NLP 期末项目 实验报告

下 - 原理与系统

一. 总体感想：

本项目 = 编译原理+大学英语 5（语法研究）

编译原理，是因为项目的重点，在于研究语法树。

大学英语，是因为研究语法树的重点，在于研究英语语法。

做这个项目最大的挑战，在于我的 parser 没那么给力（或许是因为 pyStatParser 只出到 0.0.1 版本，还是太弱），很多时候，parse 出来的结果，都是不正确的。

当我研究完了各种从句，满意地看着打印出来的 NN-VB 配对，然后上测试数据时，就会发现：明明是 VBZ，居然识别成了 VB；或者明明是动词原型，居然识别成了过去分词（come-came-come，这个怪谁呢？）；明明是 NNS（cars 这种），居然识别成了 NN。

这个时候，我就会想，假如用 JAVA+Stanford Parser，会不会好很多？然后就会想，既然 nltk 这么厉害，为何不搞个自己的 parser？（或许有，但是我在 google 和 stackOverflow 上没找到）居然还要自己定义语法规则才能 parse……

抱怨无益，后期处理中，我要为 parser 修补漏洞，这期间要做一些无奈的妥协，例如：

为了避免“NNS 居然识别成了 NN”这种情况，我把识别成 NN 的，以 s 结尾的名词，都改成了 NNS。

然而，有一个人叫做"David Adkins"……恩，只好牺牲他，换取大局了。

至于把 VB 识别成 JJ 或者 ADVP（如 rely），这个就只能彻底放弃了。

补丁不断打，程序变得臃肿而奇怪，一点都不 pythonic。作为刷 OJ 的孩子，感到痛心疾首。

后来我想明白了，对于自然语言这种千变万化的东西，无论哪个 parser，都不能完美通杀，毕竟这不是 OJ 的算法题。所以，后期处理也是很好的学习过程。

不能做到完美，只能接近完美，这是 NLP 最大的挑战，或许也是它最大的魅力吧。

二. 系统架构：

整个 main.py，大概分 3 个阶段：

1. 前期处理：parse 出语法树

2. 后期处理：修补语法树，寻找出合适的 NN-VB 配对

3. 输出答案：根据 NN-VB 配对是否合法，输出答案

下面，分三个章节来叙述。

三. 前期处理：

由于本人之前从未用 JAVA 做过项目，所以放弃了 Stanford Parser，改用 python+nltk

用 nltk 试验了 POS tagging 和 Chunking，但是由于找不到内置的 parser，于是去 GitHub 上 clone 了 pyStatParser 下来。（一开始想过不 parse，发现不可能，比如定语从句 He is a man who runs fast，不 parse 很难找到 man 和 runs 的配对）

pyStatParser 有一部分内容，比如语法树可视化，是依赖 nltk 才能有的。语法树可视化在我前期分析的时候，起了很大作用，向作者表示感谢。

pyStatParser 是通过 QuestionBank and Penn treebank 里面的一些语料（%1 左右），用一个统计的 CKY 算法，parse 出语法树的。这个算法，据我观察，似乎是动态规划，O(wordNum^3)，对付短句秒杀，长句还是很弱，前面已有所叙述。

声明一个 Parser，调用 raw_parse()方法，可以得到 list 格式的语法树：

```
indentTree = parser.raw_parse(line)      # raw parse for indent tree list
```

当然这里要用 try except 来避免报错，一个 error 会打断之后所有的测试数据。对于 parsing error 的句子，直接 print 'Parsing Error'然后 continue

然后，我定义了 class node，加了很多方法。写了一个 trans 函数，将 list 格式转换成 node 格式的语法树，并得到 root：

```
root = trans(indentTree)                    # recursively transform list to "tree of node"
```

前期处理到这部分，就算结束了。config.py 中，1 到 7 行的变量，可以随便设为 1，查看句子和语法树的初始状况。

四. 后期处理：

首先要遍历一遍树，给每个叶结点加编号，并建立 nodeList，这个由 assignCode(root)函数完成。

assignCode()函数还有很重要的任务，修补语法树，举个例子：

```
if root.getData()[last]=='s' and root.getParent().getData() in [u'NN', u'NNP']:
            root.getParent().setData(u'NNS')
```

这段代码，可以将识别成 NN 的，以 s 结尾的名词，都改成 NNS。

不止修补了这一点。我的每个函数，前面都有注释，写的关键之处也有注释，可以参考。

接下来，默认的 parse()方法，可以得出可视化语法树：

```
nlktTree = parser.parse(line)          # nlktTree, could be drawn into graph
```

之前提到过了，这个功能需要 nltk package 支持。

可视化语法树主要用来分析和 debug，还是很重要的。

然后重头戏开始了，开始 check。我写了两个方法，preCheck()和 totalCheck()

preCheck()主要考虑的是，有些动词，它没有名词依赖，但是不能用第三人称单数，比如：

I have met you before.（时态决定）

I need to talk. （特殊词 to）

I must go now.（情态动词(MD)决定）

这种我放在最前面检测，检测方法就是，假如一些特殊的单词，或者是情态动词，后面跟了动词第三人称单数，那么就是错误，放入 errorCode 中。

这里展示一下 MDList：

```
MDList = ['can', 'could', 'may', 'might', 'must', 'need', 'dare',
'dared', 'shall', 'should', 'will', 'would', "'ll", "'d"]
```

totalCheck()是最费精力的部分，它站在句式的角度，寻找 NN+VB 配对。

首先，我放弃了某些句型，如主语从句(Subject Clause)和 there-be 句型，这两种太过复杂，而且 parsing 一直很混乱，只好放弃。碰到它们，一般直接输出-1 然后 continue.

然后，处理特殊疑问句（Special Question）和一般疑问句（General Question），这部分的难点，是存在一个 VB+NN 的配对（What do you... Is he...），以及一个特殊的 NN+VB 的配对，特殊的意思，是这个配对中的 VB，必须用原型。

为了找这两个配对，写了两个函数 findFirstVBNN()和 findFirstNNVB()，同时还要判断找到的两个 NN 是否一样，着实费了一番功夫。

这里闲话一句：28 定理在 CS 领域一直适用。这个项目假如想偷懒，POS Tagging 一下，找 NN+VB 的序列，估计也能找到八成，像我这样血泪地分析语法，最后也只能稍微好那么一点点，仔细想来，这是为了情怀，为了业界良心。

接下来考虑宾语从句（Object Clause），定语从句（Attributive Clause）和状语从句（Adverbial Clause）。

这里面，状语从句是最好的，没有特殊情况要考虑。

宾语从句也不错，大部分都没什么担心的，少部分（如 I wonder whose bike is put here）在之后也可以通过别的方法发现，无所谓。

定语从句很有搞头：

1) 像这样的语法树：root -> NP + SBAR + VP ,其中 NP 与 VP 有依赖关系，果断找出。

2) 像这样的子树：NP + SBAR ，SBAR 中假如开头为 WHNP，WHNP 之后为 VP，那么这个 VP，必然依赖之前的 NP（如 He is a man who runs fast），找出。

恩，主要就这两点，并且第二点出现更频繁。

接下来，就是找 NP+VP 这样的子树了，这是陈述句（Declarative Sentence）最多见的结构，NP 的最右后代结点，和 VP 的最左后代结点，一般互相依赖，找出来。

totalCheck()到此结束。之后还有一轮遍历 nodeList，找 NN+VB 的漏网之鱼，这个是为了防止语法树 parsing 不正确，该有的 NP+VP 子树没有，而做的修补工作。前面所叙的 I wonder whose bike is put here，这里可以找到。

这部分还有很多小技巧，比如 RB（如 often, once）要跳过，这里不赘述了。

五. 输出答案：

既然 NN-VB 配对都有了（严格来说，我们程序可控范围内的都有了，可控范围外的依旧逍遥自在），开始判断动词正误。核心代码如下：

```
    for i in range(len(NNNode)):
        n = NNNode[i]
        v = VBNode[i]
        If n.getData().lower() in PRPThirdList or
n.getParent().getData() in [u'NN', u'NNP']:      # single noun
            if v.getParent().getData() in [u'VB', u'VBP']:
                        errorCode.append(v.getCode())
                else:
                    if v.getParent().getData()==u'VBZ':
                        errorCode.append(v.getCode())
```

然后打印 errorCode，完工。

六. 项目感想：

一路写下来，时钟指向了凌晨三点。之前也间杂着写了很多感想了，这里再补充点。

首先，细节很重要。python 对中文的支持一般，很大的感受就是切换中文输入法，删除东西的时候，出现的是小方框，然后必须切回英文才能删掉。更加坑的是，python 对缩进要求严格，必须是四个空格，而 python IDLE 中，英文输入法按 tab 出四个空格，中文输入法下按 tab 输出的是 tab，编译的时候报 Unexpected Indent Error，百思不得其解，不是都一样

么？其实，编译器认为它们并不一样。

还有，python 的 IDLE 不是很聪明，有的时候必须关了再看，才会看到：之前怎么看都是正确的缩进，关了再开，居然隔着 n 个空格。重启大法好，古人诚不我欺。

然后，规范很重要。经过这一番钻研英语语法，我才发现，相比于一个强大的 C++编译器，假如真有一个强大的英文语法编译器，那才叫震古烁今，惊为天人。英语语法，大框架上不错，细节上还是有点乱，你怎样才能教会编译器搞定 Long Time No See，搞定 Gotta be kidding me 呢？statistical based 这时候估计可以发挥一番，rule based 估计 parsing error 吧？英语语法相对来说已经不错了，还这么乱，那你让中文这种“怎么说都行，怎么说都有新的意思”，“现代语法已经毁灭，文言文倒是不错”（via Hai Zhao）的语言，怎么玩？

所以，假如以后哪位大人物能下狠心，搞一种语言，语法规范成 context-free grammar 强制推广，那么对于人类社会的进步，或许是有益的……

最后，好玩最重要。之前也说了，这里再说一遍：

“对于自然语言这种千变万化的东西，无论哪个 parser，都不能完美通杀，毕竟这不是 OJ 的算法题。

不能做到完美，只能接近完美，这是 NLP 最大的挑战，或许也是它最大的魅力吧。”

模糊之美，大学里之前还没有一门课，能带给我这种感觉。

所以，这个项目，这门课，还是很好玩的~

七. 谢辞

感谢 Hai Zhao 老师一学期的辛勤授课！做项目，讲 paper，获益匪浅！

感谢 Chuck Chen 同学的答疑解惑！叨扰了您数十封邮件，thanks for your time!

青山不改，绿水长流，bug 恒在，debug 长存。后会有期！

Tianze Wang

Jun. 11th, 2015

3:24 a.m.