**基于HMM的词性标注器 设计文档**

1. **任务定义**

对一个已分词、已做标注的中文文本进行基于HMM模型的词性标注模型的构建。本实验中，给定输入为已做分词、已做标注的中文文本（人民日报标注语料），将输入集通过数据清洗，划分为训练集和测试集，训练集用于训练HMM模型参数，而测试集用于重新标注词性，并与基准的测试集进行对比。输出为词性标注之后的测试集以及标注结果的评价。

1. **源码运行环境**

采用python2.7作为编程语言，编程环境为win10操作系统下的wing IDE 5.1。脚本为test\_seg.py，输入文件为renminribao.txt，输出为tagged.txt，均采用gbk编码。

1. **输入输出**

**程序输入：**

给定输入为作业包中附带的人民日报语料（reminribao.txt），其语料特点为：

1. 已分词，且所有的符号（标点、限界符等）均算作一个词。
2. 已标注，标注格式均为“/\w+”（利用python的正则表达式来表示）。
3. 有空行，在windows下，空行被表示为\r\n。
4. 有字音标注，如多音字结{jie1}。
5. 有分界符，以标注专有名词，如[亚太经合/j 组织/n]nt。
6. 每一行都以日期开始。

将数据清洗过的前18000行作为训练集，来对18001-20000行的数据进行词性去除和再标注。

**程序输出：**

输出一个根据HMM模型重新标注的文件（tagged.txt），内容是对输入的测试集去除词性标记后再标记的结果，输出标注的准确率，和与一个平凡标注器的准确率的对比。

1. **方法描述**
2. **实验框架**

本实验的总体框架用流程图表示如下：

测试数据

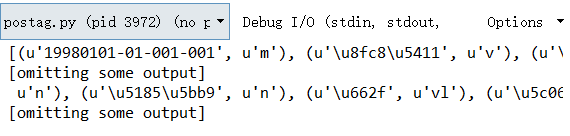
训练数据

去除标注

1. **数据处理**

由于输入文本是已分词，已标注的文本，我们首先要做的就是去除一些对于本次标注任务会产生干扰的信息。比如，去除命名实体标志符号，将命名实体打散为若干个普通词的集合。再者，需要剔除多音字的标识，以上两种信息对于词性标注是会产生干扰的。

将训练集以序列的形式存储，序列的元素是一个二元组，即一个词和它的标注共同组成的元组，这位我们以后的操作提供了很大的方便。



此后，为了计算状态转移矩阵A和观测矩阵B，我们需要如下的一些过程作为前期数据准备：

1. **标注频次字典**

即在训练集中，每一个词性标注都出现了多少次。由于标注种类并不多，因此我们希望在构建HMM之前就准备好数据，需要的时候直接查阅字典即可，而不需要现场计算，这样可以简化模型并提高标注效率。

1. **bigram标注频次字典**

由于在计算状态转移矩阵时利用到了MLE估计，我们需要统计以标注组成的bigram的频次字典。

1. **标注类型宽度**

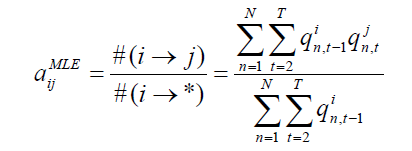
即一共有多少种标注，很容易得到。

1. **观测频次字典**

即一个标注对应一个观测的频次字典，若以矩阵表示则过于稀疏，为了以后运算方便我们还是提前构造出这个字典。

1. **构造状态转移矩阵Aij和观测矩阵Bjk**
2. 状态转移矩阵Aij

采用MLE估计来计算Aij的值。



注意到凡是应用到了MLE估计，则就有引入数据稀疏的可能,数据稀疏在HMM中会从一点向后传播，导致之后的一系列概率均为0，如果PI分布稀疏，那么在一开始的情况下就会引入0概率并向后蔓延至所有以它为起点的状态序列，无论之后的状态序列多么合理，其累计概率都将表现为0。而矩阵A和矩阵B的稀疏问题则体现在计算Viterbi的v时某一项会变为0，导致之后所有以v点出发的状态序列的概率均为0。

在状态转移矩阵A中，数据稀疏表现在不存在这样的转移，即从某个状态转移到另一个状态。因此我们有必要引入平滑方法，在这里我们采用delta=0.5的delta平滑，实验表示效果较好。

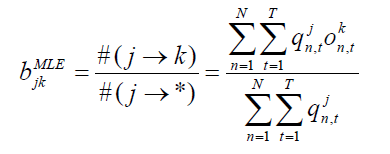
通过过程

*def getAij(tag\_width,tag\_i,tag\_j,tag\_freq\_dict,tag\_bigram\_freq\_dict,delta = 0.5):*

来在标注器工作时得到某个Aij的值，这里仍然是用python字典存储的。显然我们构造一个A矩阵需要如下信息：标注宽度，标注二元组（i和j），标注频次字典，二元标注频次字典和平滑参数。

1. 观测矩阵Bjk

同理采用MLE估计来计算Bjk的值。



同A矩阵一样，观测矩阵B同样有数据稀疏的问题，且数据稀疏问题更加严重。这是因为B矩阵本身就十分稀疏，如果出现一个新词，或者某个标注从未发射到某个观测上，就会引起数据稀疏。我们同样采用delta平滑。

此外，观测矩阵的宽度为词表长度乘以标注宽度，这是一个相当大的矩阵，如果事先构造好而不存储它，那么这样的构造是费时费力的，因此在本程序中，我们采用“现用现查”的模式来构造一个虚拟的B矩阵，而非在训练集输出之前就得到所有的B矩阵的值。

通过过程*def getBjk(tag\_width,tag\_j,obs\_k,tag\_freq\_dict,tag\_obs\_freq\_dict,delta = 0.01):*

来得到某个Bjk的值，与状态转移矩阵A类似，我们需要传入j和k的信息，以及构造这个矩阵的必要信息，来实现“现查现用”的目的。

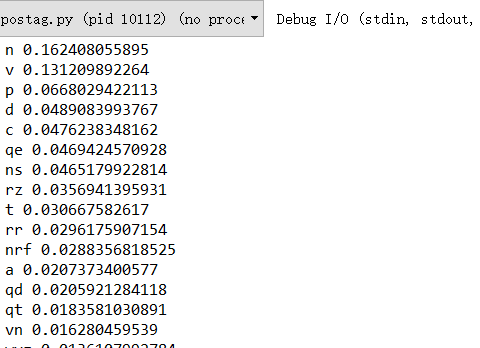
1. **计算初始概率分布PI**

这是一个简单的过程，目的是求得每一个标注的初始分布。如何从训练集中判断一个标注作为了一个句子的开始呢？观察训练集可以看到，所有的标点符号的标注均为以w开头的标注，我们利用这个特征可以得到每一个标注作为起始标注的频次。值得注意的是，由于训练集本身结构的原因，每一行开始都是一个日期，而日期之后的词的标注同样是一个句子的开始，因此我们用这样的逻辑来判断一个标注是否为起始标注：

*if ( tag\_i[0] == u'w' or tag\_i == u'm' ) and tag\_j != u'm'*

但我们不应该把所有的标注m（日期）作为起始标注，即使我们预先知道测试集所有的行都是以日期开始的。这是由于我们希望得到一个通用性较好的HMM标注器，而非仅仅针对于训练集这样的数据结构，因此我们忽略所有的m作为句子开始标注的事实。

我们打印PI分布并排序（部分），有如下结果。



左侧为标注，右侧为标注的初始概率分布，可以看出名词作为句子开始的情况最多，动词第二，介词第三，这与我们的直觉是相符的，而且我们派出了日期标注m的干扰。

1. **Viterbi算法与HMM标注器**

再有了训练参数后（解决了HMM的第三个问题），我们可以开始着手构建解决HMM第二个问题——解码问题的Viterbi算法。本质上是，给出一个观测序列（测试集上的若干句子），给出这个观测序列对应的最可能的状态序列（标注序列）。其实这是一个有限制的最短路径（Kruskal）算法，其限制为路径的方向只能由t到t+1，算法是在求解一个从t=0开始到t=T结束的最短路径，只不过这个“短”是由概率权重表征的。而HMM在求得最短的基础上加入了观测序列，但由状态发射到观测的权重不依赖于任何别的状态，这些限制因素都是基于HMM的独立性假设而来的。

在python中实现Viterbi算法也十分简洁，伪代码如下：

*def Viterbi（长度为T的观测序列，状态宽度为N，A，B，PI）：*

*初始化路径矩阵，这个矩阵是（N+2）\*T面积的*

*对于每一个tag，T=0时做初始化，利用PI矩阵求v[tag,0]*

*初始化back[tag,0]为一个结束标记*

*for i 从 1 到 T：*

*for 每个状态n：*

*v[n,i] = max{v[pre\_n，i-1]\*a[pre\_n,n]\*b[n,word[i]]}*

*back[n,i] = 取到max值的那个pre\_n*

*return v,back*

在程序中忽略了句子结尾的标注的概率，即结束分布，而是采用一个完整句子的最后一个标注就作为了viterbi路径的终点。

为了得到最优路径，我们找到最大的v[tag,end\_idx]作为回溯的起点，依次向前递推，就能还原出标注序列。

值得注意的是，计算v的过程是一个概率累乘的过程，加之标注规模较大且B矩阵的值十分小，造成若干次乘积过后，v的值会下溢，在程序中我们采用取对数相加的方式来消除因下溢而带来的影响。

在得到了测试集的标注序列后，标注任务也随之完成，便可以统计评价指标和输出标注后的测试集于文件中。

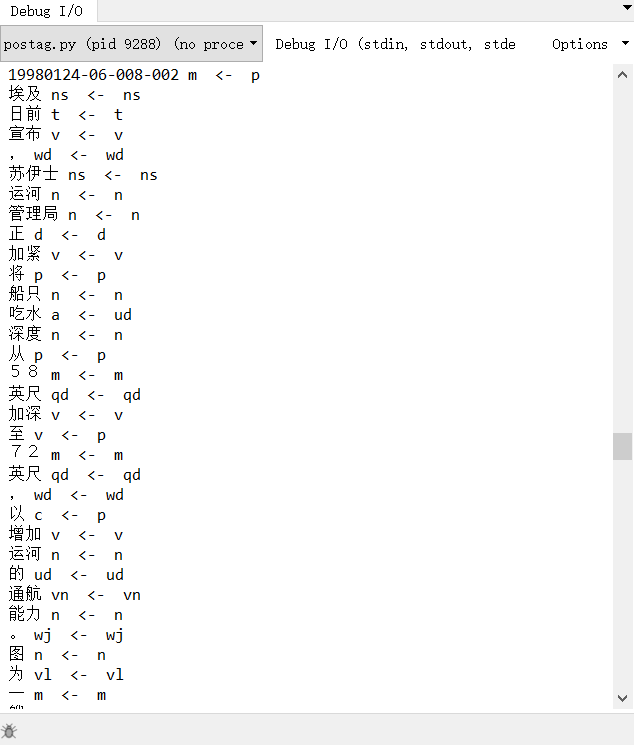
1. **结果分析及性能评价**

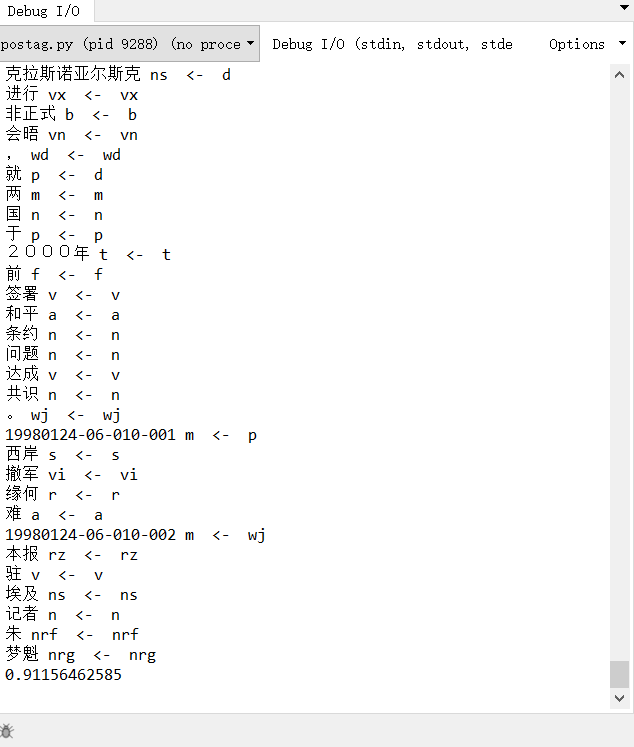
由于测试集和标注后的测试集的标注是一一对应的，因此我们直接统计正确率即可。由于标注过程不算快（本程序约1秒钟完成标注40个词，即viterbi中前进40个观测），先采用小规模数据进行效果演示。

用前18000行数据作为训练，测试集为10行，并输出标注结果和准确率（precision）。

调用train\_list,test\_list = getLinesFromFile('renminribao.txt',0,18000,18010)创建数据集。

调用HMMTagger过程完成标注，prec过程可以输出正确率，得到结果（部分）如下：

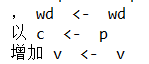




输出中左列是测试集的观测，中列是测试集本身正确的词性标注，右列是HMM输出的词性标注序列。

通过部分结果不难看出，大部分的标注均可以达到要求，少部分专有名词，命名实体等标注结果不佳。在对10行句子的标注中，准确率可以达到91%。

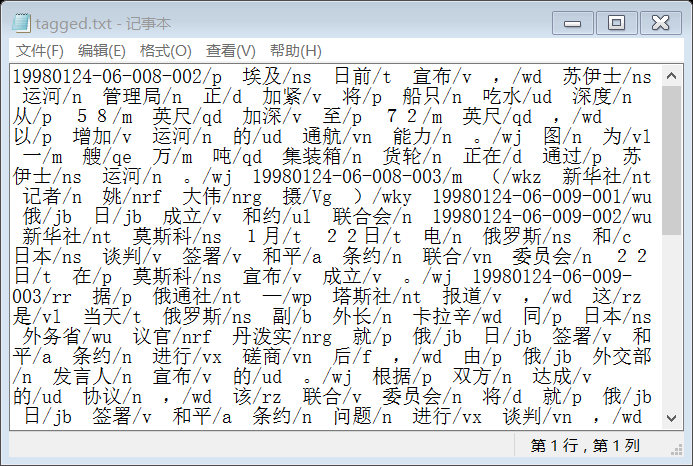
**而错误主要集中在**

****

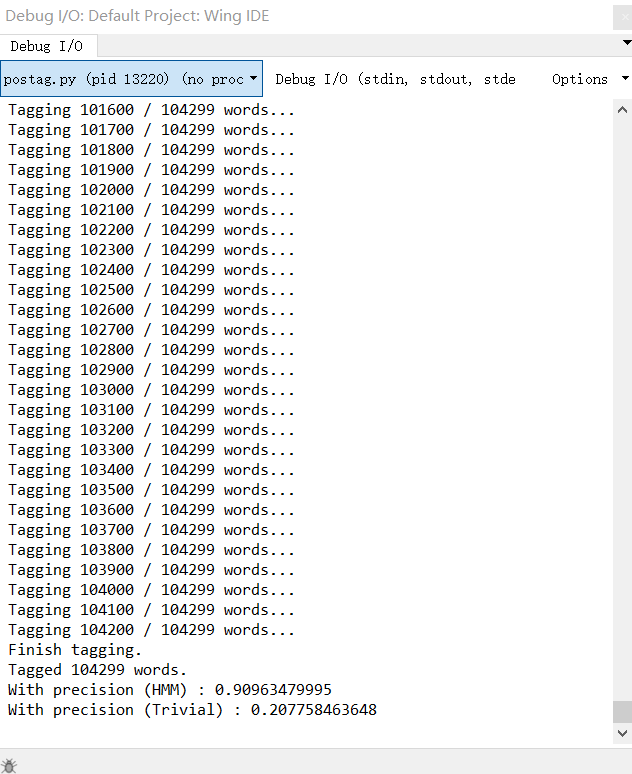
这表明，HMM由于状态转移矩阵只与前一个状态有关，而前一个状态是标点符号的时候，此时宏观上来看，标点符号之后可能出现的标注最多（作为一个句子的开始），也就是说对标点符号之后可能的输出不确定性最大，因此在标注为w类（标点）之后的标注错误率较高。

我们注意到很少有日期正确标注为m的（包括测试集开端），这是因为我们在PI中消除了这种约定数据结构的影响，HMM认为我们输入的是一个正常的句子集，这个句子集很小的概率会以日期开始。

最后再以文件形式输出，即完成了本次的词性标注任务。



我们再采用90%训练集，10%测试集的模式，即调用train\_list,test\_list = getLinesFromFile('renminribao.txt',0,18000,20000)。由于输出过于庞大，**标注结果请见附件tagged.txt，在这里只输出标注的准确率**。同时我们再构造一个trivialTagger，这个标注器选取tags中最可能出现tag标注所有的词汇。



本次标注过程约1.5小时，这说明本标注器效率还不够高，但是准确率已经相比于平凡标注器有了质的改善（约91%）。全部的标注结果请见tagged.txt。