ReKan：基于隐马尔可夫模型的对仗生成

章凌豪

13307130225@fudan.edu.cn

1 Introduction

1.1 Motivation

对联生成是一个很有趣的问题，它需要我们结合恰当的统计语言模型以及关于对联的语言知识。

如果我们将对联的定义放宽到**对仗文本**，那么对联生成可以有更丰富的应用。一个容易想到的应用是在用于在写文章或写歌词时为我们提供灵感（寻找与一个相对的意象，或是寻找对仗的词语用于前后句中）。ReKan [1] 就是这样一个基于隐马尔可夫模型的可以自动生成对仗文本的系统。

1.2 Overview

程序用 **Python 2.7** 编写

提交文件说明：

data/corpus.zip	训练用语料
src/commons.py	定义一些常量
src/gen_couplets.py	候选重排序以及生成下联的接口
src/hmm.py	HMM 参数计算
src/process_corpus.py	语料预处理
src/train.py	整合训练流程
src/utility.py	统计 Ngram 和建哈希表等辅助函数
src/viterbi.py	Viterbi 算法
ReKan.exe	可执行文件

为了方便叙述，报告中一律用**对联**指代我们所处理的**对仗文本**

[1] ReKan 是日语中“灵感”的罗马音

2 Implementation

2.1 Corpus Processing

语料主要来自于《古今实用楹联集成》、《古今对偶句选》、《唐诗 *300* 首》，以及从网络上收集的一些对联和绝句。

在预处理阶段主要是将语料中所有对仗的句子提取出来转换成方便后续处理的格式。

2.2 Language Modeling

ReKan 采用**隐马尔可夫模型**，将每个字看做一个状态。上联的字之间的关系用**转移概率**来描述，而上联某一个字到下联对应位置的字之间的关系则用**输出概率**来描述。

记上联为 $x_1, x_2, ..., x_n$，下联为 $y_1, y_2, ..., y_n$。同时用 $x_i \| y_i$ 来表示一个字对。

首先对语料进行 $Unigrams$ 和 $Bigrams$ 的统计。除此以外还要统计所有的字对 $WordPairs$。

ReKan 假设上下联是完全对称的，所以对于每个对联，下面的计算都要在对调上下联之后重复一遍。

转移概率可以由 $Bigrams$ 和 $Unigrams$ 得到：

$$\forall x_i x_j \in Bigrams \quad P_{transition}(x_i \mid x_j) = \frac{Count(x_i x_j)}{Count(x_i)} \tag{1}$$

输出概率可以由 $WordPairs$ 和 $Unigrams$ 得到：

$$\forall x_i \| y_i \in WordPairs \quad P_{output}(x_i \mid y_i) = \frac{Count(x_i \| y_i)}{Count(x_i)} \tag{2}$$

2.3 Candidate Generation

由以上的计算可以得到 HMM 的所有参数。所以对于一个给定的上联，可以用 **Viterbi 算法**求得输出概率最大的 k （这里取 20）个候选下联。不再赘述。

2.4 Candidate Ranking

最后 ReKan 通过一个评分函数对候选答案进行重排序，将前 5 个反馈给用户。

目前，评分函数使用以下指标：

1. 平仄相对

定义 $pz(x)$ 并用 1 和 -1 分别表示平直和曲折。根据**平仄相对**的原则，计算：

$$P_{pz} = \frac{\sum pz(x_i) + pz(y_i) = 0}{n} \tag{3}$$

注：ReKan 假设现代汉语中的平声和上声为平直，去声和入声为曲折。

2. 词性相对

可以认为，通过 Viterbi 算法得到的候选答案从结构来讲已经比较令人满意了。那么出于对仗的目的，我们可以重新将字对之间的匹配度纳入考虑。

首先可以考虑的就是词性。这里将同类的 POS tag 也看做相同。

$$P_{pos} = \frac{\sum pos(x_i) = pos(y_i)}{n} \tag{4}$$

3. 输出概率

类似地，可以将字对之间的输出概率重新纳入考虑。

$$P_{out} = \frac{1}{1 + e^{-\prod P_{output}(x_i \mid y_i) * amp}} \tag{5}$$

其中，为了避免概率太小导致浮点数舍入到 0，以及最终标准化到 $0 \sim 1$ 的分数，给每项乘上一个 $amp = 10^{\sqrt{n}}$ 的**增幅项**，最后再套一层 **Sigmoid** 函数。

以上的分数如有为 0 的均用其最大可能值乘以 0.001 做**平滑处理**。

最后将每项分数相乘即得到最后的分数。

3 Demonstration

可以运行 **ReKan.exe** 来通过命令行尝试 ReKan。

要注意默认编码为 **GBK**（Windows 命令行默认编码），如有必要可以将编码作为参数传给程序，否则无法正确解码中文。

运行截图如下：

```
C:\Home\Projects\ReKan\submit>ReKan.exe
Specified encoding: gbk
Please input the first half of the couplet. Input "q" to leave.
天涯何处无芳草
Building prefix dict from C:\Home\Projects\ReKan\data\dict.txt ...
Loading model from cache c:\users\linghao\appdata\local\temp\jieba
5f011d293e56be44a745.cache
Loading model cost 0.476 seconds.
Prefix dict has been built succesfully.
海上谁家有美人 0.491716715286
海上谁家有幽林 0.409859429571
海上谁家有绿云 0.369002286714
海上谁家有落花 0.246287858143
海上谁家有绿树 0.205287715286
Please input the first half of the couplet. Input "q" to leave.
```

下面是一些有趣的结果：

1. 天涯何处无芳草　海上谁家有美人
2. 孤帆远影碧空尽　别路浮光青不欢
3. 春风送暖百花开　日月催寒七叶落
4. 日月随风一叶落　年岁逐露千条归
5. 断肠人在天涯　离别客从海路
6. 浮云游子意　明月美人心
7. 久旱逢甘霖　深涸值慈云
8. 鸟宿池边树　花开楼上泉
9. 月缺花残　风摇草枯
10. 流水无情　落花有迹

4 Discussion

本小节讨论一些目前实现的不足以及未来可以改进的方向。

4.1 Language Unit

目前的语言单元是字级别的，这带来很多问题，比如很多时候一对词组对仗得很好但拆成单个字对后匹配度并不高。尝试过用分词器来拆语言单元，但遇到了上下联语言单元不对齐以及多元组概率缺失的问题。前者可能需要通过借鉴机器翻译中的**对齐**技术来解决，后者需要精心设计合理的 **Fall back** 机制。

4.2 POS Tagging

目前在候选重排序时所用的 POS Tagger 是通用的，而非针对所用的语料（大多为偏白话的文言风格），所以会有较多的错误，且无法捕捉到一词多性。这些缺陷虽然会导致一些好的候选的分数落后，但由于候选的质量都已经比较高，所以最后的结果还是能够接受的。这方面的改进要依赖于在语言单元上取得的进展。

4.3 Repeated Words

目前对叠字的处理是，将 Viterbi 算法求得的叠字所对应的输出用来填充该字所有出现的位置。这样做没有考虑到同一个字在联中的不同出现可能有不同的含义。目前没有什么更好的处理办法，或许在针对语料改进 POS Tagging 并将词性整合到模型中后能有所改进。

4.4 Pair Pre-Mining

对于一些常见、固定的意象，可以先通过 **Frequent Item Mining** 构建合法的**对仗意象库**。在输入中遇到这样的意象时，可以直接从库中随机选取一个作为输出。这样不仅在部分输入下表现更为稳定，也增加了输出的多样性。同时这样的信息也可以直接用来对语料中的对仗进行文学和语言学上的分析。这可能要依赖于一个针对语料训练得到的性能过得去的分词器。

4.5 Semantic Constructing

为了获得语义上更一致的结果，可以先分析上联中的**中心词**，然后将它的对仗词作为下联的中心词，在计算输出概率或者候选重排序阶段加入对应该中心词的**奖励项**（简单地使用 Bigram，或者 Word2Vec 这样的表征方法）。