机器翻译开发小结

张杰,MG1533082,986862436@qq.com

1 句子边缘检测(Sentence Boundary Detection)

1.1 SBD 简介

SBD 解决的问题是，对段落中句子的边界进行正确的判定，句子边界判定的正确率对于段落翻译起着至关重要的作用，一个句子边界如果界定错误，即意味着至少有两个句子翻译会出现错误。

因为英文句子中经常会出现大量的缩写词，例如 U.S.,Abbr.,Mr.，还会出现一些奇怪的符号等，这都会给 SBD 带来较大的困难。

之前常用的解决方法主要有两种，第一种是基于规则的，第二种则是基于机器学习方法的，因为 SBD 问题是一个相对比较边缘的问题，研究的人相对较少，现在基本已经没有人在研究传统的 SBD 问题。

解决 SBD 采用的方法也主要都是一些较老的机器学习方法，例如 1997 年 Palmer 和 Hearst 利用在句子边界部分的 POS 值，采用的简单的神经网络和决策树建模；1997 年的 Reynar 和 Ratna parkhi 采用'!','.','?'和缩略词的一些特征利用监督最大熵方法对问题进行求解，2002 年的 Mikheev 利用 POS,L.R,缩略词等特征建模，而 2006 年的 Kiss 和 Strunk 利用'.'周围的搭配和一些缩略词的特征，采用无监督的方法建模。

目前比较常用的 SBD 系统主要有：（如表 1 所示）

CoreNLP	R	18.7	http://nlp.stanford.edu/software/corenlp.shtml
GATE	R	60.6	http://gate.ac.uk
LingPipe	R	1.7	http://alias-i.com/lingpipe/
MxTerminator	S	2.8	ftp://ftp.cis.upenn.edu/pub/adwait/jmx/
OpenNLP	S	2.2	http://opennlp.apache.org/
Punkt	U	7.1	http://nltk.org/api/nltk.tokenize.html
RASP	R	0.3	http://ilexir.co.uk/applications/rasp/
Splitta	S	31.5	http://code.google.com/p/splitta
tokenizer	R	0.3	http://www.cis.uni-muenchen.de/~wastl/misc/

表 1. SBD 常用系统

而这些系统的系能对比可以参考下表：（如表 2 所示）

	Brown	CDC	GENIA	WSJ	All
CoreNLP	87.7	72.1	98.8	91.3	89.1
$LingPipe_1$	94.9	87.6	98.3	97.3	95.2
$LingPipe_2$	93.0	86.3	99.6	88.0	93.2
MxTerminator	94.7	97.9	98.3	97.4	95.8
OpenNLP	96.6	98.6	98.8	99.1	97.4
Punkt	96.4	98.7	99.3	98.3	97.3
RASP	96.8	96.1	98.9	99.0	97.4
Splitta	95.4	96.1	99.0	99.2	96.5
tokenizer	94.9	98.6	98.6	97.9	96.2

表 2. SBD 常用系统性能对比，行表示数据集，列表示系统

因为传统的文章段落的 SBD 已经基本没有什么人做了，所以现在也很难搜到最新的论文，而最近关于 SBD 的文章主要都集中在一些用户生成的非传统的文章中，例如微博上用户常常使用一些不规则的符号作为一个句子的结尾，例如表情符，”-”,空格符等待。

1.2 SBD 实验

因为在 LDA 上的 WSJ 文件需要用钱购买，同时考虑到现有的工具已经达到了 99%的正确率，所以最终决定放弃自己编写 SBD 处理程序。

1.3 参考文献

[1] Jeffrey C. Reynar and Adwait Ratnaparkhi*, A Maximum Entropy Approach to Identifying Sentence Boundaries ,1997

[2] Daniel J. Walker, David E. Clements, Maki Darwin and Jan W. Amtrup , Sentence Boundary Detection:A Comparison of Paradigms for Improving MT Quality

[3] Yoshihiko Gotoh Steve Renals, Sentence Boundary Detection in Broadcast Speech Transcripts ,2000

[4] Dan Gillick , Sentence Boundary Detection and the Problem with the U.S. 2009

[5]Jonathon READ, Rebecca DRIDAN, Stephan OEPEN, Lars Jørgen SOLBERG,Sentence Boundary Detection A Long Solved Problem, 2012

[6] Derek F.Wong, Lidia S. Chao, and Xiaodong Zeng , iSentenizer- Multilingual Sentence Boundary Detection Model , 2014

2　POS(Part Of Speech)词性标注

2.1　POS 简介

英文中每一个单词都会有一个或者多个词性与其对应，如图 1 所示，单词 cat 的属性就是 n.，也就是说 cat 的 POS 是名词：

图 1.cat 与其 pos

再看图 2,单词 catch 的属性有 vt.，vi.和 n.，也就是说 catch 的 POS 可以是及物动词和不及物动词以及名词：

图 2.cat 与其 pos

有些单词会存在较多的 POS,但是在一个英语句子中，每一个单词对应的 POS 应该是唯一的，而 POS 标注的意思就是对英文句子中的每一个单词进行唯一的词性标注。

对于 POS 标注最基本的 baseline 就能达到 90%的正确率，而所谓的 baseline 就是对每一个词全部都采用单词最常出现的词属性进行标注，之所以这样的主要原因在于有绝大部分的词是不存在模糊性的。

从 stanford 的课件中，我们可以发现当前最新的 POS 算法已经可以达到 97.2%的正确率，而默认的人对 POS 的识别的正确率的上界也就在 98%左右，就是说最新的 POS 分类器已经几近达到人类的水准，所以 POS 的处理直接利用了 Stanford 提供的库文件。

- Trigram HMM: ~95% / ~55%
- Maxent P(t|w): 93.7% / 82.6%
- TnT (HMM++): 96.2% / 86.0%
- MEMM tagger: 96.9% / 86.9%
- Bidirectional dependencies: 97.2% / 90.0%
- Upper bound: ~98% (human agreement)

2.2　POS 实验：

2.2.1 实验设置

实验配置： java 版本 1.8 以及以上

实验代码：去 stanford nlp 网站上下载 POS 对应的 jar 文件，设为引用即可，新建一个 class 类，然后输入如下代码即可：

```
String a="I get a number of great books .";
MaxentTagger tagger =  new
```

```
MaxentTagger("D:\\Workspace\\MT\\models\\english-bidirectional-distsim.tagger");//引用已经训练好的包
     String tagged = tagger.tagString(a);//对字符串进行词性标注
    System.out.println(tagged);
```

2.2.2 实验结果

具体的可参见图 3。

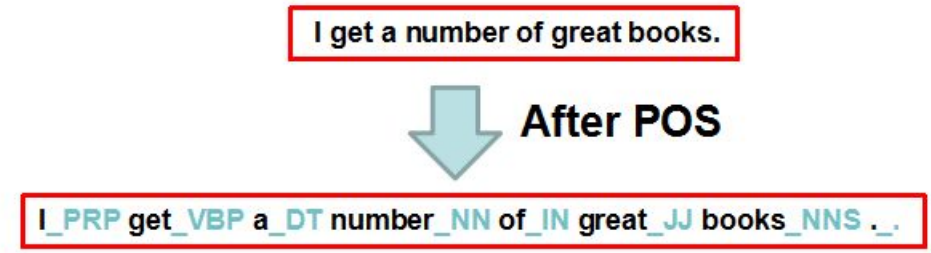


图 3.POS 实验结果

2.3 参考文献：

[1] Bracketing Guidelines forTreebank Ⅱ Style Penn Treebank Project.

[2] http://nlp.stanford.edu/software/lex-parser.shtml#Download(Stanford NLP Parser)

[3] http://spark-public.s3.amazonaws.com/nlp/slides/Maxent_PosTagging.pdf

3　对一些几乎不存在歧义的情况进行结合（因为每次合并之后仅保留最后一个作为下次进行判定的标记）

3.1　简介

此处所谓的不存在歧义的情况指的是对一些单词的结合并不会影响整体句子的结构，以及整个句子的主要意思，例如：

Beautiful(JJ) girl(NN)　　----　　Beautiful girl (JJ+NN)　　----　　girl

Extremely(RB) beautiful(JJ)　　----　　Extremely beautiful(RB+JJ)　　---　　beautiful

左边是原先的进行完 POS 标注之后出现的情况，而中间是结合之后的情况，最右边是结合之后我们选择最终保留的单词，也就是说原先的 beautiful girl 到最后我们选择仅保留 girl，extremely beautiful 我们最终仅保留 beautiful，这样我们的句子就变得更加简短，方便后续的处理。

3.2　筛选的所有组合以及算法介绍

3.2.1 实验结果

此处主要对下述认为的特征进行筛选

a. WRB+JJ　how great,how much,

b. MD+VB　情态动词+动词的情况

c. RB+CC+RB,RB+”,”+RB　副词+连词+副词的情况

d. JJ+CC+JJ,JJ+”,”+JJ　形容词+连词+形容词的情况

e. RB+JJ　副词+形容词的情况

f. JJ+NN　形容词+名词的情况

g. PDT/DT/CD/WDT+NN　冠词/基数+名词的情况

h. DT/CD+VBG/VBD(动名词/动词分词)+NN　冠词/基数+动名词/动词分词的情况

The smiling face，the broken glasses

I. NN+POS　名词+’s 的情况

J.POS+NN　‘s+名词的情况

k. PRP$+NN　所有格代名词（如 his/her)+名词

l. NN+NN　名词+名词的情况（因为采用递归的方式，要求后面不能是 DT/PDT/CD 等情况，只能是并列在一起的名词）

m. RB+VB　副词+动词的情况

n. 其他情况，直接将词性放入数组

3.2.2 算法描述

输入：改变符 change，单词数组（整个句子包含的所有单词和符号组成的数组），POS 数组（整个句子包含的所有单词对应的 POS 标记组成的数组），postmp（用来存储组合的 POS 标记）

```
public void Merge(int change,String[] words,String[] pos,String []postmp)
```

（1）扫描 pos 数组一遍，如果出现上述 3.2.1 中描述的情况中的任何一种，则将 change 标注为 1，表示进行了融合，同时将可以融合的两个单词作为一个新的单词重新存储到 words 中，两个单词的 pos 也作为一个新的组合的 pos 存储到 tmppos 中，而 pos 则存储最后一个单词（也就是第二个单词）的 pos 标记；

（2）整个数组遍历完，如果 change=1，我们就递归调用该算法；如果 change=0，则返

回。

输出：融合之后的 words 数组，融合之后的 pos 数组，以及融合之后的 postmp 数组。

3.3 实验结果：

假设我们输入如下的数据：`Merge(0,words,pos,null);`

Change=0;

words 数组： This , is , my , mother , 's , beautiful , bag , .

pos 数组：DT , VBZ , PRP$, NN , POS , JJ , NN , .

postmp 数组：null

第一步融合之后：

words 数组：This , is , my mother , 's , beautiful bag , .

pos 数组：DT , VBZ , NN , POS , NN , .

postmp 数组:DT , VBZ , PRP$ NN , POS , JJ NN , .

第二步融合之后：

words 数组：This , is , my mother 's , beautiful bag , .

pos 数组：DT , VBZ , POS , NN , .

postmp 数组:DT , VBZ , PRP$ NN POS , JJ NN , .

第三步融合之后：

words 数组：This , is , my mother 's beautiful bag , .

pos 数组：DT , VBZ , NN , .

postmp 数组:DT , VBZ , PRP$ NN POS JJ NN , .

第四步我们发现 change=0，于是我们就将上面的三个数组作为输出结果返回。

3.4 小结：

上述的规则只是个人的一些简单的总结，可能并不是很全，也可能总结的合并规则中可能存在一些特殊的情况，例如如果 POS 为 NN,DT,NN 这样的标记，我们算法在实践中会先对 DT+NN 进行结合，然后变为 NN,NN，然后再将 NN,NN 进行结合得到最后的 NN,而这明显是错的，所以我们在算法中加了判定，前一个单词如果 POS 为 NN,后一个单词必须是 NN,不能是其他的标记，否则就不进行合并。

3.5 参考文献：

无

4 查找固定词组，并对这些新词组进行重新标记

4.1 简介

这一步要做的就是查找句子中一些常见的固定词组，并对它们的 POS 进行重新标记，将一个词组作为一个新的单词，然后对这个新的单词重新赋予一个新的 POS，例如：现在我们有词组，a lot of，如果按照原来的方法，我们可以得到的 POS 标注为：DT NN IN，按照 3 中的方法进行组合之后我们可以得到:DT NN,IN 而这很明显不符合我们平时直观上的对它的一个理解。而如果我们将它作为一个整体重新赋予新的含义，即我们将 a lot of 作为一个单词，并将其 POS 改变为 JJ,即形容词，这样就比较符合我们的直观理解。

这一步的目的有二，其一就是希望将原先的句子变为我们直观上更易理解的组合，另一个目的是为了将句子进一步精简化。

目前我们发现固定词组搭配存在一些主要问题:

问题一.有一些固定词组是存在一些错误的，例如：a number of,如果我们直接将其重新标记为 JJ（形容词），在一些场合会引发一些错误，例如：This is a number of my sister.如果我们直接重新标记，很明显这个句子就错了；

问题二.固定词组形式是不变的，但是会存在一些插曲，例如 a number of 和 a great number of，我们理解时都是当做一个词组来进行理解的，所以如果遇到了第二种情况我们又该怎么处理？

问题三.有一些固定搭配很复杂很模糊，该不该算作一个固定词组加进来?例如 both … And …这样的词组，虽然我们会把 and 左右两个词都加进来结合 both and 作为一个词组，但是这样的词组不知道如何限定界限，即 and 右边应该扩展到哪里终止。

问题四.一些固定词组不知道如何赋予新的 POS 值，例如，as well as，它可以作为连词，如 you as well as he,也可以作为介词，例如，he plays as well as you.所以这种情况又该如何处理？

解决方法：

对于问题一.我们直接加入新的 POS 进行模糊性删除，即 a number of NNS，此时我们就可以判定它为 NNS,即这个时候我们就把这整个当做一个词组,最终我们将它 POS 标注为 NNS，PS:这边的 NNS 是名词复数的意思。

对于问题二.我们直接引入新的符号，”(...)”，a (...) number of NNS,而这个符号表示的意义就是说 a 和 number 之间可以是形容词或者副词或者形容词与副词的组合，也可以为空。

对于问题三，我们引入新的符号,”...”，表示可以是形容词，副词或者名词，抑或可以是它们之间的组合，也可以为空，然后我们将 both and 变为 both ... And,将右边的...省略，最后我们将其标记为... + CC 意思就是...的属性，加上连词，例如 both beautiful and,标记为 JJ+CC（形容词+连词），这边这样做省去了查找词组右边界的问题，但是感觉还是可以改进。

对于问题四，这是一个类似于歧义的问题，这种情况姑暂时就不放入固定词组，这个需要根据上下文来断定，如果 as well as 第一个 as 之前是动词，则可以把其当做是介词来看，而如果是名词则可以当做是连词来看，但是因为处理较为麻烦，这边选择没有进行处理。

4.2 词组 POS 重标记算法

4.2.1 实验中的一些 trick

（1）因为在处理 a number of NNS 等问题时，会出现例如 a number of extremely beautiful flowers 的情况，所以我们在进行词组重新标记时，先采用算法二对问题进行基本的组合，此时我们就可以将 extremely beautiful flowers 标记为 NNS，这样我们便可

以快速的进行匹配，达到较好的效果；

（2）考虑到后续词汇库可能较大，如果对词汇库从上往下一一匹配所消耗的时间会较长，所以我们先对所有的固定词组进行存储，然后将每个词汇的第一个单词以及其对应的固定词汇表取出建立 map，map 的 key 是词组的第一个单词，而 map 的 value 是以该单词开头的所有固定词汇，例如单词 a，对应的 value 就是{a lot of , a lot, a (...) number of}等，考虑到每个单词对应的词汇不是很多，所以如果找到以该单词开头的所有词汇，再遍历所有的词汇所需的时间就不会很长。

（3）在进行固定词组 POS 重标记的实验时，因为提前调用了算法 3 进行了第一步的操作，所以这边有的部分需要拆分重组，例如，both beautiful and nice，之前我们会把 both 放在一边，而 beautiful and nice 放在另外一边，作为两个部分进行处理，但是因为我们需要构造固定词组并重新修改成份，所以这边我们会把 and 拆开，重组为 both beautiful and 放在一边并标注为 JJ CC，而 nice 放在另外一边，标注为 JJ。

4.2.2 实验注意：

对固定词组进行重新标注时，因为需要对单词进行匹配，需要遍历整个句子中的每个单词，相当于拆开重组，最终我们可以得到的结果是：每个单词的 POS，这边词组算作一个新的“单词”。

4.3 实验结果：

（1）原句子：

There are a great number of flowers and a lot of beautiful things .

（2）经过 stanford 进行 pos 标注后：

There_EX are_VBP a_DT great_JJ number_NN of_IN flowers_NNS and_CC a_DT lot_NN of_IN beautiful_JJ things_NNS ._.

（3）第一轮词组组合之后：

There EX are VBP a great number DT JJ NN of IN flowers NNS and CC a lot DT NN of IN beautiful things JJ NNS . .

（4）固定词组 POS 标记修改之后：

There_EX are_VBP a great number of flowers _NNS and_CC a lot of _JJ beautiful_JJ things_NNS ._.

（5）第二轮词组组合之后：

There EX are VBP a great number of flowers NNS and CC a lot of beautiful things JJ JJ NNS . .

4.4 实验小结：

本小节最大的特色在于：引入了词组重新标注法，使得对于句子进行的标注更加趋于人性化，更加符合我们对于句子的理解方式，虽然这些在之前的一些语言计算科学的杂志中提到过，但是在最近最新的 parser 中却从未出现过这样的方式，所以引入这样的词组重新标记还是至关重要的。

但是这也带来了很大的挑战，此处我们处理的固定词组主要包含：全固定词组和半固定词组，全固定词组有 in short，by and large 这种，是不可以拆分也不可以在中间添加其他成分，而半固定词组则指的是 an (extremely great) number of 等，这种中间可以添加形容词副词或者它们之间的组合的，还有一种稍加复杂的半固定词组，就是中间可以是名词的，例如 both a man and，这种则较难判别，因为中间的名词是可以用多种方式进行修饰的，比如这边的，我们可以改写为 both a man with a gun in a blue shirt and，这个时候我们就

无法识别出 both...and 这个词组，因为这边我们设置较为简单，仅仅进行一次简单组合后的词汇进行识别，如果 both 后面是 NNS 然后后面跟着的是 and，则我们可以正确识别，但是这里中间穿插了 IN+NN+IN+NN，我们的做法是拒绝识别然后往后进行识别，所以这边我们暂且不进行处理。

4.5 参考文献：

[1] Parsing Models for Identifying Multiword Expressions，Spence Green, Marie-Catherine de Marneffe Christopher D. Manning,2013

5 介词词组搭配

5.1 简介

目前的 parser 主要分为两种，一个是 constituent parser，另外一个是 dependency parser，而个人认为 dependency parser 更加通俗易懂，更符合人们对于一个句子意思的理解过程，也就是说包含了词义以及上下文之间的关系。

dependency parser 中词组搭配问题是较难的，动词和名词的搭配还算比较简单，动词后面如果直接跟着名词我们就直接默认后面一个名词是修饰这个动词的，而如果动词后面跟着介词介词后面跟着名词则这个介词词组也肯定是修饰名词的，这样的情况不存在歧义。但是名词和介词词组，动词和介词词组之间的搭配则较为难以确定，解决的问题是找到每一个介词词组所修饰的那个词，例如：

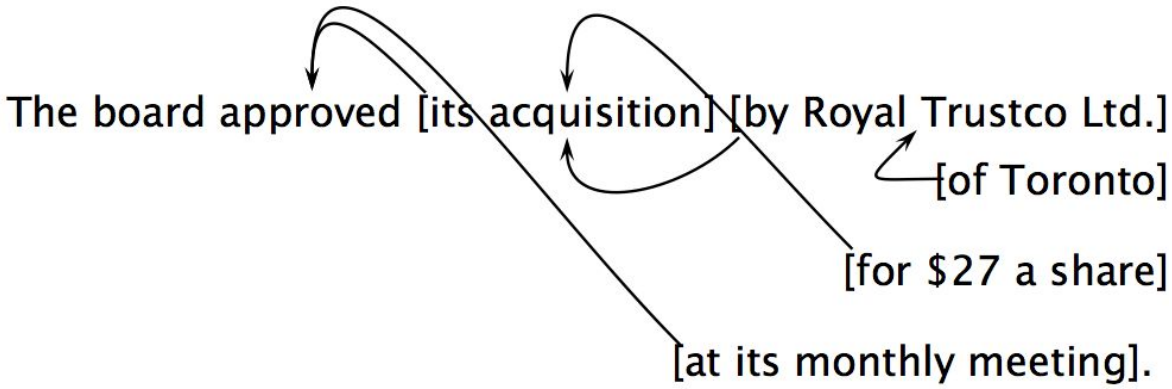


图 4.dependency parser 的例子

词组搭配问题的关键在于如何确定每一个词修饰的词，比如这边 its acquisition 修饰的是 approve，by royal trustco ltd 修饰的是 acquisition 等等，这是 dependency parser 里面最为困难的问题，我们现在的 parser 是不会关心上下文关系的，仅仅是依赖于句子进行处理，所以如图 5 所示：

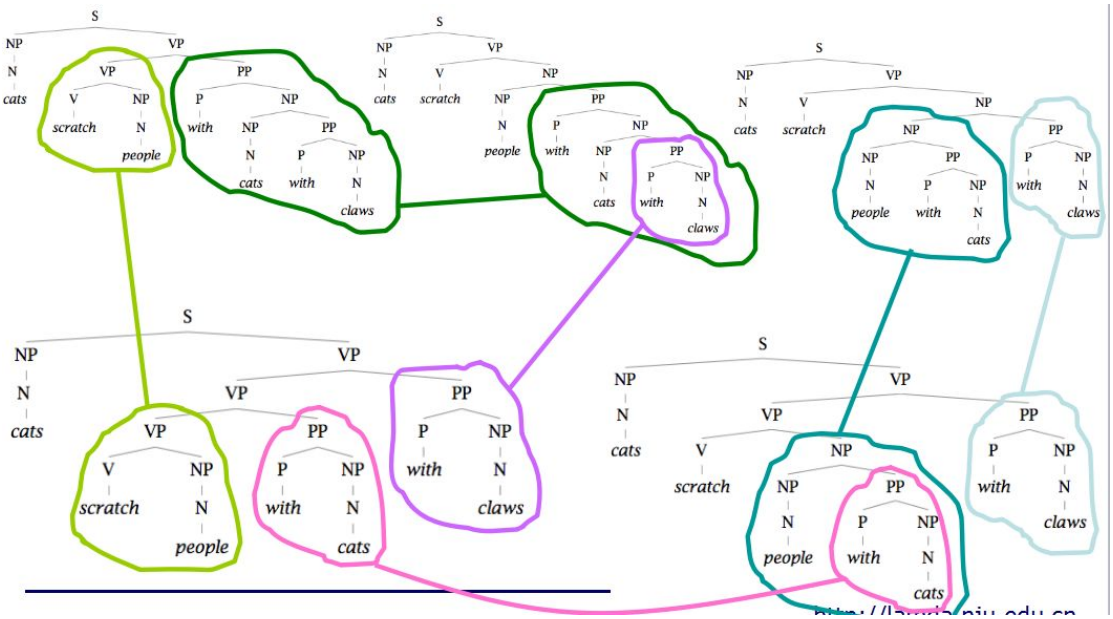


图 5.parser 带来的模糊性

对于 cats scratch people with cats with claws.共可以得到 5 种不同的 parser 结果。虽然很多是几乎错误的，例如图 5 中第三个 people with claws 这种修饰情况几乎是不会出现的，

但是仅在句子的基础上进行分析的话，那么 with claws 用来修饰 scratch 或者用来修饰 cats 都是正确的，猫用爪子转了一个带有猫的人，猫抓了一个带有猫的人都是可以理解的，只是前者出现的概率更大一些。但如果是 The man shot a bear in a blue shirt.在传统的 parser 中都不会用 shirt 来修饰 man，因为传统的 parser 是不存在跳跃性的，因为传统的 parser 需要最后生成一棵解析树，而树的生成则是按照从左到右的顺序构建，而且每个修饰完成之后都只会保留一个所谓的 head，就是被修饰的那个词，例如 beautiful flower 中 flower 就会成为 head，因为如果句子中删去 beautiful，那么句子的大体意思是不会改变的。而动词和名词结合，例如 shot bear,最终我们只会保留 shot，所以最后永远不会得到 shirt 修饰 man 的结果。

例如最近比较新的采用 RNN 算法实现的 Stanford Dependency parser 对于上述句子的 parser 结果为：

```
(ROOT
  (S
    (NP (DT The) (NN man))
    (VP (VBD shot)
      (NP (DT the) (NN bear))
      (PP (IN in)
        (NP (DT a) (JJ blue) (NN shirt))))
      (. .)))
```

很明显，这边将 shirt 用以修饰 bear，现在的 parser 拘泥于这种从底向上的结构，仅根据相邻的元素之间进行结合，一旦中间出现动词，而动词左右两边如果是存在修饰关系，则基本上就无法得到满意的结果，所以肯定也就无法得到上述的结果。这种有跨度的修饰仍然是一个比较大的问题。

我们针对这种情况提出了一种新的算法，具体参见 5.2。

5. 2　介词短语搭配算法。

我们在观察所有的合理搭配中，几乎所有的搭配都是不会相交的，举例来说，假设我们有 A,B,C,D 这样一个句子，其中绝对是不会出现 D 修饰 B,C 修饰 A 的情况。

如何解决这个问题？

我们首先对这些 A,B,C...分别进行索引，用 1,2,3...分别表示 A,B,C...

如何表示修饰的情况呢？

例如 C 用来修饰 A,则我们就用(1,3)来表示，后面的表示被修饰的词的索引，前面的表示修饰的词的索引。

什么情况算作是合理的修饰呢？

这边我们假设只要是不交叉相交，就算做是合理的情况。用符号表示就是，假设我们有两个修饰的情况(a,b),(c,d)

只要 b<=c 或者 a<=c 并且 b>=d，则我们就认为这是不想交的，而其他的情况我们就认为是相交的。

例如：cats scratch cats with claws with claws.

我们先进行词汇的还原，得到：cat scratch cat with claw with claw.

然后我们通过查找词库发现在介词 with 组成的词组的情况下(cat,claw),(scratch,claw)都是可以互相搭配的，而因为 people 作为名词是直接跟在动词 scratch 后面的，在这种情况下，我们会默认 people 是用来修饰 scratch 的。

Cat scratch cat with claw with claw.

0　1　　2　3　4　5　　6.

通过上述的算法我们可以得到：

(1 , 2) (2 , 4) (2 , 6)

(1 , 2) (2 , 4) (1 , 6)

(1 , 2) (2 , 4) (0 , 6)

(1 , 2) (1 , 4) (1 , 6)

(1 , 2) (1 , 4) (0 , 6)

(1 , 2) (0 , 4) (0 , 6)

此时我们就会发现：(1 , 2) (1 , 4) (1 , 6)两个 with claw 全都修饰了 scratch，(1 , 2) (0 , 4) (0 , 6)两个 claw 全部用来修饰第一个 cat，这样很明显是错误的 ，也存在连续两个 with claw 用来修饰第二个 cat(1 , 2) (2 , 4) (2 , 6)，而剩余的情况则是都能解释的通的，那么该如何去掉这两种情况呢？

我们采取的方法是，对所有的获取的修饰序列，如上所示，我们可以获得 6 个序列，我们从前往后扫描一遍，对于同一个词有多个词对其进行修饰的，我们对修饰的词进行比较，主要判断这些词是不是为同一个词以及它们前面的介词是否相同，例如这边都是介词 with，如果是，我们就放弃这个序列，这样我们就不会出现两个 claw 同时出现。此外，我们加上第二条规则，如果出现左侧相交的情况，例如（1,2）（1,6）则左侧就相交了一个 1，这样我们就降低这个序列的优先级，使得那些没有相交的序列排在前面，这边没有出现未相交的情况，所示最终我们得到的序列为：

(1 , 2) (2 , 4) (0 , 6)

(1 , 2) (2 , 4) (1 , 6)

(1 , 2) (1 , 4) (0 , 6)

而这两个理论上都是解释的通的。所以说这个算法还是挺合理的。

5.3 结论

本次实验中提出的算法在实验的过程中得到的效果相对来说是非常棒的。

5.4 参考文献

[1] http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Probabilistic.pdf

[2] http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Lexicalization.pdf

[3] http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf

[4] Michael Collins , Probabilistic Context-Free Grammars (PCFGs)

[5] Michael Collins, Lexicalized Probabilistic Context-Free Grammars

[6]https://zh.wikipedia.org/wiki/%E8%AA%9E%E6%B3%95%E5%88%86%E6%9E%90%E5%99%A8

[7] Advaith Siddharthan , Advaith Siddharthan , Advaith Siddharthan, Resolving Attachment and Clause Boundary Ambiguities for Simplifying Relative Clause Constructs (2002)

[8] Tejaswini Deoskar,Mats Rooth, Khalil Sima’an, Smoothing fine-grained PCFG lexicons,2009

6 句子拆分从句重构

6.1 简介

所谓的句子拆分从句重构，就是将一个完整的句子拆分为一个个字句，每个字句又可以单独存在，并且可以被准确翻译出来。例如：

一个长的完整的句子为：

I want to give the boy with a telescope which is made in China and is worth 100 yuan who is from China a book the beautiful girl bought for him written by my mother.

经过拆分后就变为：

a telescope which is made in China and is worth 100 yuan

the boy who is from China

the boy with a telescope

a book the beautiful girl buys for him

a book written by my mother

I want to give the boy a book .

这么做的灵感在于，我们发现在百度翻译谷歌翻译中中，长的组合的句子是很难被正确翻译的，但是一旦拆分为这些短的句子之后，就可以很准确的将短句子翻译正确。

例如上述的长句的翻译结果为：

我想给这个男孩用一个望远镜，这是中国制造，是值得 100 元谁是从中国一本书的美丽女孩买了他写的我的母亲。

而每一个短句的翻译结果为：

中国制造的望远镜，价值 100 元。

来自中国的男孩

我想给这个男孩一本由我母亲写的书

带着望远镜的男孩

我们发现长句的翻译明显存在很多错误，而短句的翻译虽然零散，但是却全部正确，如果我们能够将这些短句再进行正确的组合变成长句子，那么翻译结果的质量会有很大的提升。

但是这些都需要一些规则，这边我只找了一些简单的规则：

规则

1：名词(NN)+介词(IN)+which/who，例如：the place in which，此时我们对 NN 进行判定，如果 NN 确定是可以修饰 where,when 之类的，则表明这是一个字句，反之判断前一个（这边的前一个是指：I like the place which is like a big cat in which I grow up.这边 in which 是不可以修饰 cat 的，所以需要找到 the place 进行修饰）。

2：名词(NN)+动词被动式(VBN)+介词(IN),例如：The book written by my mother

3：名词(NN)+动词现在时(VBG)+介词(IN),例如：The boy playing basketball there

4：名词(NN)+介词(IN)+名词(NN)，例如：The boy in a blue shirt,the boy with a cat 等等，这边需要注意，如果介词不是修饰第一个名词的，它可能修饰前面的名词或者动词，而如果是修饰动词的我们就将其保留在当前句子，反之另外构造一个新的字句，例如：The boy with long hair in a blue shirt,后面的 in a blue shirt 是修饰 boy 的而不是修饰 hair 的，再例如：The boy hit the girl with long hair with a hammer,这边的 hammer 可以修饰 girl 也可以修饰 hit，假设其修饰 hit，我们就把这个句子完整的保留下来，最终为：The boy hit the girl with a hammer 和 the girl with long hair 两个句子

5：名词(NN)+WH(如 which,who,where,when...)开头的词或者 that,例如：The man who is...,

The cat which is sitting there...

6：名词(NN)+WH(如 which,who,where,when...)开头的词但是后者却不可修饰前者，例如：I put the drug where kids can not reach

7：名词(NN)+名词(NN)+动词(这边动词形式不是过去分词和现在进行时)，这边先判定第一个名词前的动词，如果是像 tell 这样的就需要注意省略 that 的情况，例如：I tell the boy the cat is not interesting,我们把 tell sb 后面可以加从句这种情况归为 3，而例如 give sb sth 后面不能加从句但是可以加名词的情况归为情况 2，而像 like sth 这种后面只可以跟一个名词的归为情况 1。如何区分呢？我们首先判定如果不是情况 3，那么就可以断定这是一个从句，例如 The book my mom bought for me,反之如果是 3，我们就认为后面的从句和前面的成份应该是一个整体，所以在中间添加一个 that 作为标记，同时也方便翻译。例如上述的 I tell the boy the cat is not interesting 就会变为 I tell the boy that the cat is not interesting.

1-7 都是名词起头的

8：动词(这边动词形式不是过去分词和现在进行时)+名词(NN)+动词(这边动词形式不是过去分词和现在进行时)，例如：I think my brother is right.

9：人称代词（例如 you...）++名词(NN)+动词(这边动词形式不是过去分词和现在进行时)，例如：I tell you the cat is not interesting...

10：介词(IN)+名词(NN)+动词(这边动词形式不是过去分词和现在进行时)，例如 I believe in my brother is right.

上述的规则主要用于查找字句的初始位置，那么如何才能判定其结束的位置呢？

因为这边只是一个参考的模板，此处仅列举一两个作为例子：

例如第一种情况（名词(NN)+介词(IN)+which/who），我们就判定：

①往后扫描如果发现当前扫描到的词是动词（并且不是动名词形式和过去分词形式），而且当前已经存在动词，也就是说如果现在将这个动词添加进来，则当前就会有两个动词，此时需要判定当前动词前的一个词是不是连词，如果是连词则添加进来，否在就终止；

②如果扫描到的是介词，并且后一个词不是名词也不是人称代词则终止；

③如果扫描到的是标点符号或者一些特殊的符号，则终止。

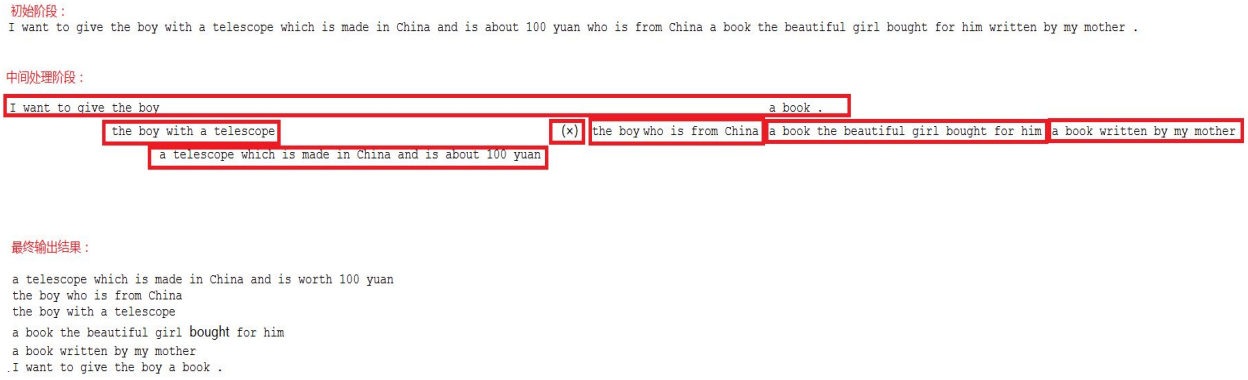
④如果扫描到的是名词，并且该名词前一个词也是名词，则终止。

⑤如果扫描到的是 WH 开头的例如 when，where 之类的词，则终止。

不同的情况会有不同的判定结束的方式。

6. 2　句子拆分算法

下面是简单的示意图。



算法伪代码：

```
getAllClauses(int type,int start,String mainw,String mainp,String
```

w,String[] mergeWords,String[] mergePos,String[] words,String[] Pos)

其中 type 表示的是字句的类型情况，初始化为-1，start 为开始的位置，mainw 是主体单词，mainp 是主体单词的 pos 标记，w 是包含主体单词的整个词组部分，mergeWords 是整个句子的主体单词，而 mergePos 是整个句子的主体 Pos,words 是整个句子的单词词组，Pos 是单词词组中每个单词的组合。

具体的细节参见附件中的 FindFixedPhraseandPrepmatch.java 中的 getAllClauses(int type,int start,String mainw,String mainp,String w,String[] mergeWords,String[] mergePos,String[] words,String[] Pos)函数。

6.3 实验结果：

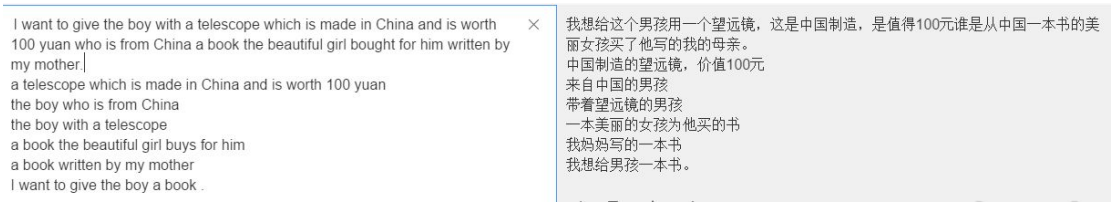
原句 1：

I want to give the boy with a telescope which is made in China and is worth 100 yuan who is from China a book the beautiful girl buys for him written by my mother .

结果 1：

```
a telescope which is made in China and is worth 100 yuan
the boy who is from China
the boy with a telescope
a book the beautiful girl buys for him
a book written by my mother
I want to give the boy a book .
```

翻译结果 1：



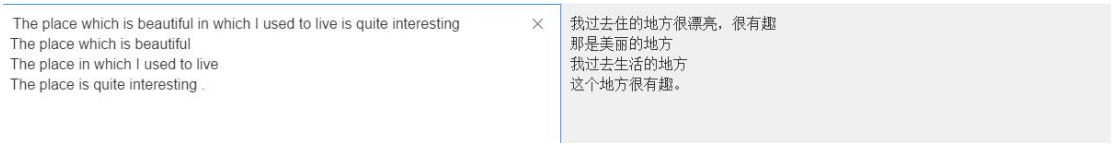
原句 2：

The place which is beautiful in which I used to live is quite interesting .

结果 2：

```
The place which is beautiful
The place in which I used to live
The place is quite interesting .
```

翻译结果 2：



6.4 结论

上述所讲算法的最大的亮点之处在于发现了现有翻译系统的一些优点，也就是对于短的

句子现有的翻译系统翻译的正确率是极高的，而对于一些长的带有组合的句子，翻译系统的翻译正确率是极低的，所以本算法的思想就是将长的句子转化为短的句子，对短的句子进行翻译然后对翻译之后的句子重组为长句子。

本算法有三大难点，第一就是如何判定句子的断点的起始位置，第二就是如何判定字句的结束位置，第三就是所有规则的处理。

此外，本次算法我也仅是处理了一些常见的简单情况，因为本算法是扫描式的，对于一些例如：The cat scratch the cat with claws 这些情况暂未处理，因为扫描到第一个 with claws 我们发现它是可以修饰 cat 的，所以就会将其表示为 the cat with claws,虽然这是一种合理的方式，但是毫无疑问这也带来了较大的问题，因为它修饰 scratch 也是可以的，所以这边是否可以像 5 中讲的一样全部枚举出来也是问题（因为这边只是做一个简单的模板，所以这个问题再本算法中暂时未解决）。

对于这样的问题毫无疑问是很有研究价值的，同时本算法也可以扩展采用机器学习的方式进行处理，因为没有数据集的缘故，此处就仅是采用了简单的规则枚举的方式。

6.5 参考文献

暂无。