

# 自动作诗系统

## 计算语言学 大作业

### 1. 实验目标

唐诗是中国的传统文化之一，它对仗工整，平仄协调，是一字一音的中华语言独特的艺术形式。本次实验的目的，是综合运用课上所学知识，设计实现一个自动作诗系统，通过解决实际问题，进一步加深对计算语言学的理解。

### 2. 实验准备

#### 2.1 实验环境

- Ubuntu 14.0.4
- Python 2.7
- Python 库函数：jieba、flask

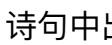
#### 2.2 使用语料

唐诗语料库.txt

### 3. 实验步骤

#### 3.1 预处理“唐诗语料库.txt”

观察到给定的唐诗语料库存在以下噪声：

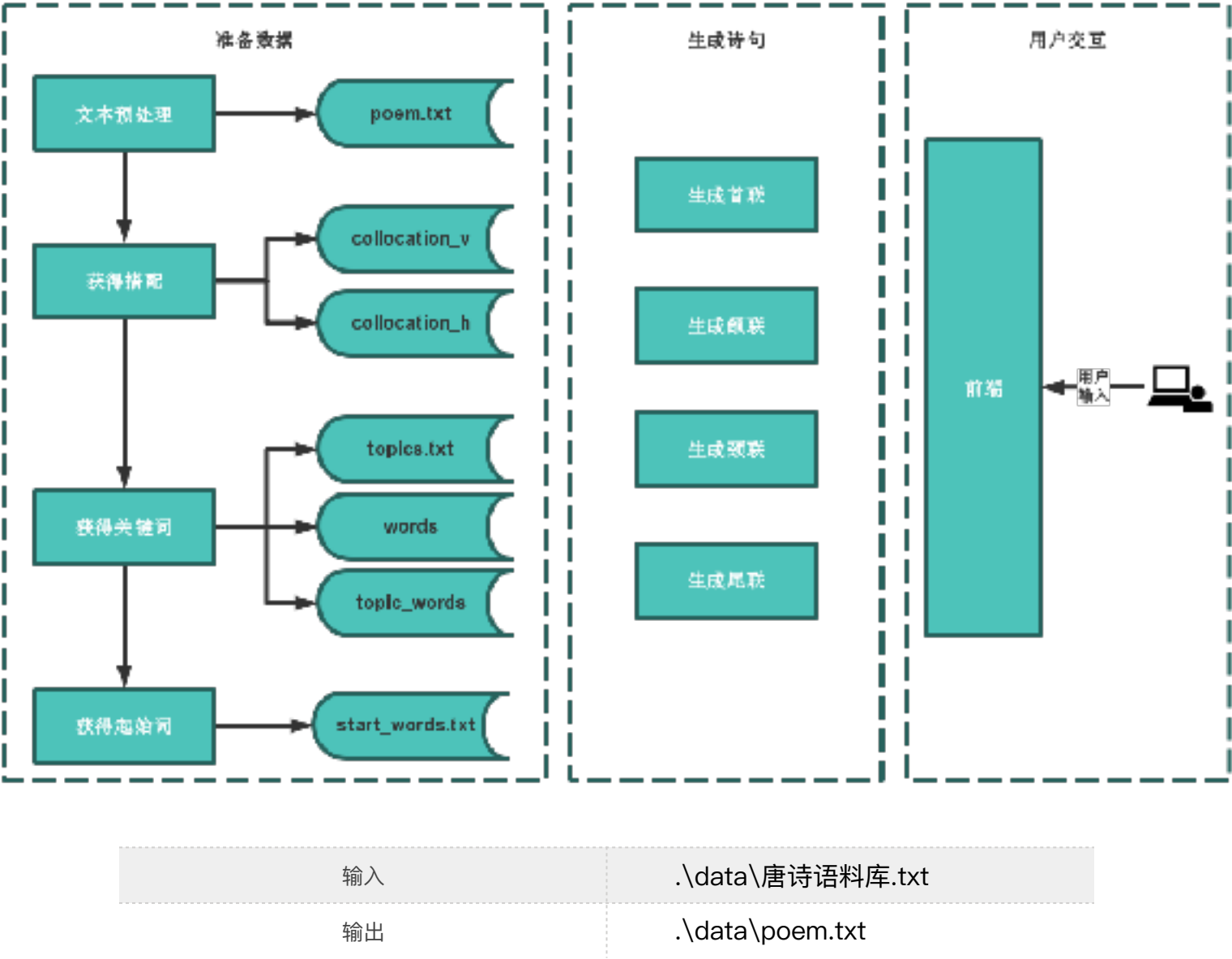
- 诗句中出现类似的HTML标签。
- 出现空格、“.”等字符。
- 诗句中出现注释，用“（”、“）”标出来。
- 诗句不完整，出现方框字符。

对于前三种情况的噪声，直接去掉即可。对于最后一种噪声，直接把这行诗句忽略考虑。（此外，对于第三种噪声，“（”、“）”不在同一行时未处理。）

由于暂时只需要用到唐诗标题和诗句，故只提取这两部分内容。

源文件

preprocess.py



3.2 分词

对上联分词，然后找到对应的下联。看看效果。如果扔掉上下句切词方式不同的 pair，感觉本来不大的语料库就浪费很大。

对于中文分词，这里采用在工业界上较广泛应用的“结巴”中文分词组件。该分词组件主要采用以下算法：基于Trie树结构实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图（DAG）；采用动态规划查找最大概率路径，找出基于词频的最大切分组合；对于未登录词，采用了基于汉字成词能力的HMM模型，使用了Viterbi算法。

由于唐诗中的每一个字基本都是有用的，故停用词（Stop Words）主要为标点符号，这里直接使用默认的停用词。

### 3.3 获得搭配信息

搭配包括横向搭配和纵向搭配。横向搭配指每句诗中每个词与下一个词的搭配关系，纵向搭配指每两句诗中，第一句诗中的词与下一句诗中对应相等长度的词的搭配关系。

分词之后把唐诗（不含标题）按句子切割，对句子总数为偶数的唐诗，遍历每两句诗，第一句诗中的词与第二句诗中对应相等长度的词形成一个纵向搭配。对每一句诗，每两个词形成一个横向搭配。

易知，使用似然比、频率、t检验等搭配发现方法都能得到较好结果，这里为了方便，直接使用频率来发现搭配。

源文件	get_collocations.py
输入	.\data\poem.txt
输出	横向搭配.\data\collocations_h
	纵向搭配.\data\collocations_v

### 3.4 获得主题信息

对每首诗，提取TF-IDF特征并构建矩阵，然后使用非负矩阵分解提取唐诗主题类别。考虑到唐诗分类数量有限，这里只生成10个类，每个类用频率最高的20个词来表示。

源文件	get_topic.py
输入	.\data\poem.txt
输出	主题.\data\topics.txt
	词.\data\words
	每个主题-词对应的得分. \data\topic_words

### 3.5 生成起始词

对每首诗，分词后取第一句诗的第一个词作为起始词。统计所有起始词，并输出出现超过两次的词。

源文件	get_start_words.py
输入	.\data\poem.txt
输出	起始词.\data\start_words.txt

### 3.6 生成唐诗

输入的参数除了上述生成的部分文件（如搭配、主题等）外，还需要指定诗句数量、诗句长度、主题和起始词（若不指定则随机产生）。

对于给定诗句长度 $l$ ，起始词 $start\_word$ 和主题 $topic\_id$ ，设 $a[i]$ 为第 $i$ 个词的id，我们可以把产生第一句诗抽象成一个子问题：

$$\begin{aligned}
 \max \quad & \prod_{i=2}^n collocation\_h\_score[a[i-1]][a[i]] \\
 & + \lambda \sum_{i=1}^n topic\_word[topic\_id][a[i]] \\
 \text{s.t.} \quad & \sum_{i=1}^n len(word[a[i]]) = l \\
 & a[1] = start\_word
 \end{aligned}$$

其中 $collocation\_h\_score[a[i-1]][a[i]]$ 表示第 $i-1$ 个词与第 $i$ 个词的横向搭配分数， $\lambda$ 为平衡参数。若以上问题的最优解为 $a[i]$ ，那么所生成的较为合理的第一句诗即 $word[1]$ ,  $word[2]$ , ..... $word[n]$ 。

显然，对于该问题，可以把目标函数中的乘积部分用log来使其变成求和。于是该问题可以用动态规划来求解：

设 $f[i][j]$ 表示长度为 $i$ ，最后一个单词id为 $j$ 的最大目标函数值，则

$$f[i][j] = \max\{f[i - len(word[j])][k] + \log\_collocation\_h\_score[k][j] + \lambda topic\_word[j]\}$$

其中 $(k,j)$ 为一个横向搭配。

初始时 $f[len(start\_word\_id)][start\_word\_id] = \lambda topic\_word[start\_word\_id]$

最后最优值为 $f[l][j]$ ，对所有 $j$ ，路径可通过与 $f$ 同大小的矩阵 $pre$ 来记录前一个单词的id。

产生下一句诗，则需要考虑纵向搭配。同理我们也可以把产生下一句诗抽象成一个子问题：

$$\begin{aligned} \max \quad & \prod_{i=2}^n \text{collocations\_h\_score}[a[i-1]][a[i]] \\ & + \lambda_1 \prod_{i=1}^n \text{collocations\_v\_score}[\text{pre\_a}[i]][a[i]] \\ & + \lambda_2 \sum_{i=1}^n \text{topic\_word}[\text{topic\_id}][a[i]] \\ \text{s.t.} \quad & \text{len}(\text{word}[a[i]]) = \text{len}(\text{word}[\text{pre\_a}[i]]), i = 1, \dots, n \end{aligned}$$

其中pre\_a[i]表示上一句诗的第i个词的id，collocations\_v\_score[pre\_a[i]][a[i]]表示上一句诗第i个词与这一句诗第i个词的纵向搭配分数， $\lambda_1, \lambda_2$ 均为平衡参数。同理也用动态规划来求解：

设f[i][j]表示第i个词，最后一个单词id为j的最大目标函数值，则

$$\begin{aligned} f[i][j] = \max \{ & f[i-1][k] + \log\_collocations\_h\_score[k][j] \\ & + \lambda_1 \log\_collocations\_v\_score[\text{pre\_a}[i]][j] \\ & + \lambda_2 \text{topic\_word}[j] \} \end{aligned}$$

其中(k,j)为一个横向搭配，(pre\_a[i],j)为一个纵向搭配。

初始时f[0][j]=max{ $\lambda_1 \log\_collocations\_v\_score[\text{pre\_a}[i]][j] + \lambda_2 \text{topic\_word}[j]$ }

求最优值与最优解方法同上。



### 3.7 实现网站

为了更好的用户体验，使用Flask简单搭建了一个自动作诗系统的网页。

若是用户没有输入，则随机生成唐诗；若是用户输入第一句诗或更多句诗，则生成剩下的诗。



## 4. 参考资料

1. “结巴”中文分词. <https://github.com/fxsjy/jieba> ↩
2. TF-IDF. 维基百科. 最后修订于2015年9月27日. <https://zh.wikipedia.org/wiki/TF-IDF> ↩
3. sklearn.feature\_extraction.text.TfidfTransformer. scikit-learn developers. [http://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfTransformer.html](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html) ↩
4. Non-negative matrix factorization. Wikipedia. [https://en.wikipedia.org/wiki/Non-negative\\_matrix\\_factorization](https://en.wikipedia.org/wiki/Non-negative_matrix_factorization) ↩
5. sklearn.decomposition.NMF. scikit-learn developers. <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html> ↩
6. He J, Zhou M, Jiang L. Generating chinese classical poems with statistical machine translation models[C]//Twenty-Sixth AAAI Conference on Artificial Intelligence. 2012. ↩