在 wiki 数据篇章中的位置。
4. 分句子求出每个词的词向量
5. 求出句子平均词向量
   a) 我们使用所有词向量的平均、最大、最小来组成复合特征
6. 为每个句子查找距离最近的 10 个句子

使用到的外部资源

1. Jieba 分词
2. Word2vec
3. Xml.sax

5.2.3 实验

实验过程中，最大的问题是数据量太大。我们采用 64g 内存的工作站进行实验。
词向量模型训练经过了约 2 个小时，每次序列处理 xml 需要 7 小时左右。

5.2.4 思考

实验中，我们发现在大数据量面前，很难使用全局的指标来构造算法。例如，我们很难找到“与问题最相近的 10 个页面”，但对于便于序列化的指标，如找出“含有问题关键词的所有页面”则相对较为容易。事实上可以尝试使用 KDTree 等方式来查询全局类型的指标。

5.3 答案抽取：

5.3.1 方法概述&算法细节

在答案抽取这个步骤，可以利用到的两个资源是问题类型（关键词）和相关句。

问题类型包括 WHICH（哪、何），WHAT（什么）、WHO（谁）、WHEN（何时）、HOW（怎样）、HOWMANY（多少）、LIST（哪些）、HOWMUCH（多少）、RANK（第几）、YES/NO（是否）。我们采取的策略较为简单，即根据问题类型来直接确定答案可能的词性，从而作为对候选答案的第一道筛选。其中对应关系如下：

- WHICH、WHAT、LIST：n、nr、ns、nt、nz、t
- WHO：nr
- WHEN：m、t
- WHERE：ns
- HOWMANY、HOWMUCH、RANK：m
- YES/NO：

（词性标记参考 ICTCLAS 标注集）

相关句有两个用途：候选答案集的来源；评估候选答案的准确性。

作为候选答案集的来源，我们会先将所有相关句进行分词，并挑出与如上所述的问题类型对应词性相符的所有词，构成候选答案集。

接下来是评价每一个候选答案的准确性。我们采用的方式是为每一个候选答案计算准确度的打分。分数由四个标准确定：

1. 候选答案与问题中出现的词在相关句中的词距

   对于候选答案 w 与问题中的词 wq，如果他们出现在同一相关句 s 中，那么他们的打分为1 / abs(index(w) – index(wq)) (index(w)代表词 w 在当前相关句中的下标)

   所以在这一标准下，候选答案 w 的得分即为上述所有共现的词对在所有相关句中的得分和，即

$$Score(w) = \sum_s \sum_{wq} \frac{1}{abs(index(w) - index(wq))}$$

2. 候选答案与问题中出现的词在相关句中的最小词距

   对上面的标准类似，不同点在于在每个相关句中取词距分的最大值，而不是取得分之和。即

$$Score(w) = \sum_s \min_{wq} \frac{1}{abs(index(w) - index(wq))}$$

3. 候选答案在相关句中的词频

候选答案在所有相关句中出现的次数

$$Score(w) = \sum_s \text{occur}(w, s)$$

4. 候选答案放入问题中后与相关句的文本匹配程度

将候选答案放入问题中的各个位置，然后通过正则表达式，与所有的相关句进行匹配，根据匹配程度的高低来确定分数。

四个标准得出分数也分别有各自的权重，对各个分数加权并求和后得到的就是候选答案准确度的打分。取分数最高的候选答案作为最终答案。

5.3.2 实验结果

通过对于有限的样例进行观察，我们的方法对于 WHO，WHERE、HOWMANY、HOWMUCH、RANK 这几类问题拥有不错的效果，究其原因是这几类问题的答案拥有较为明确的词性，所以第一步筛选起到了很大的作用。在这几类问题中，错误往往是因为相关句的质量较差或者分词系统错误地标记了相关句中词的词性（比如 jieba 分词就把“夏威夷”标记成了人名 T_T）。对于分词系统以及词性标注能力的依赖度还是很高的。

WHICH、WHAT、LIST 这一类问题，常常有较大的候选答案集，我们粗糙的打分标准很难万中挑一。

对于 LIST、WHEN、YES/NO 这类问题，则需要规则的辅助来进行多个答案的选取或者答案的重新表达，这方面仍有可做的工作。同理还有“下一句是什么”这种问题。

分词系统及词性标注的工具都是 jieba。

5.3.3 总结与思考

打分方法中的权重调整是一个难点。我们并未找到较为合适的权值调整策略。通过对结果的观察，词频（标准 3）在相关句质量较高时，对评分的影响占主导。而答案匹配程度（标准 4）则在绝大部分时候打分为零，这是受限于正则表达式的匹配效率问题，所以我们不得不降低了匹配答案的灵活度，使得固定的句式很难在相关句中匹配上。

在对问题提取关键词后，我们可以知道答案的属性，但是这个信息也没能较好的利用上。可能的想法是将属性与答案放入语料库中进行反查或者利用中文的知识图谱，但代价都比较大。

5.3.4 关于开放测试

开放测试中，我们是直接将每个问题放入百度搜索引擎，然后将搜索结果中第一页的所有网页概述作为问题的相关句，补充到从维基提取的相关句中，并重复步骤，即得到开放测试的答案。开放测试的答案在相比封闭测试有较大提升。

抓取搜索结果时使用的是 selenium+ChromeDriver。