一文读懂NLP中的HMM（附代码）

小明 小明AI学习 2017-11-19

导读

本文的大部分知识点整理于Michael Collins 教授的Tagging Problems, and Hidden Markov Models，文末会附上我实现的python代码github链接（github上会有较详细的代码说明，欢迎follow & star & fork 三部曲）

阅读本文之前，推荐阅读上文 一文读懂NLP中的语言模型（附代码）

目录

本文主要分三部分讲解：
1. 认识词性标注和命名实体识别
2. 认识隐马尔可夫模型
3. 通过例子讲述如何“实地操作”

01 词性标注和命名实体识别

POS & NER

词性标注和命名实体识别是常见的NLP任务。

1. 什么是词性标注（POS, Part Of Speech）呢？

从字面上应该可以猜到，就是把句子中每个词的词性识别出来
举个例子：比如说有句子

```
小明 爱 狗
```

词性标注的结果应该是

```
小明/nr 爱/v 狗/n
```

（在这里约定nr为人名， v为动词， n为名词）

2.什么是命名实体识别（NER，**Named Entity Recognition**）呢？

命名实体一般是指把句子中的机构名、人名、地名、专有名词等识别出来

举个例子：比如说有句子

```
小明 在 深圳 向 红十字会 捐款
```

命名实体识别的结果应该为

```
小明/U 在/U 深圳/S 向/U 红十字会/O 捐款/U
```

（在这里约定U为非命名实体， S为地名， O为机构名）

可以看到其实词性标注和命名实体识别任务是非常相似的。下文以词性标注为例讲解。

隐马尔可夫模型

Hidden Markov Models

假设我们已经有标注好的数据

```
sentence = x1 x2 ... xn
tag = y1 y2 ... yn
```

（x和y一一对应）

现在我们要做的是找到一个函数 ***f(x)***，给定句子sentence = x1 x2 ... xn 输入 ***f(x)*** 后能得到对应的词性标记 y1 y2 ... yn，***f(x)*** 应为：

```
f(x1,x2,...xn)
= argmax( p(tag) * p(sentence | tag) )
= argmax( p(sentence, tag) )
```

```
= argmax( p(x1, x2, ..., xn, y1, y2 ... yn) )
```

和上章一样，当语料库很庞大时直接计算
p(x1, x2, ..., xn, y1, y2 ... yn)
不是一个好方案，所以我们需要用到马尔可夫模型了：
假设当前标记 yi 只与前n个元素有关，即如果是bigram的话，那么yi只与yi-1 有关；如果是trigram的话yi只与yi-2, yi-1有关 。

下面以trigram为例。
trigram 隐马尔可夫模型下有如下定义:

```
p(X1=x1, X2=x2, ... , Xn=xn, Y1=y1, Y2=y2, ... , Yn+1=yn+1)
= Π p(Yi=yi | Yi-2=yi-2, Yi-1=yi-1) * Π p( Xi=xi | Yi=yi )
```

（其中yn+1为规定的结束标记STOP，代表句子的结束。当i=1时，会发现y-1, y0并没有定义，所以在这里规定y0 = y-1 = * , * 为规定的开始标记）

我们用频率值来表示概率，有

```
p(X1=x1, X2=x2, ... , Xn=xn, Y1=y1, Y2=y2, ... , Yn+1=yn+1)
= Π q(Yi=yi | Yi-2=yi-2, Yi-1=yi-1) * Π e( Xi=xi | Yi=yi )
```

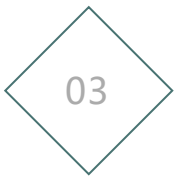
那么 q 和 e 如何算呢?

接下来来定义几个量：
1. 词性标记的集合V
2. 对于u, v, w ∈ V 有

```
c (u, v, w) 为三元组(u, v, w) 出现的次数
c (u, v) 为二元组(u, v) 出现的次数
c (w->x) 为输入词x被标记为w的次数
c (w) 为标记w出现的次数
```

接下来有

```
q(w|u, v) = c (u, v, w) / c(u, v)
e(x|w) = c(w->x) / c(w)
```



如何实地操作

How to do

好了现在你应该对trigram隐马尔可夫模型有一定了解了，接下来举个例子教你怎么实际操作。

假设有语料库：

```
猫/n 抓/v 老鼠/n
狗/n 追/v 猫/n
小明/nr 爱/v 狗/n 和/c 猫/n
```

可知有词性标记集合

```
V = { n, v, nr, c }
```

接下来统计各元组出现的次数

#	二元组	次数
1	(*, *)	3
2	(*, n)	2
3	(n, v)	2
4	(v, n)	3
5	(n, STOP)	3

6	(*, nr)	1
7	(nr, v)	1
8	(n, c)	1
9	(c, n)	1

#	三元组	次数
1	(*, *, n)	2
2	(*, n, v)	2
3	(n, v, n)	2
4	(v, n, STOP)	2
5	(*, *, nr)	1
6	(*, nr, v)	1
7	(nr, v, n)	1
8	(v, n, c)	1
9	(n, c, n)	1
10	(c, n, STOP)	1

#	词性标记	次数
1	n	6
2	v	3
3	nr	1

4	c	1

#	词性对应词	次数
1	n->猫	3
2	n->抓	0
…	…	…
…	v->猫	0
…	v->抓	1
…	…	…

现在假设有句子

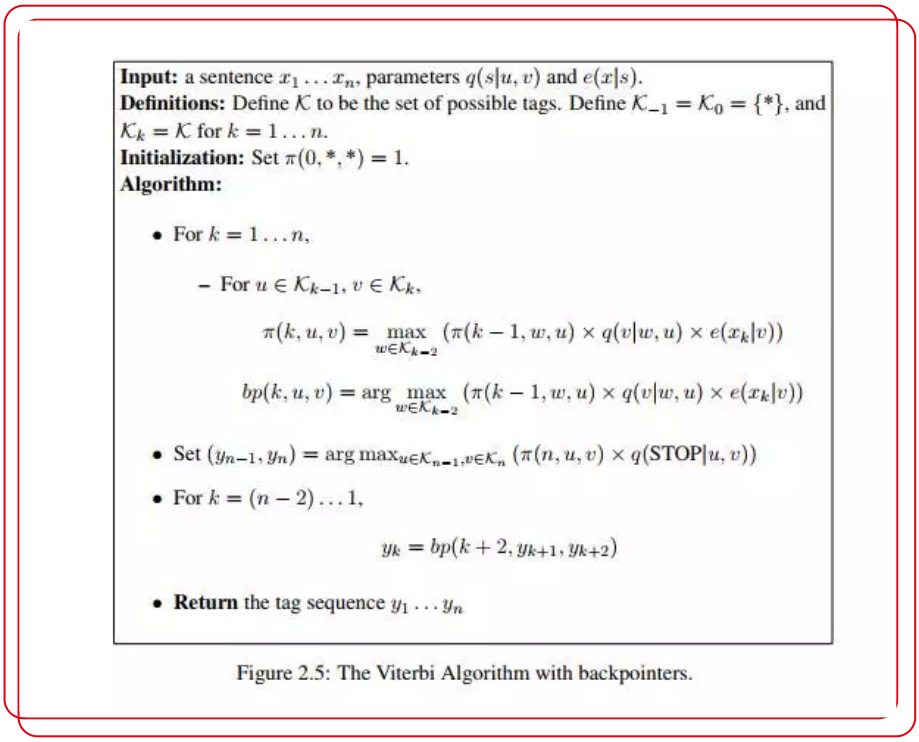
```
sentence = “小明 追 老鼠 和 猫”
```

下面我们用伪代码来描述最简单的解码方法：

```
postags, max_prob, argmax_prob
for word in sentence:
        for tag1 in V:
                for tag2 in V:
                        for tag3 in V:
                         p(x1, x2, ... xn, y1, y2, ... , yn)
                               = q(tag3 | tag1, tag2) * Π e( word | tag3
)
            = [ c(tag1, tag2, tag3) / c(tag1, tag2) ] / [ c(tag3-
>word) / c(tag3) ]
          if p > max_prob:
             max_prob = p
             argmax_prob = tag3
        postags.append( argmax_prob )
```

最后就得到postags就是sentence每个词对应的词性标记了。

不过现在问题来了，上面的方法很简单但效率太低。
那么是否有高效一点的算法呢？有！维特比（Viterbi）算法，Viterbi算法如下图（截屏自Michael Collins 教授的Tagging Problems, and Hidden Markov Models）：



Input: a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
Definitions: Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \dots n$.
Initialization: Set $\pi(0, *, *) = 1$.
Algorithm:

- For $k = 1 \dots n$,
  - For $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$,

$$\pi(k,u,v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

$$bp(k,u,v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

- Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
- For $k = (n-2) \dots 1$,

$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \dots y_n$

Figure 2.5: The Viterbi Algorithm with backpointers.

可以看到上面的Viterbi算法采用了递归实现，在这里不做重点介绍（原文有详尽的介绍），源码中我使用了动态规划实现bigram, trigram HMM的Viterbi算法，欢迎查阅！

后记