# 中文信息处理大作业

## —宋词自动生成器

计算机科学学院 于竟成 12307130164
计算机科学学院 王舒敏 12307130359

**【问题描述】**

编写一个轻量级自动化程序，在给定词牌格式和主题词的情况下生成宋词。

选取这个主题的灵感来自于前一阵子网上流行的“汪峰作词法”。汪峰的歌挺不错的，而且歌词也写得好。不过许多网友通过观察却发现他的歌有很多共性。于是就有人统计了他发行的117首歌的歌词，并得到了如下表格：

| | 形容词 | | 名词 | | 动词 |
|---|---|---|---|---|---|
| 0 | 孤独：34 | 0 | 生命：50 | 0 | 爱：54 |
| 1 | 自由：17 | 1 | 路：37 | 1 | 碎：37 |
| 2 | 迷惘：16 | 2 | 夜：29 | 2 | 哭：35 |
| 3 | 坚强：13 | 3 | 天空：24 | 3 | 死：27 |
| 4 | 绝望：8 | 4 | 孩子：23 | 4 | 飞：26 |
| 5 | 青春：7 | 5 | 雨：21 | 5 | 梦想：14 |
| 6 | 迷茫：6 | 6 | 石头：9 | 6 | 祈祷：10 |
| 7 | 光明：6 | 7 | 鸟：9 | 7 | 离去：10 |
| 9 | 理想：6 | 8 | 瞬间：8 | 8 | 再见：9 |
| 9 | 荒谬：5 | 9 | 桥：5 | 9 | 埋：6 |

然后随机取一个数字，将其对应的词语连接起来并稍加润色就可以组成很有汪峰风格的歌词。例如圆周率3.1415926，对应的词为坚强，路，飞，自由，雨，埋，迷惘。连接之后可以组成：

坚强的孩子，依然前行在路上，张开翅膀飞向自由，让雨水埋葬他的迷惘。

不过这个例子里还是需要手动调整的，那么能不能做出一个完全机器自动生成的歌词呢？略加尝试后发现，非常大的一个问题在于歌词的格式。如果没有固定的格式要求，完全自由的编写的话，

计算机程序很容易弄出完全不通顺的句子来。同时又考虑到宋词是由固定的词牌格式要求的，包括字数，平仄，韵脚等等都非常严格。并且古文言简意赅，宋词长度一般较短，所以是一个很好的切入方向。

诗歌的机器自动生成并不是一个冷门的主题，国内外很多人都有在做，而且做宋词生成的也已经有了先例[1]。这篇文章所使用的方法相当复杂，同时也卓有成效。然而，本次课程设计并不准备重新实现一遍该论文的方法，而是使用python以及nltk工具包制作一个轻量级的简单算法。

## 【总体思路】

1.使用全宋词作为基本语料库，以及字典库来提供拼音。

2.处理全宋词，提取高频词以及词之间的转移概率

3.根据主题提取主题词，在生成时提高使用主题词的概率

4.分别使用马尔科夫模型以及遗传算法，用这两种方式根据词牌格式生成宋词。

## 【分工】

处理词牌格式，使用马尔科夫模型生成宋词，编写文档：于竟成

处理原始数据，提取高频词、主题词以及词之间转换概率，使用遗传算法生成宋词：王舒敏

## 【具体过程】

### 一、搜集语料

语料主要采用《全宋词》和《汉语拼音数据库》（来源于互联网）。

### 二、处理原始数据

1. 《全宋词》

找到的这个版本的《全宋词》是txt文件，每个词占两行，第一行为词牌名，第二行为内容。文本内容中夹杂了一部分广告网页以及词的注释。将每行字符数大于10且小于100的作为有效内容，拼接起来，并去除所有标点符号，组成一个纯内容的文本。

2. 《拼音》

---

[1] 周昌乐, 游维, and 丁晓君. "一种宋词自动生成的遗传算法及其机器实现." Journal of Software 21.3 (2010): 427-437.

这是一份数据库文件，包含的内容比较多。主要提取出了每个字的拼音，组成了形如“中zhong1”格式的文本，标识了字的拼音和音调。多音字保留了所有发音。

三、提取高频词

宋词创作的特点在于每小句字数较少，基本不超过十个字，并且较为紧凑。于是考虑提取候选词汇，使用词汇堆砌的方式作词。但一般的分词软件处理古文效果并不是特别好，可以使用最为简单暴力的方式解决。将全宋词中所有长度为二或者三的子串全部提取，并建立频率表，选择频率最高的词作为候选词汇。截取频率最高的十个的二字和三字词：

| 二字 | 三字 |
| --- | --- |
| 东风 | 倚阑干 |
| 何处 | 知何处 |
| 人间 | 广寒宫 |
| 风流 | 到如今 |
| 归去 | 东风吹 |
| 春风 | 留不住 |
| 西风 | 人何处 |
| 归来 | 有谁知 |
| 江南 | 三十六 |
| 相思 | 西风吹 |

可以看出虽然未使用分词算法，高频词汇也都是有一定意义的。

当然，这种方法也不可避免有一些问题。尤其在三字词中，会常常出现将四字词语截取三个出来的情况。例如“知何处”排在了第二位，而“不知何”排在了第十三位。直观来说“知何处”和“不知何处”在宋词中都是会经常出现的。但是这种提取方法就会出现“不知何”这种奇怪的词。同样的还有排在60位的“个人人”以及63位的“一枝春”。

消除这种词的方法是可以进一步统计四字词和五字词，并且判断是否有常用词是另一个的 子串。

以下是词频最高的十个四字和五字的词：

| 四字 | 五字 |
| --- | --- |
| 不知何处 | 七十古来稀 |
| 不如归去 | 何处难忘酒 |
| 人间天上 | 人间何处难 |
| 明月清风 | 三万六千场 |
| 归去来兮 | 间何处难忘 |
| 落花飞絮 | （商调十二 |
| 广寒宫殿 | 商调十二首 |
| 天上人间 | 调十二首之 |
| 江南江北 | 心事有谁知 |
| 一觞一咏 | 侍玉皇香案 |

其中“（商调十二”是原文本的注释中常出现的句子。实际上这样不符合规范的词还有很多，不过之后都通过处理删除掉了。

四字词的第一个是“不知何处”，正好三字词的“知何处”和“不知何”都是它的子串。但如果只是简单地把这两个子串都删除的话，正常情况下也会出现的“知何处”就不能生成了。而同时五字词实际上并不是那么常用，权衡之下我决定将四字词加入候选词库。

至此便将所有高频词汇，也就是候选词汇提取了出来。

**四、计算词之间转移概率**

将每一首词去掉标点联结在一起之后，就可以用来统计搭配的频率了。之所以将一首词作为一个单位，而不是一个分句，是考虑到即便有标点符号的间隔，下一分句的开头与上一分句的结尾也是有一定联系的。

这里选择了仅仅统计相邻关系，即一个词紧接着的下一个词是什么。不选择前两个推出后一个的原因是，这样的话生成速度会慢很多。并且实际运行之后发现效果并没有显著的区别，于是为了实验的方便就舍弃不用了。

一些转移概率的例子如下：

| | | |
|---|---|---|
| 旧时 | 携手 | 0.025316 |
| 渺渺 | 何处 | 0.020408 |
| 渺渺 | 烟波 | 0.061224 |
| 芳草斜阳 | 杏花零 | 0.25 |
| 芳草斜阳 | 杏花 | 0.25 |
| 芳草斜阳 | 杏花零落 | 0.125 |
| 人间 | 天上 | 0.089936 |
| 天上 | 人间 | 0.204082 |
| start | 有佳人 | 0.000053 |
| start | 帘幕深深 | 0.000014 |
| start | 夜夜 | 0.000284 |

其中start表示宋词的开头。而“芳草斜阳”后连“杏花”，“杏花零”，和“杏花零落”的概率很有意思。“杏花”和“杏花零”概率相同表示其后只要出现“杏花”一定是“杏花零X”的形式。而除了“杏花零落”之外似乎还有别的可能。不过那些的出现频率较低，所以并没有在概率转移矩阵中出现。查找原始文本可以看到还可能出现的是“杏花零乱”。

**五、实现马尔科夫模型**

从start开始，每次根据上一个词选择概率最大的下一个词。在仅仅采用这个策略的时候，会这样:

“东风 不管 人间 天上 人间 天上 人间 天上 人间 天上”。

古人还是很喜欢人间天上的。

另一种做法是考虑词性之间的转换。可以简单地从词性中建立转移矩阵，并给每一个候选词安排词性。或者也可以利用宋词中每个小句长度较短的特性，直接枚举所有可能的句子结构。
在此选择仅考虑词之间的转换关系。

**六、处理词牌格式**

完整的词牌格式较为复杂，这里只考虑字数，平仄和押韵的要求。

清平乐:

平平仄仄， 仄仄平平仄（韵）。

仄仄平平平仄仄， 仄仄平平仄仄（韵）。

平平仄仄平平， 平平仄仄平平（韵）。

仄仄平平仄仄， 平平仄仄平平（韵）。

卜算子：

中仄仄平平，中仄平平仄（韵）。

中仄平平仄仄平，中仄平平仄（韵）。

中仄仄平平，中仄平平仄（韵）。

中仄平平仄仄平，中仄平平仄（韵）。

念奴娇：

中平中仄，仄平中中仄，中平平仄（韵）。

中仄中平平仄仄，中仄中平平仄（韵）。

中仄平平，中平中仄，中仄平平仄（韵）。

中平中仄，仄平平仄中仄（韵）。

中仄中仄平平，中平中仄，中仄平平仄（韵）。

中仄中平平仄仄，中仄中平平仄（韵）。

中仄平平，中平中仄，中仄平平仄（韵）。

中平平仄，仄平平仄平仄（韵）。

将其转换成易处理的格式，0表示中，1表示平，2表示仄，x表示韵。实际上是可能出现上下阙换韵的情况的，但是方便起见设置为全文同一个韵脚。

以下皆以清平乐作为词牌。

**七、使用马尔科夫模型生成符合字数要求的宋词**

采用最简单的递归填词方法，即从前往后依次填词，如果发现不能满足要求，便返回至上一层从新填。以下生成带平仄和带押韵的宋词都是使用这样的方法。

每个词都选择概率最大的搭配作为下一个词，这样的结果是：

芙蓉重阳，佳节是重阳。重阳佳节是重阳，重阳佳节重阳。

佳节重阳佳节，重阳佳节重阳。佳节重阳佳节，重阳佳节重阳。

发现已经出现了循环。这是因为仅仅选择出现概率最大的词这个策略太简单了，所以可以加上一定的随机性。即根据概率的大小，随机选择下一个词。

结果：

| |
|---|
| 鸳鸯不道，朱颜年年今。日长人静酒初醒，乐事真个情多。<br>无处安排旧日，章台天香院宇。东风海棠枝上，朦胧一夜相思。 |
| 明月清风，何处难忘酒。中秋明月人千里，思悠悠那堪更。<br>别离愁无风轻，袅袅天风露湿。浓香为寿一个，心肠章台何处。 |
| 黄昏人独，莫待西风吹。不断问何人从此，不须儿女此情。<br>烟水兰舟云中，独立西风斜照。江天秋色人家，人家怎生为主。 |

辞藻还是很华丽的。

### 八、使用马尔科夫模型生成符合平仄要求的宋词

现在的四个声调，前两声是平声，后两声是仄声。宋词格式很简单，每个词牌都是有固定的平仄要求的。所以只需要在生成词时判断该词是否满足对应位置的平仄要求即可。而已经处理得到的拼音数据库就可以派上用场了。

然而实际操作时发现了一个之前没有考虑到得问题：多音字。例如人间的间字，可以发第一声jian1，也可以发第四声jian4。在“人间”这个搭配中，它应该是发第一声的，所以放入宋词中应该处于平声的位置。而判断某字在某词中发什么音需要专门的词典。不过网上现有的一些多音字词典并不包括许多宋词中的常用词汇，所以实际可行性也不够大。于是这里选择了判断所有音调，即只要有一个音调符合该位置的平仄要求即认为该词符合要求。

加上了平仄要求之后，生成的词如下：

| |
|---|
| 黄花一醉，人间如何得。风又送春归院宇，燕子未归多少。<br>人家潇洒兰亭，修竹池上夕阳。时候人间都是，凄凉多少心期。 |
| 萧萧白发，不胜愁归去。人间人间尘不到，惟有落花流水。<br>小桥芳草相思，相思芳意不如。归去扁舟个中，消息好是秋风。 |
| 归来斜日，无语空凝伫。昨夜山阴何处是，天教收拾芳菲。<br>与谁芳意婆娑，临风柳絮帘栊。日日倚阑高处，凭阑有恨时时。 |

大约是因为宋词很多都走的婉约路线，即便是不定任何主题随意发挥，也总是透出一股凄凉的味道来。

### 九、使用马尔科夫模型生成符合押韵要求的宋词

押韵是词牌规则的最后一步。为了检查是否押韵，必须能够提取词的韵母。使用如下正则表达式：

```
pattern = r'^.*?(a|e|i|o|u|v|ai|ei|ui|ao|ou|iu|ie|ue|er|an|en|in|
un|vn|ang|eng|ing|ong)$'
ret = re.findall(pattern, pinyin)
```

```
return ret[0] if len(ret) != 0 else ''
```

在开始生成宋词之前，先随机一个韵，每次到需要押韵的位置时便检查是否符合即可。

生成结果：

| |
|---|
| 多情行乐，憔悴那堪更。一叶扁舟争知道，梦里长安谁问。<br>东风一夜朱颜，相逢笑语十分。和气自然天地，凭谁说与佳人。（韵：en） |
| 明年何处，人在斜阳里。几点碧天天似水，银屏此时此意。<br>谁知好事如今，凭栏无语当时。携手西园公子，登高一笑疏篱。（韵：i） |
| 少年双燕，楼外远山横。潇洒东篱人不到，老去不堪春梦。<br>尊前心事遥山，南楼只恐随风。今日今年自是，春风今日衰翁。 （韵：eng） |

至此，生成的宋词已经基本符合了所有格式的要求，并且辞藻还是很华丽的。然而内容还是十分空洞。所以可以考虑作一些围绕主题的宋词，类似于咏物词这样的。

**十、提取主题词**

这里考虑在给定一个主题的情况下，如何找到和主题相关的词。而且，日常生活经验告诉我们，词和主题的关联程度是由区别的。例如对于主题“月”，那么“新月”，“残月”，“婵娟”等等是最相关的词语，而“皎洁”，“明亮”等等应该是次一级的相关。为了得到这样的规则，我们可以选择手动给出主题词，或者通过聚类等方法将词语归类，并根据词之间的距离划分等级。我们选择的方式是这样的：

1.给出主题

2.将包含了主题的宋词中出现的所有候选词去除，并建立频率表

3.根据频率排名，选择主题词和决定相关程度。

4.在概率转移矩阵中扩大转移到主题词的概率。

例如，在主题为“月”的情况下，其频率最高的十个词如下：

| |
|---|
| 明月 |
| 风月 |
| 何处 |
| 东风 |
| 月明 |
| 人间 |
| 梅花 |
| 黄昏 |
| 春风 |
| 相思 |

其中很多都是与月有关的词汇。然而也有一些是本来频率就很高，所以才会出现在这个表格里的词，他们本身和月的关系并不是很大。例如人间，东风，何处等等。所以在取主题词的时候需要去除这些词。

去除之后的十个频率最高的为：

| |
|---|
| 月明 |
| 月下 |
| 新月 |
| 三月 |
| 月华 |
| 夜月 |
| 月上 |
| 风露 |
| 清风 |
| 梨花 |

这些都的确和月有很大的关系。

**十一、使用马尔科夫模型生成围绕主题词的宋词**

获得了主题词之后，简单地将这些主题词的概率按照一定的比率增大即可。但是这个增大的比率也是需要调节的。

我们采用了非常简单的分级策略：频率最高的前三十个词为一级主题词，随后三十个为二级主题词，再后面三十个为三级主题词。

| 一级 | 二级 | 三级 |
| --- | --- | --- |
| 月明 | 花影 | 携手 |
| 月下 | 二月 | 明朝 |
| 新月 | 秋风 | 容易 |
| 三月 | 相见 | 往事 |
| 月华 | 花枝 | 今年 |
| 夜月 | 花月 | 珠帘 |
| 月上 | 一夜 | 月淡 |
| 风露 | 行云 | 月照 |
| 清风 | 分明 | 不禁 |
| 梨花 | 小楼 | 月如 |
| 疏影 | 精神 | 相对 |
| 酒醒 | 可怜 | 梦断 |
| 今宵 | 月中 | 花飞 |
| 夜深 | 不堪 | 娟娟 |
| 中秋 | 小窗 | 从今 |
| 玉人 | 清夜 | 独自 |
| 几时 | 分付 | 花前 |
| 日月 | 又是 | 离别 |
| 思量 | 风光 | 楼台 |
| 月满 | 西楼 | 记得 |
| 暗香 | 残月 | 风前 |
| 一曲 | 明日 | 枝上 |
| 只有 | 有人 | 月影 |
| 淡月 | 夜寒 | 皓月 |
| 夜来 | 倚阑 | 碧云 |
| 笙歌 | 朦胧 | 何时 |
| 庭院 | 梦回 | 人去 |
| 人在 | 旧时 | 何人 |
| 落花 | 青山 | 楼上 |
| 惟有 | 婵娟 | 月冷 |

如果将一级词提升10倍，二级词5倍，三级词2倍，会生成如下的宋词：

| |
|---|
| 当时阑干，花下风吹醒。月下有人当此际，浑似当年风景。<br>可怜红粉婵娟，秋风千里功名。似我一生怀抱，不须年少心情。 |
| 绿杨依旧，似当时风月。相对年年长相见，杨柳归来忘却。<br>当时人在春风，不知春色依约。见说今年芳草，芙蓉照水清绝。 |
| 如今憔悴，无限凭谁说。庭院日迟迟何处，因甚江头烟水。<br>连天何处一片，古今多少为谁。醉倒今宵不忍，思量因甚不归。 |

第一篇感觉还挺好的，又有月下，又有婵娟。而第二和第三篇就不关月亮什么事情了。

那么将倍数增加，提升到一级词50倍，二级词30倍，三级词10倍看会怎么样。

| |
|---|
| 婵娟依旧，留住三更月。不见故人谁念我，重见依约见说。<br>小楼何处花阴，满城春色依约。见说新愁一笑，倾城一笑不觉。 |
| 旧时携手，肠断不知何。处是蓬莱何处是，只有江南红叶。<br>可怜时候谁知，月明枝上留得。老子今朝不管，东风一笑来何。 |
| 清风一笑，白发还自笑。应是江山知何处，客里不知春到。<br>家山好在夕阳，留春尽道今宵。一醉劝君看取，和羹风度蟠桃。 |

看起来也并没有更多的出现月亮。实际上只是顺序扫一遍的话，随机性还是很大的。

**十二、实现遗传算法**

考虑到顺序生成的随机性很大，并且一整个宋词由多个词片段组成，这种形式非常适合使用遗传算法。在计算概率矩阵，填词，平仄，押韵等部分都已经完成了的情况下，实现一个遗传算法也就不是那么困难了。

算法流程如下：

1.确定n条初始基因

2.计算各条基因的适应性函数

3.将已有基因作为父代基因，加入基因池。适应性函数值高的基因占有更多的比率

4.循环n/2次，每次选择基因池中的两个基因，并将其交叉，得到两个子代基因或者随机变异得到两个子代基因。由此得到n条下一代基因

5.进行若干代后，取出适应性函数值最高的作为输出结果。

其中初始基因可以通过原有的马尔科夫模型生成，而适应性函数简单地设为这个序列的概率。如果初始基因都有同样的韵脚，那么只需要在变异的时候考虑好押韵和平仄的规则就可以了。

在生成初始基因的时候，某些韵脚可能会较难生成。所以在连续10次生成失败之后会换韵重新生成。

生成结果如下（主题依然是月）：

| |
|---|
| 婵娟应念，老子今日是。明日花前携手处，月满中秋次第。<br>笙歌庭院深深，梅花今日不知。何处谪仙醉倒，花前月下花枝。 |
| 有人间世，不知人易老。惟有旧时携手处，月满西楼春到。<br>江南千里婵娟，又何如玉今宵。酒醒断肠人在，玉堂元是今朝。 |
| 玉人天上，人间如何得。君不奈情何好是，清夜一尊和泪。<br>倚阑人在蓬壶，寻思只有双飞。飞上蓬莱三岛，风烟人间安得。 |

**十三 生成其他词牌的宋词**

卜算子：

何处是江南，两岸花飞絮。满长安千古行人，十里香芬馥。

桃李又一番，春暮知何处。燕子归时候扁舟，桃李纶巾羽。

念奴娇：

梨花庭院，醉蓬莱日月，已成千岁。

多少一枝何处是，还是杏花休说。

今夜分明，冰壶影里，岁岁花前醉。

而今总是，当时风月富贵。

如许阑干莺声，江南憔悴，金缕衣憔悴。

楼上一声天似水，昨夜匆匆一醉。

未必江山，暮天风景，此意凭谁说。

阑干芳草，莫惜今夜同醉。

## 【总结】

仅仅从词汇堆砌的角度上来看，计算机能够做到比较好的效果。这个程序使用马尔科夫模型生成一首宋词只需要不到一秒。而使用遗传算法虽然时间会增长一些，大部分情况下也不需要一分钟（生成初始基因时失败，然后换韵占据了大部分计算时间）。但是想要得到真正句意通顺，有内涵的诗句恐怕就是一个庞大的课题了。将这种生成手法当做一种作词的辅助，或许会有一些效果。

## 【附录】

一 、程序使用方法

参见　try.py。

python版本为 2.7

NLTK版本为3.0

二 、部分主要程序注解

| | |
|---|---|
| main.py | 主要运行步骤 |
| process_data.py | 处理原始数据 |
| format.py | 生成过程的大部分核心内容 |
| gene.py | 遗传算法 |
| get_theme_pair.py | 处理主题词 |
| try.py | 样例 |

---

format.py:

```
# -*- coding: utf-8 -*-
from gene import *
import re
import codecs
import pdb
import random


def get_pairs():
    infile = codecs.open('pairs.txt', 'r', 'utf-8')
    ret = []
    for l in infile:
        ret.append(l[:-1])
    return ret


class MyFormat:
    def __init__(self):
        # 词牌的格式，每小句的平仄要求
        self.sen_form = []
        # 存储需要押韵的位置
        self.yun = []
        # 表示需要押什么韵,每次随机
        self.yayun = ''
        # 存储生成的结果
        self.sents = []
        # 存储每个词后面可能搭配的词及其概率
        self.cfd = dict()
```

```
        # pinyin : 字->拼音,声调
        self.pinyin = dict()
        # 所有韵母
        self.vowel = ['a', 'e', 'i', 'o', 'u', 'v', 'ai', 'ei', 'ui',
'ao', 'ou', 'iu',
                      'ie', 'ue', 'er', 'an', 'en', 'in', 'un', 'vn',
'ang', 'eng', 'ing', 'ong']
        # 存储词对的概率
        self.cfd_dict = dict()

    def load_format(self, filename):
        '''
        读取格式
        :param filename:  存储格式的文件, 文件格式为一行，0表中，1表平，2表仄，x
表韵
        '''
        f = codecs.open(filename, 'r', 'utf-8')
        a = []
        k = 0
        for l in f:
            for i in range(len(l)):
                if l[i] == '0' or l[i] == '1' or l[i] == '2':
                    # 0, 1, 2代表这是字
                    a.append(l[i])
                elif l[i] == 'x':
                    # 需要押韵，加入押韵表
                    self.yun.append(k)
                elif len(a) > 0:
                    # 否则是句子的分隔
                    self.sen_form.append(a)
                    self.sents.append([])
                    a = []
                    k += 1

    def checkpinyin(self, s, p):
        '''
        检查平仄规则
        :param s: 词
        :param p: 该词位置需要的平仄
        :return: 是否符合平仄规则
        '''
        for i in range(len(s)):
            flag = False
            if not s[i] in self.pinyin:
                return False
            for item in self.pinyin[s[i]]:
                if int(p[i]) == 0 or (item[1] + 1) / 2 == int(p[i]):
                    flag = True
            if not flag:
                return False
        return True

    def check_yayun(self, item, k, p):
        '''
        检查是否押韵
```

```
        :param item: 词
        :param k: 句号
        :param p: 词在句子中的位置
        :return: 是否符合押韵规则
        '''
        if k in self.yun and len(item) + p == len(self.sen_form[k]):
            flag = False
            for a in self.pinyin[item[-1]]:
                if a[0] == self.yayun:
                    flag = True
                    break
            idx = self.yun.index(k)
            for i in range(idx):
                if item[-1] == self.sents[self.yun[idx - 1]][-1][-1]:
                    flag = False
                    break
            return flag
        else:
            return True

    def fit_func(self, x):
        '''
        提供给遗传算法的适应性函数，计算序列概率*1000
        :param x: 词序列
        :return: 词序列的概率*1000
        '''
        tmp = ('start', x[0])
        f = (self.cfd_dict[tmp] if tmp in self.cfd_dict else 0.000001) *
1000
        for i in range(len(x) - 1):
            tmp = (x[i], x[i + 1])
            f *= (self.cfd_dict[tmp] if tmp in self.cfd_dict else
0.000001) * 1000
        return f

    def gen(self, words, k, p):
        '''
        马尔科夫模型生成的主要部分
        :param words: 上一个词，初始为start
        :param k: 第k句
        :param p: 句中第p个字
        :return: 是否生成成功
        '''
        if k >= len(self.sen_form):
            return True
        if words not in self.cfd:
            return False
        g = 1
        for item in self.cfd[words]:
            u = item[1]
            item = item[0]
            # 按照其概率选择
            g = random.uniform(0, 1) / g
            if g > float(u):
                continue
```

```
            if len(item) + p <= len(self.sen_form[k]):
                # 检查平仄
                if not self.checkpinyin(item, self.sen_form[k][p:p
+len(item)]):
                    continue
                # 检查押韵
                if not self.check_yayun(item, k, p):
                    continue
                # 加入结果
                self.sents[k].append(item)
                q = len(item) + p
                j = k
                if q >= len(self.sen_form[k]):
                    q = 0
                    j += 1
                # 处理下一个词
                if self.gen(item, j, q):
                    # 生成成功，返回
                    return True
                # 生成失败，换词
                self.sents[k].pop()
        return False

    def gen_genetic(self):
        '''
        遗传算法生成宋词
        '''
        # 读入初始数据
        self.getcfd()
        self.getpinyin()
        words = 'start'
        # 随机韵脚
        self.yayun = self.vowel[random.randint(0, len(self.vowel) - 1)]
        initdata = []
        number = 50
        failnumber = 0
        # 随机生成50个初始宋词
        for i in range(number):
            while True:
                self.clear_sents()
                if self.gen(words, 0, 0):
                    # 生成成功
                    print 'init %d' % i
                    tmp = []
                    for a in self.sents:
                        for b in a:
                            tmp.append(b)
                    initdata.append(tmp)
                    failnumber = 0
                    break
                else:
                    # 生成失败
                    failnumber += 1
                    if failnumber > 10:
                        # 连续失败十次，换韵
```

```
                        self.yayun = self.vowel[random.randint(0,
len(self.vowel) - 1)]
                        i = 0
                        initdata = []
                        failnumber = 0
                    print 'fail'

        print 'initial data ready'


        # 调用遗传算法
        g = Gene(get_pairs(), number, initdata)
        time = 3000
        g.geneNum(time, self.fit_func, 0.1, self.check_pinyin_and_yun)

        # 取出生成结果
        ret = g.get_max()
        k = 0
        p = 0

        self.clear_sents()
        for item in ret:
            p += len(item)
            self.sents[k].append(item)
            if p >= len(self.sen_form[k]):
                p = 0
                k += 1

    def check_pinyin_and_yun(self, s, p):
        '''
        检查词是否符合平仄和押韵要求,在遗传算法的变异中使用
        :param s: 字
        :param p: 位置
        :return: 是否符合要求
        '''
        # 计算出在哪一句
        k = 0
        while p >= len(self.sen_form[k]):
            p -= len(self.sen_form[k])
            k += 1

        # 做检查
        if not self.checkpinyin(s, self.sen_form[k][p:p+len(s)]):
            return False
        if not self.check_yayun(s, k, p):
            return False
        return True

    def clear_sents(self):
        '''
        清空结果
        '''
        for i in range(len(self.sents)):
            self.sents[i] = []
```

```
    def gen_maximum_prob(self):
        '''
        使用马尔科夫模型生成宋词
        '''
        # 读入初始数据
        self.clear_sents()
        words = 'start'
        self.getcfd()
        self.getpinyin()


        while True:
            # 随机韵脚,并生成
            self.yayun = self.vowel[random.randint(0, len(self.vowel) -
1)]
            if self.gen(words, 0, 0):
                break

    def output(self):
        '''
        输出结果
        '''
        for item in self.sents:
            print ''.join(item)



    def getcfd(self):
        '''
        读入概率转移矩阵
        :return:
        '''
        # 如果要主题就选择bias_prob.txt
        probfile = codecs.open('bias_prob.txt', 'r', 'utf-8')
        # probfile = codecs.open('prob.txt', 'r', 'utf-8')
        for l in probfile:
            l = l.split(' ')
            if not l[0] in self.cfd:
                self.cfd[l[0]] = []
            self.cfd[l[0]].append((l[1], float(l[2])))
            self.cfd_dict[(l[0], l[1])] = float(l[2])

        for keys in self.cfd.keys():
            self.cfd[keys].sort(key=lambda x: x[1], reverse=True)


    def getpinyin(self):
        '''
        读入拼音字典
        '''
        infile = codecs.open('data/pinyin.txt', 'r', 'utf-8')
        for l in infile:
            l = l.split(' ')
            if l[0] not in self.pinyin:
```

```
                self.pinyin[l[0]] = []
            # 提取韵母
            vowel = self.get_vowel(l[1][:-2])
            # 将an，ang、in，ing、en，eng合并
            if vowel.endswith('g'):
                vowel = vowel[:-1]
            self.pinyin[l[0]].append((vowel, int(l[1][-2])))


    def get_vowel(self, pinyin):
        '''
        得到拼音的韵母
        :param pinyin: 拼音
        :return: 其韵母部分
        '''
        pattern = r'^.*?(a|e|i|o|u|v|ai|ei|ui|ao|ou|iu|ie|ue|er|an|en|
in|un|vn|ang|eng|ing|ong)$'
        ret = re.findall(pattern, pinyin)
        return ret[0] if len(ret) != 0 else ''
```

---

gene.py

```
# -*- coding: utf-8 -*-
import random


class Gene:
    def __init__(self, gene_data, num, initdata):
        # 变异时的候选词库
        self.gene_data = gene_data
        # 当前代的基因
        self.data = initdata
        # 基因个数
        self.num = num
        # 基因池
        self.pool = []
        # 适应性函数
        self.func = 0
        # 检查押韵和平仄的函数
        self.check = 0

    def fitness_fun(self, f):
        '''
        计算适应性函数，并更新基因池
        :param f: 适应性函数
        '''
        total = 0
        score = []
        # 计算各个基因的适应性函数
        for i in self.data:
```

```
            x = f(i)
            score.append(x)
            total += x

        # 根据概率大小加入基因池
        for idx, i in enumerate(self.data):
            for j in range(int(score[idx] / total * self.num * 100)):
                self.pool.append(i)

    def select(self):
        '''
        :return: 从基因池中选择一个基因
        '''
        i = random.randint(0, len(self.pool)-1)
        return self.pool[i]

    def generate(self):
        '''
        交叉
        '''
        x = self.select()
        y = self.select()
        t = random.random()
        if t < 0.7:
            while True:
                pos = random.randint(0, len(x) - 1)
                # 检查长度是否符合
                if len(''.join(x[:pos])) == len(''.join(y[:pos])):
                    break
            m = x[:pos] + y[pos:]
            n = y[:pos] + x[pos:]
            x = m
            y = n
        # 添加进当前代基因
        self.data.append(x)
        self.data.append(y)

    def vari(self):
        '''
        变异
        '''
        x = self.select()
        n = random.randint(0, len(x) - 1)
        while True:
            m = random.randint(0, len(self.gene_data) - 1)
            if len(self.gene_data[m]) != len(x[n]):
                continue
            # 检查符合词牌要求
            if self.check(self.gene_data[m], len(''.join(x[:n]))):
                break
        x[n] = self.gene_data[m]
        self.data.append(x)

    def geneNum(self, time, f, bian, check):
        '''
```

```
        执行遗传算法的主要部分
        :param time: 代数
        :param f: 适应性函数
        :param bian: 变异系数
        :param check: 检查函数
        '''
        self.func = f
        self.check = check
        for i in range(time):
            print 'Epoch %d' % i
            self.pool = []
            self.fitness_fun(f)
            self.data = []
            for j in range(self.num / 2):
                r = random.random()
                if r > bian:
                    self.generate()
                else:
                    self.vari()
                    self.vari()


    def get_max(self):
        '''
        return 取出适应性函数最大的作为生成结果
        '''
        ret = 0
        maxi = -1
        for i in self.data:
            x = self.func(i)
            if x > maxi:
                ret = i
                maxi = x
        return ret
```