

基于 HMM 的词性标注器 设计文档

一、 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。

二、 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。

三、 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。

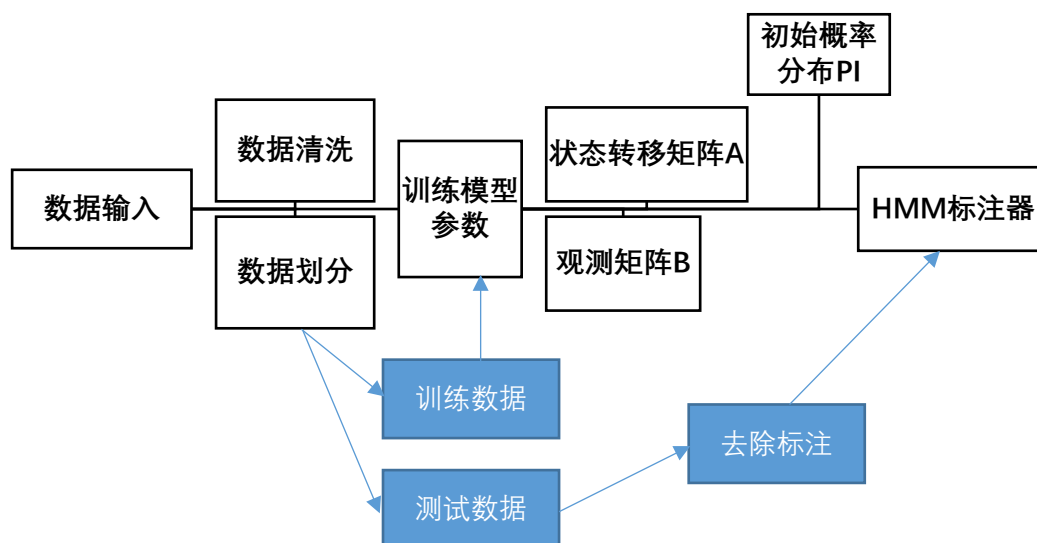
程序输出：

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。

四、 方法描述

1、实验框架

本实验的总体框架用流程图表示如下：



2、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，这为我们以后的操作提供了很大的方便。

```

postag.py (pid 3972) (no p Debug I/O (stdin, stdout, Options
[(u'19980101-01-001-001', u'm'), (u'\u8fc8\u5411', u'v'), (u'
[omitting some output]
u'n'), (u'\u5185\u5bb9', u'n'), (u'\u662f', u'v1'), (u'\u5c06
[omitting some output]

```

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。

2) 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。

3) 标注类型宽度

即一共有多少种标注，很容易得到。

4) 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。

3、构造状态转移矩阵 Aij 和观测矩阵 Bjk

1) 状态转移矩阵 Aij

采用 MLE 估计来计算 Aij 的值。

$$a_{ij}^{MLE} = \frac{\#(i \rightarrow j)}{\#(i \rightarrow *)} = \frac{\sum_{n=1}^N \sum_{t=2}^T q_{n,t-1}^i q_{n,t}^j}{\sum_{n=1}^N \sum_{t=2}^T q_{n,t-1}^i}$$

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 概率并向前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。

2) 观测矩阵 Bjk

同理采用 MLE 估计来计算 Bjk 的值。

$$b_{jk}^{MLE} = \frac{\#(j \rightarrow k)}{\#(j \rightarrow *)} = \frac{\sum_{n=1}^N \sum_{t=1}^T q_{n,t}^j o_{n,t}^k}{\sum_{n=1}^N \sum_{t=1}^T q_{n,t}^j}$$

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。

4、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。

```
postag.py (pid 10112) (no proc... Debug I/O (stdin, stdout,
n 0.162408055895
v 0.131209892264
p 0.066802942213
d 0.0489083993767
c 0.0476238348162
qe 0.0469424570928
ns 0.0465179922814
rz 0.0356941395931
t 0.030667582617
rr 0.0296175907154
nrf 0.0288356818525
a 0.0207373400577
qd 0.0205921284118
qt 0.0183581030891
vn 0.016280459539
... 0.0136407003704
```

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。

5、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:
```

```

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

$$back[n,i] = \text{取到 } \max \text{ 值的那个 } pre\_n$$

$$\text{return } v, back$$

```

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 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。

五、 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：

Debug I/O

postag.py (pid 9288) (no proce

Debug I/O (stdin, stdout, stde

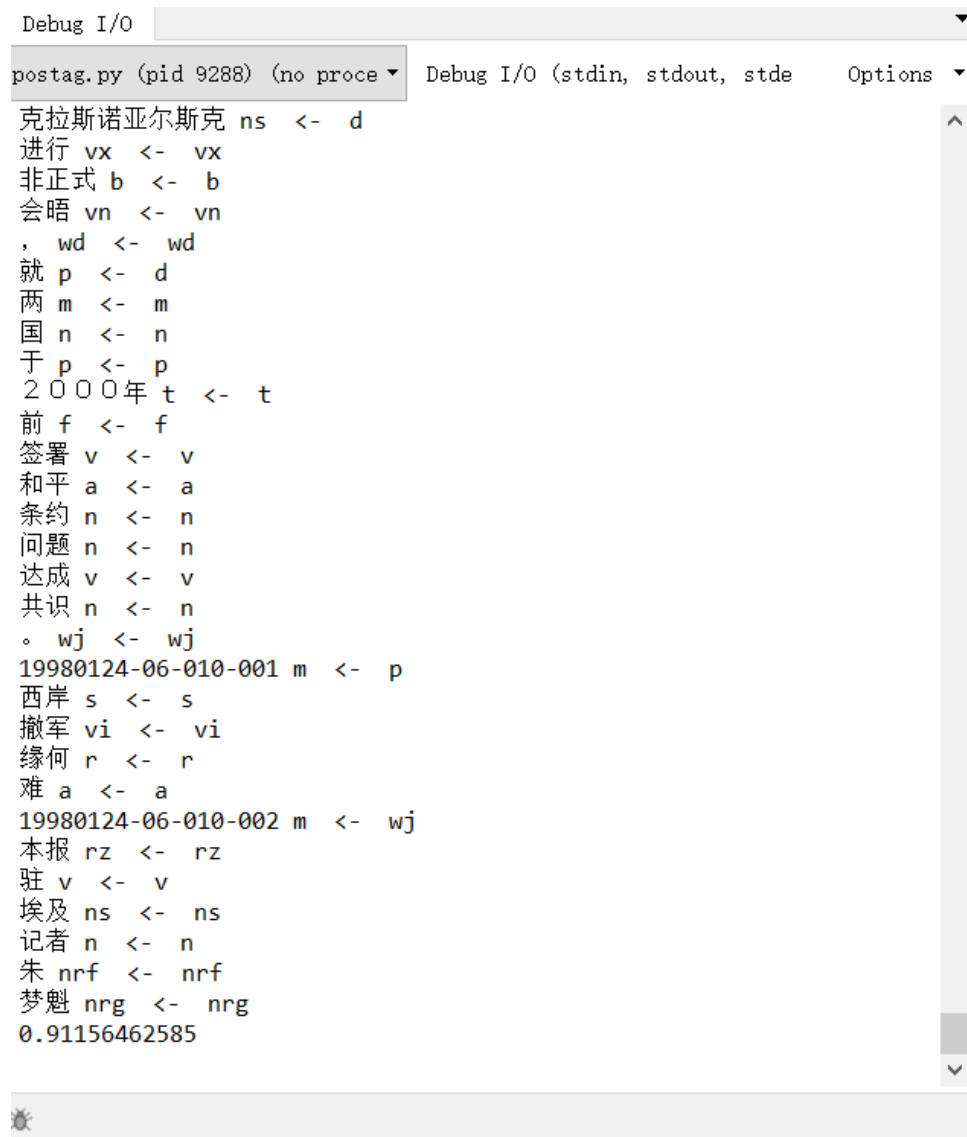
Options

```

19980124-06-008-002 m <- p
埃及 ns <- ns
日前 t <- t
宣布 v <- v
, wd <- wd
苏伊士 ns <- ns
运河 n <- n
管理局 n <- n
正 d <- d
加紧 v <- v
将 p <- p
船只 n <- n
吃水 a <- ud
深度 n <- n
从 p <- p
5 8 m <- m
英尺 qd <- qd
加深 v <- v
至 v <- p
7 2 m <- m
英尺 qd <- qd
, wd <- wd
以 c <- p
增加 v <- v
运河 n <- n
的 ud <- ud
通航 vn <- vn
能力 n <- n
。 wj <- wj
图 n <- n
为 vl <- vl
一 m <- m

```





```

Debug I/O
postag.py (pid 9288) (no proce
Debug I/O (stdin, stdout, stde
Options
克拉斯诺亚尔斯克 ns <- d
进行 vx <- vx
非正式 b <- b
会晤 vn <- vn
, wd <- wd
就 p <- d
两 m <- m
国 n <- n
于 p <- p
2000年 t <- t
前 f <- f
签署 v <- v
和平 a <- a
条约 n <- n
问题 n <- n
达成 v <- v
共识 n <- n
。 wj <- wj
19980124-06-010-001 m <- p
西岸 s <- s
撤军 vi <- vi
缘何 r <- r
难 a <- a
19980124-06-010-002 m <- wj
本报 rz <- rz
驻 v <- v
埃及 ns <- ns
记者 n <- n
朱 nrf <- nrf
梦魁 nrg <- nrg
0.91156462585

```

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

```

, wd <- wd
以 c <- p
增加 v <- v

```

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。



```
Debug I/O: Default Project: Wing IDE
Debug I/O
postag.py (pid 13220) (no proc) Debug I/O (stdin, stdout, stde Options
Tagging 101600 / 104299 words...
Tagging 101700 / 104299 words...
Tagging 101800 / 104299 words...
Tagging 101900 / 104299 words...
Tagging 102000 / 104299 words...
Tagging 102100 / 104299 words...
Tagging 102200 / 104299 words...
Tagging 102300 / 104299 words...
Tagging 102400 / 104299 words...
Tagging 102500 / 104299 words...
Tagging 102600 / 104299 words...
Tagging 102700 / 104299 words...
Tagging 102800 / 104299 words...
Tagging 102900 / 104299 words...
Tagging 103000 / 104299 words...
Tagging 103100 / 104299 words...
Tagging 103200 / 104299 words...
Tagging 103300 / 104299 words...
Tagging 103400 / 104299 words...
Tagging 103500 / 104299 words...
Tagging 103600 / 104299 words...
Tagging 103700 / 104299 words...
Tagging 103800 / 104299 words...
Tagging 103900 / 104299 words...
Tagging 104000 / 104299 words...
Tagging 104100 / 104299 words...
Tagging 104200 / 104299 words...
Finish tagging.
Tagged 104299 words.
With precision (HMM) : 0.90963479995
With precision (Trivial) : 0.207758463648
```

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。