自然语言处理技术基础

Natural Language Processing，NLP

网络空间安全与计算机学院

词法分析 Morphological Analysis

中文分词

中文分词是将连续的自然语言文本，切分成具有语义合理性和完整性的词汇序列的过程

词性标注

词性标注（Part-of-Speech tagging 或 POS tagging）是指为自然语言文本中的每个词汇赋予一个词性的过程

专有名词

命名实体识别（Named Entity Recognition 简称NER），即"专名识别"，是指识别自然语言文本中具有特定意义的实体，主要包括人名、地名、机构名、时间日期等

词法分析-百度AI开放平台 (baidu.com)

第7章 词法分析

7.1 中文自动分词及其基本问题

7.2 基本分词方法

7.3 中文姓名识别

7.4 中文自动分词系统的评价

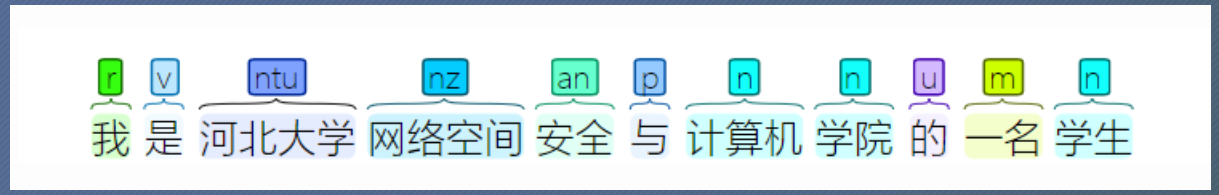
7.5 英语形态还原

7.6 词性标注

7.1 中文自动分词及其基本问题

分类：

- 人工分词：工作量大，准确率高
- 计算机自动分词：速度快，准确率低于人工分词



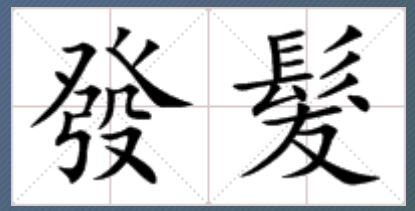
7.1 中文自动分词及其基本问题

应用领域:

- 中文语音合成: 多音字辨识 快乐 音乐; 重量 重新
- 中文输入: 同音字辨识 以为, 一定, 容易
- 中文繁体简体转换: 一个简体字对应多个繁体字
- 中英文聪明选词: 双击鼠标选词

迅速发展的计算机技术

迅速發展的計算機技術



她有一头黑亮的头发

她有一頭黑亮的頭髮

7.1.1 分词规范与词表

《信息处理用现代汉语分词规范》1992

不同目标的应用，对词的要求不同：

1. 校对系统：　　将含有易错字的词和词组作为词单位，分词单位较大
2. 简繁转换系统：　词和词组转换常常是确定的
3. 语音合成系统：　词和词组中，多音字读音是确定的
4. 检索系统：　　注重术语和专名，倾向于分词单位较小化
5. 键盘输入系统：　互现频率高的相互连接的几个字作为输入的单位

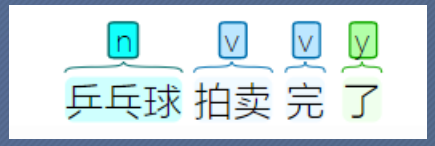
7.1.2 切分歧义问题

1. 交集型切分歧义

【例 7.3】 乒乓球拍卖完了

乒乓球 / 拍卖 / 完 / 了

乒乓球拍 / 卖 / 完 / 了



HanLP

LTP

7.1.2 切分歧义问题

2. 组合型切分歧义

【例 7.4】马上： 他/从/马/上/跳/下/来； 我/马上/就/来

【例 7.5】将来： 他/将/来/我/校/参观； 将来/我/校/会/有/很/大/的/发展

【例 7.6】个人： 屋子/里/只/有/一/个/人； 这/是/我/个人/的/意见

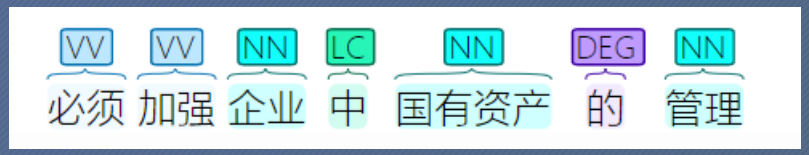
7.1.2 切分歧义问题

3. “真歧义”和“伪歧义”

真歧义：存在两种或两种以上的可实现的切分形式

伪歧义：一般只有一种正确的切分形式

必须加强企业中国有资产的管理



中国有能力解决香港问题

7.1.3 未登录词识别问题

分类：

1. 专有名词：人名、地名、机构名
2. 实体名词：日期、地址、电话号码
3. 衍生词
   - ① 重叠形式：散散步、高高兴兴、一个个
   - ② 派生词：非教师、开发者、洗个澡
   - ③ 离合词：睡了一个觉、留点儿意、发什么时候理
4. 新涌现的普通词汇或专业术语：超女、博客、禽流感
5. 网络用语、拼音缩写：喜大普奔、 xswl：笑死我了

命名实体识别 Named Entity Recognition，NER

命名实体：专有名词 + 实体名词

- 他还兼任何应钦在福州办的东陆军军官学校的政治教官
- 林徽因此时已离开了那里
- 赵微笑着走了
- 南京市长江大桥

识别模板

- 地名识别模板

  LN→LN D* LocKeyWord

  LN→PN D* LocKeyWord

  LN：地名；PN：人名；D：地名中间词； LocKeyWord：地名关键词

- 机构名识别模板

  N→LN D* OrgKeyWord

  ON→PN D* OrgKeyWord

  ON→ON D* OrgKeyWord

  ON：机构名；LN：地名；PN：人名；D：机构名中间词； LocKeyWord：机构名关键词

吴友政 2006 中国科学院自动化研究所 http://www.irgrid.ac.cn/handle/1471x/976582#

7.2 基本分词方法

1、基于词典分词算法（字符串匹配分词算法）

把句子按照字典切分成词，再寻找词的最佳组合方式

常用算法：
最大匹配法、最短路径法、基于n-gram model的算法等

2、基于统计的机器学习算法（基于字的分词）

由字构词，先把句子分成一个个字，再将字组合成词，
寻找最优的切分策略，同时也可以转化成序列标注问题。

常用算法：
N-Gram、HMM、CRF、SVM、深度学习 等

基本的分词方法包括：

① 最大匹配法
② 最少分词法（最短路径法）
③ 最大概率法（最短加权路径法）
④ 与词性标注相结合的分词方法
⑤ 基于互现信息的分词方法
⑥ 基于字分类的方法
⑦ 基于实例的汉语分词方法

https://zhuanlan.zhihu.com/p/50444885

https://github.com/liuhuanyong/WordSegment

1.最大匹配法 Maximum Match Method

需要一个词表，用文本的候选词去跟词表中的词匹配
如果匹配成功，则认为候选词是词，予以切分；
否则就认为不是词。

原则：尽可能的用最长的词来匹配句子中的汉字串

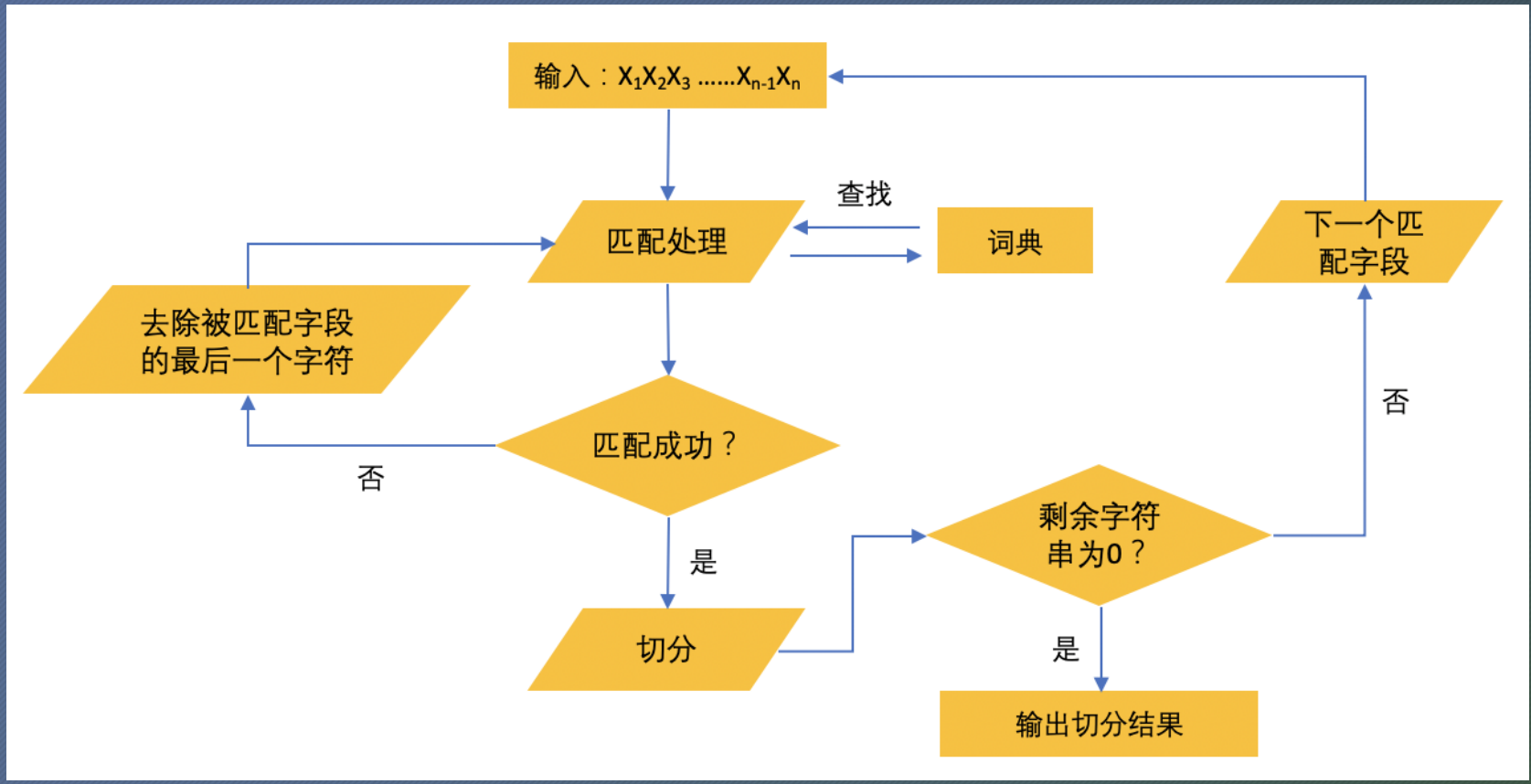
优点：程序简单易行，开发周期短，不需要词法、句法、语义资源

缺点：切分歧义消除的能力差（比如：“使用户满意”->“使用/户/满意”）

1.最大匹配法-算法：

（1）待切分汉字串s1，已切分汉字串s2（s2初始为空串）
（2）如果s1为空串，转（6）
（3）从s1的左边赋值一个子串w作为候选词，尽可能长，但不要超过MaxWordLength
（4）如果在词表中找到w，或者w的长度为2（也就是一个汉字），
　　那么将w和一个词界标记一起加到s2的右边，并从s1的左边去掉w，转（2）
（5）去掉w中最后一个汉字，转（4）
（6）结束

1.最大匹配法-流程图



https://www.cnblogs.com/jielongAI/p/10362314.html

1.最大匹配法-编程实现：

```
user_dict = ['时间', '就', '是', '生命'] # 词典
s1 = '时间就是生命' # 待切分句子
s2 = ''
w = '' #从s1取出的字
step = 0
index = 4

while 1:
    if len(s1) <= 0 :
        break
    step += 1

    w = s1[0:index]

    if(w in user_dict):
        s2 = s2 + w + '/'
        s1 = s1[index:len(s1)]
        index = 4
        if index > len(s1):
            index = len(s1)
    else:
        index += -1

    fmt = "{:1}\t{:15}\t{:15}\t{:15}"

    if s1=='':
        print(fmt.format(step,'null\t',s2,w))
    elif s2== '':
        print(fmt.format(step,s1,'null\t',w))
    else:
        print(fmt.format(step,s1,s2,w))
```

1	时间就是生命	null	时间就是
2	时间就是生命	null	时间就
3	就是生命	时间/	时间
4	就是生命	时间/	就是生命
5	就是生命	时间/	就是生
6	就是生命	时间/	就是
7	是生命	时间/就/	就
8	是生命	时间/就/	是生命
9	是生命	时间/就/	是生
10	生命	时间/就/是/	是
11	null	时间/就/是/生命/	生命

```
sentence = '使用户满意' # 待切分句子
user_dict = ['使用', '用户', '满意'] # 词典
max_len = max([len(item) for item in user_dict])# 词典中最长词长度
start = 0

while start != len(sentence):
    index = start + max_len
    if index>len(sentence):
        index = len(sentence)
    for i in range(max_len):
        if (sentence[start:index] in user_dict) or (len(sentence[start:index])==1):
            print(sentence[start:index], end='/')
            start = index
            break
        index += -1
```

使用/户/满意/

2.最少分词法（最短路径）

基本思想：使每一句中切出的词数最少

可以将最少分词法看成是“最短路径搜索”问题：

根据词典，找出字串中所有可能的词，构造词语切分的无环有向图DAG，每个词对应图中的一条有向边，边长为1。

然后，针对该切分图，在起点到终点的所有路径中，寻找长度最短的路径。

优点：同最大匹配法类似

缺点：当最短路径有多条的时候，选择最终的输出结果缺乏应有的标准

N-最短路径法（前N个最短路径）

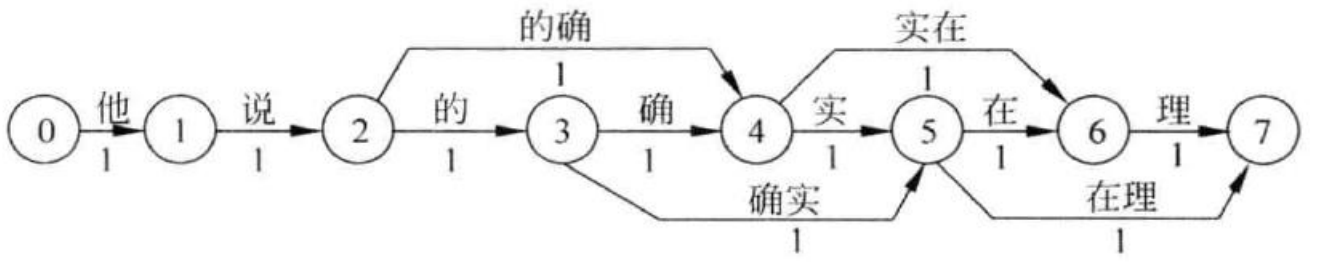


Table (4)

编号	长度	前驱
1	3	(2,1)
2	4	(3,1)

Table (5)

编号	长度	前驱
1	4	(3,1) (4,1)
2	5	(4,2)

Table (6)

编号	长度	前驱
1	4	(4,1)
2	5	(4,2) (5,1)
3	6	(5,2)

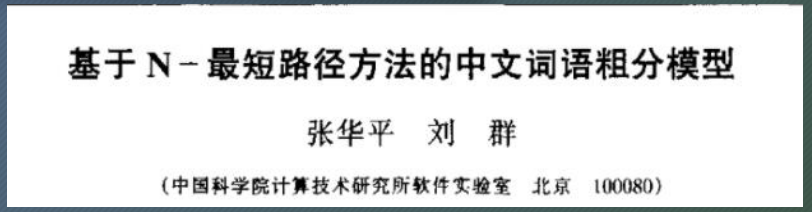
Table (7)

编号	长度	前驱
1	5	(5,1) (6,1)
2	6	(5,2) (6,2)
3	7	(6,3)

Table (2)　Table (3)

(1,0) ← (2,0)

图 7-2　句子“他说的确实在理”的求解过程（$N=3$）



基于 N－最短路径方法的中文词语粗分模型

张华平　刘　群

（中国科学院计算技术研究所软件实验室　北京　100080）

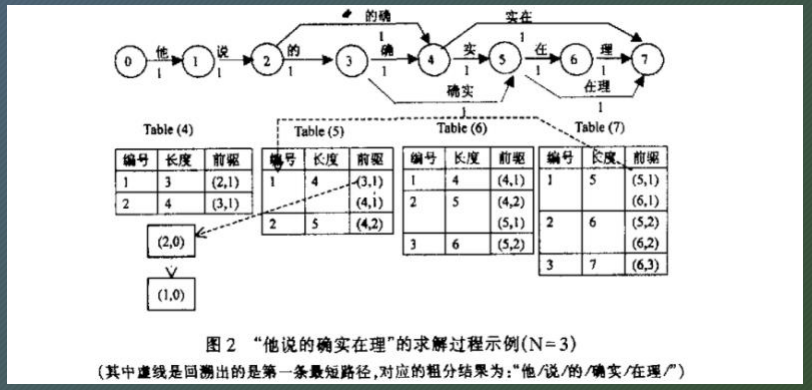


图 2　“他说的确实在理”的求解过程示例（N=3）

（其中虚线是回溯出的是第一条最短路径，对应的粗分结果为：“他/说/的/确实/在理/”）

- 收稿日期：2001－12－18

基金项目：国家重点基础研究项目（G1998030507－4，G1998030510）.

作者张华平，男，1978 年生，硕士生，主要研究方向为自然语言处理与中文词语分析. 刘群，男，1966 年生，在职博士生，副研究员，主要研究方向为机器翻译，自然语言处理与中文信息处理.

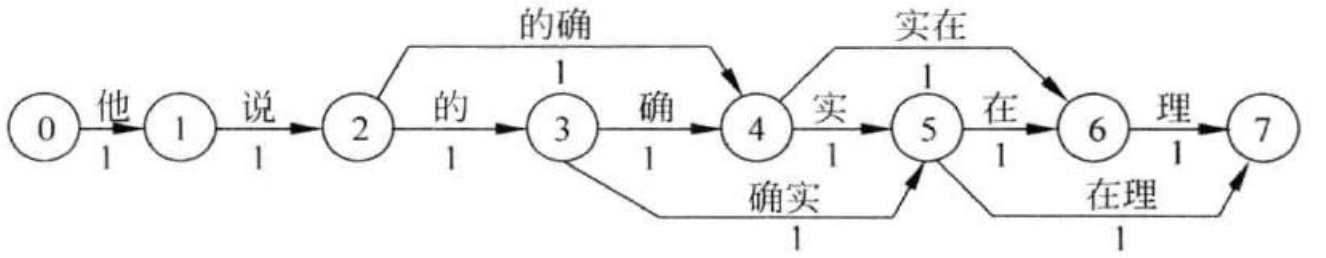


Table (4)

编号	长度	前驱
1	3	(2,1)
2	4	(3,1)

Table (5)

编号	长度	前驱
1	4	(3,1) (4,1)
2	5	(4,2)

Table (6)

编号	长度	前驱
1	4	(4,1)
2	5	(4,2) (5,1)
3	6	(5,2)

Table (7)

编号	长度	前驱
1	5	(5,1) (6,1)
2	6	(5,2) (6,2)
3	7	(6,3)

Table (2) (1,0) Table (3) (2,0)

图 7-2　句子“他说的确实在理”的求解过程($N=3$)

虚线是回溯出的是第一条最短路径,对应的粗分结果为:“他/说/的/确实/在理/”,

Table(2),Table(3)…Table(7)分别为结点2、3、…、7对应的信息记录表,

Table(0)、 Table(1)的信息记录表没有给出。

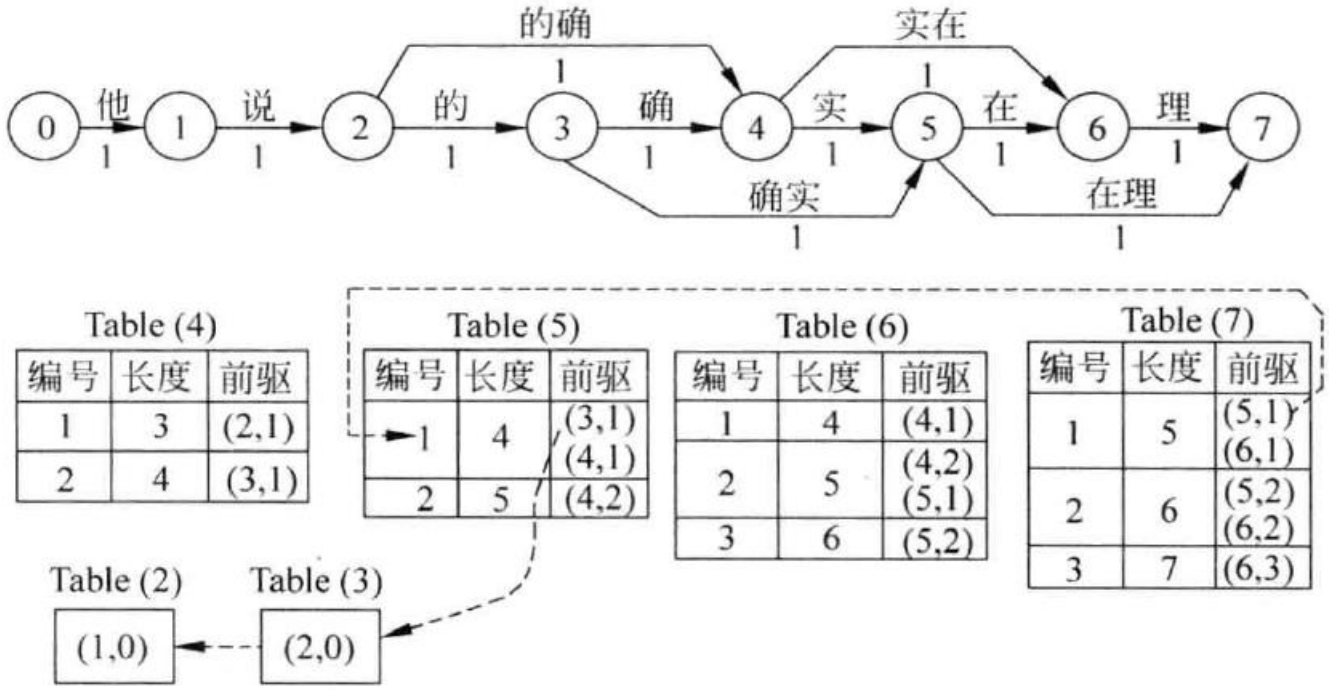


图 7-2　句子“他说的确实在理”的求解过程（$N=3$）

每个结点的信息记录表里的编号为路径不同长度的编号，按由小到大的顺序排列，编号最大不超过N。

如Table(5)表示从结点0出发到达结点5有：

2条长度为4的路径(0-1-2-4-5和0-1-2-3-5)

1条长度为5的路径(0-1-2-3-4-5)。

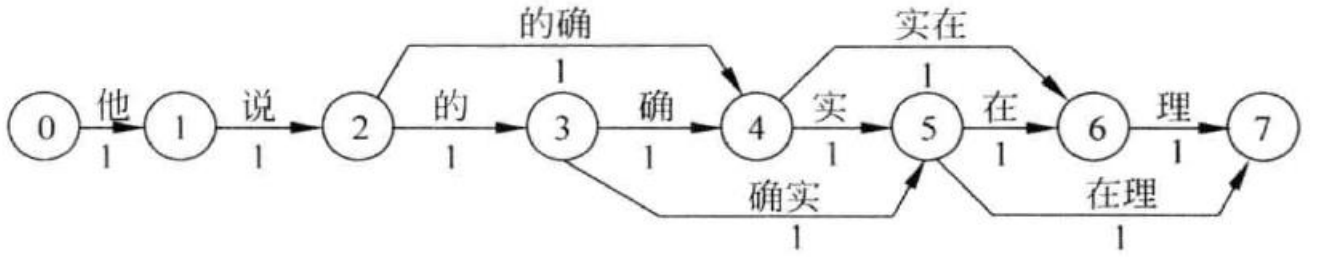


Table (4)

编号	长度	前驱
1	3	(2,1)
2	4	(3,1)

Table (5)

编号	长度	前驱
1	4	(3,1) (4,1)
2	5	(4,2)

Table (6)

编号	长度	前驱
1	4	(4,1)
2	5	(4,2) (5,1)
3	6	(5,2)

Table (7)

编号	长度	前驱
1	5	(5,1) (6,1)
2	6	(5,2) (6,2)
3	7	(6,3)

Table (2) (1,0)　Table (3) (2,0)

图 7-2　句子“他说的确实在理”的求解过程($N=3$)

前驱(i,j)表示沿着当前路径到达当前结点的最后一条边的出发结点为i,
即当前结点的前一个结点为i，相应的边为结点i的信息记录表中编号为j的路径。
如果j=0,表示没有其他候选的路径。
如结点7对应的信息记录表 Table(7)中,
编号为1的路径前驱(5,1)表示:前一条边为结点5的信息表中第1条路径。

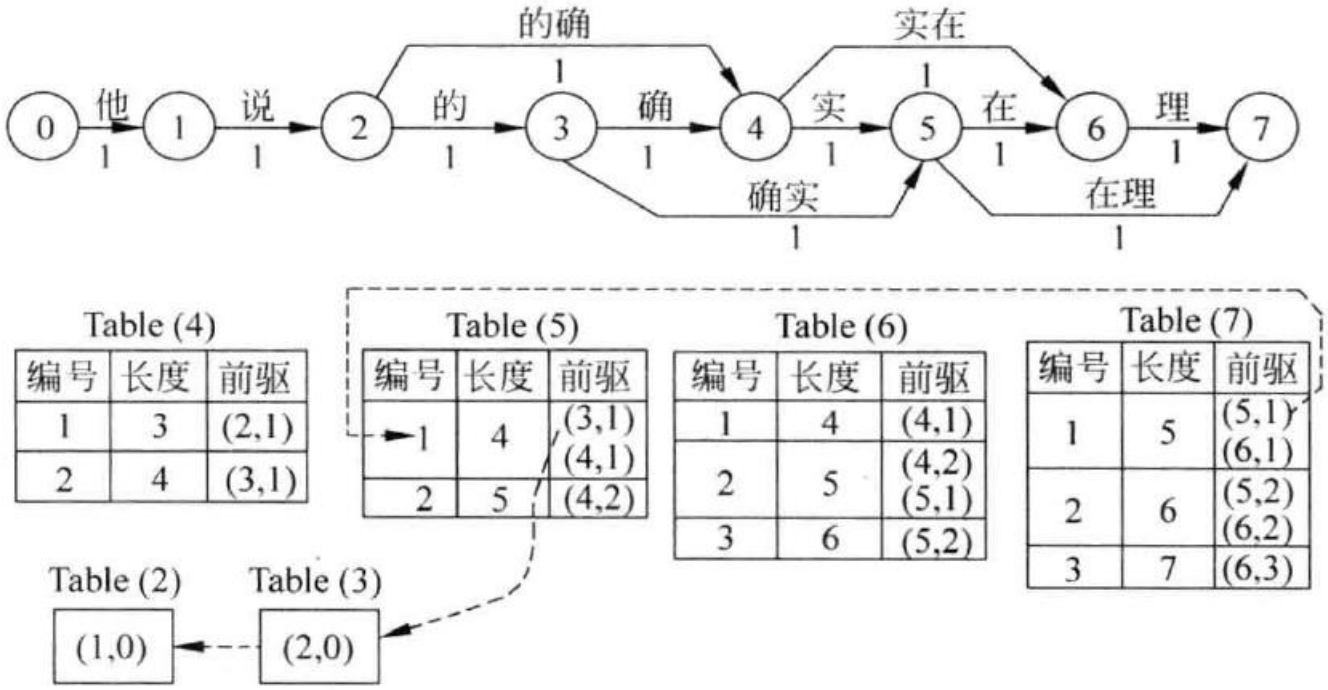


图 7-2　句子“他说的确实在理”的求解过程($N=3$)

Table(5)中的(3,1)表示:前驱为结点3的信息记录表中的第1条路径。

Table(3)中的(2,0)表示:前驱为结点2,没有其他候选路径。

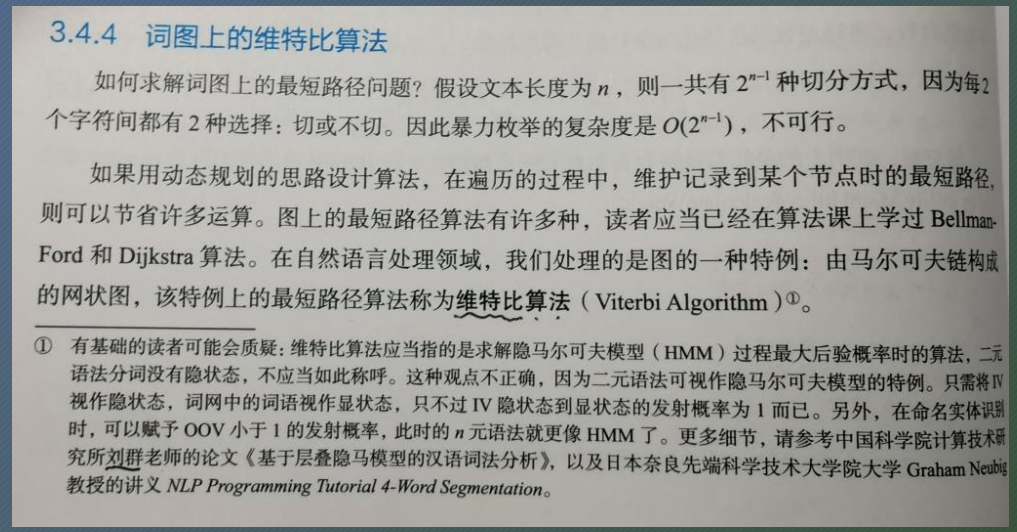
信息记录表为系统回溯找出所有可选路径提供了依据。

词图上的维特比算法

图的一种特例：马尔科夫链构成的网状图

该特例上的最短路径算法：维特比算法

二元语法可视作HMM的特例：
将IV视作隐状态，
词网中的词语视作显状态，
发射概率为1。



3.4.4 词图上的维特比算法

如何求解词图上的最短路径问题？假设文本长度为 n，则一共有 2^{n-1} 种切分方式，因为每2个字符间都有 2 种选择：切或不切。因此暴力枚举的复杂度是 $O(2^{n-1})$，不可行。

如果用动态规划的思路设计算法，在遍历的过程中，维护记录到某个节点时的最短路径，则可以节省许多运算。图上的最短路径算法有许多种，读者应当已经在算法课上学过 Bellman-Ford 和 Dijkstra 算法。在自然语言处理领域，我们处理的是图的一种特例：由马尔可夫链构成的网状图，该特例上的最短路径算法称为**维特比算法**（Viterbi Algorithm）①。

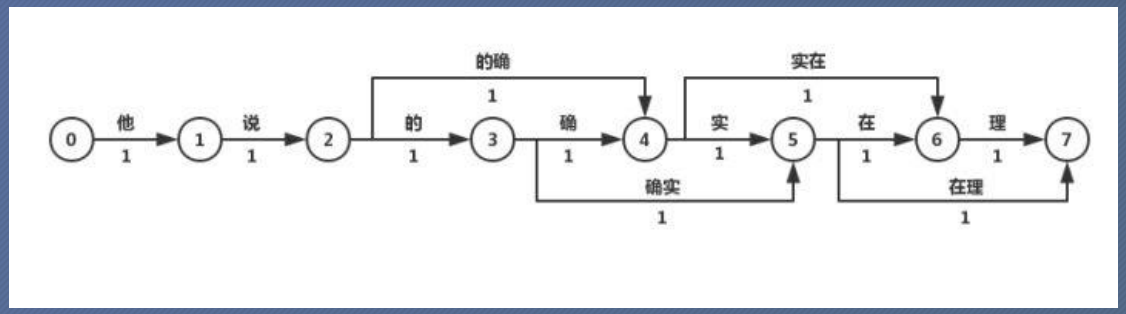
① 有基础的读者可能会质疑：维特比算法应当指的是求解隐马尔可夫模型（HMM）过程最大后验概率时的算法，二元语法分词没有隐状态，不应当如此称呼。这种观点不正确，因为二元语法可视作隐马尔可夫模型的特例。只需将 IV 视作隐状态，词网中的词语视作显状态，只不过 IV 隐状态到显状态的发射概率为 1 而已。另外，在命名实体识别时，可以赋予 OOV 小于 1 的发射概率，此时的 n 元语法就更像 HMM 了。更多细节，请参考中国科学院计算技术研究所刘群老师的论文《基于层叠隐马模型的汉语词法分析》，以及日本奈良先端科学技术大学院大学 Graham Neubig 教授的讲义 *NLP Programming Tutorial 4-Word Segmentation*。

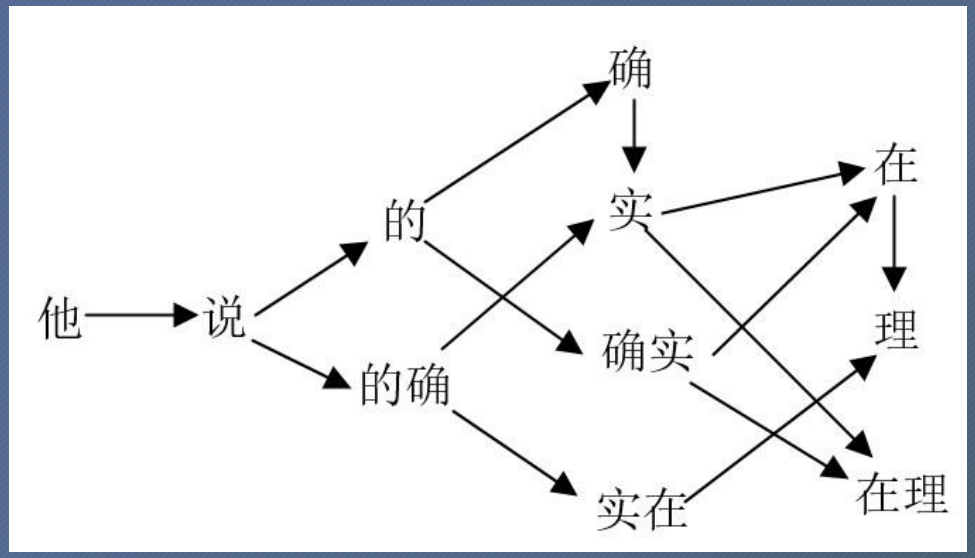
《自然语言处理入门》– 何晗 - P112

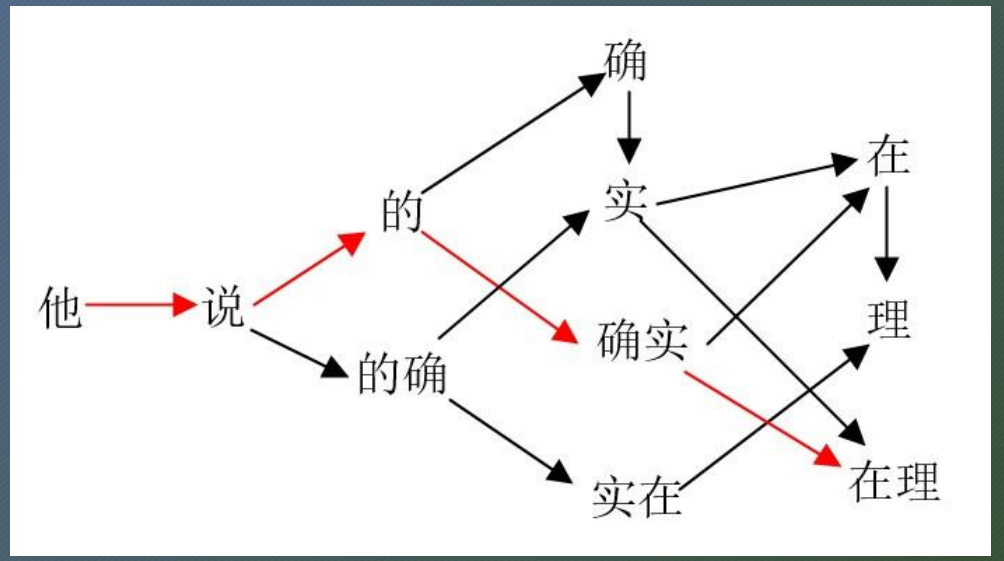
https://github.com/NLP-LOVE/Introduction-NLP

Viterbi算法

http://zeag.farbox.com/post/fen-ci-ren-wu-step-by-step







《统计自然语言处理》

CODE

```
def viterbi(self, word_net):
    word_list = []
    for i,item in enumerate(word_net):
        if i == len(word_net)-1:
            break
        for from_node in item:
            if from_node.s == "#开始#":
                to_nodes = word_net[i+1]
            else:
                to_nodes = word_net[i + len(from_node.s)]
            for to_node in to_nodes:
                dis = from_node.dis + self.calcDistance(from_node.s, to_node.s)
                if to_node.from_node == None or to_node.dis > dis:
                    to_node.from_node = from_node
                    to_node.dis = dis
    word = word_net[-1][0]
    while(word.from_node.s != "#开始#"):
        word_list.append(word.from_node.s)
        word = word.from_node
    return word_list[::-1]
```

```
if __name__ == '__main__':
    s = "他说的确实在理"
    demo = Segdemo()
    word_net = demo.conventToWordNet(s)
    word_list = demo.viterbi(word_net)
    print(word_list)
```

['他', '说', '的', '确实', '在理']

3.最大概率法

$Z = z_1 z_2 \cdots z_n$ 表示字串，

$W = w_1 w_2 \cdots w_m$ 表示切分后的词串。

汉语词语切分可以看作是：

求使P(W|Z)最大的切分

$p(W|Z) = P(W)P(Z|W) / P(Z)$

P(Z)是汉字串的概率，它对于各个候选词串都是一样的，不必考虑；

P(Z|W)表示出现词串的条件下汉字串的概率，显然该值为1，不必考虑；

仅需考虑P(W)即词串的概率。

词串的概率可以通过**n元语法**来求

用二元语法 $P(W)=p(w_1|w_0)p(w_2|w_1)\cdots p(w_m|w_{m-1})$ （1）

用一元语法 $P(W)=p(w_1)p(w_2)\cdots p(w_m)$ （2）

7.2.2 基于词的 n 元语法模型的分词方法

基于词的 n 元文法模型是一个典型的生成式模型，早期很多统计分词方法均以它为基本模型，然后配合其他未登录词识别模块进行扩展。其基本思想是：首先根据词典（可以是从训练语料中抽取出来的词典，也可以是外部词典）对句子进行简单匹配，找出所有可能的词典词，然后，将它们和所有单个字作为结点，构造的 n 元的切分词图，图中的结点表示可能的词候选，边表示路径，边上的 n 元概率表示代价，最后利用相关搜索算法（如 Viterbi 算法）从图中找到代价最小的路径作为最后的分词结果。以输入句子“研究生物学”为例，图 7-3 给出了基于二元文法的切分词图。

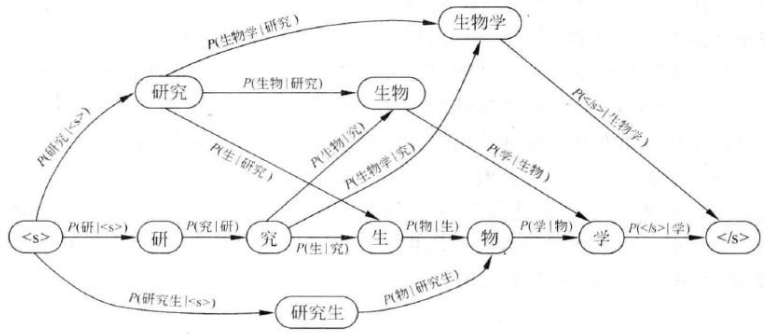


图 7-3 基于词的生成式模型的二元文法切分词图

算法描述

(1)对一个待分词的字串S,按照从左到右的顺序取出全部候选词W1,W2,.... Wi , Wn;

(2)计算每个候选词的概率值P(Wi),记录每个候选词的全部左邻词;

(3)计算每个候选词的累计概率，累计概率最大的候选词为最佳左邻词;

(4)如果当前词Wn是字串的尾词,且累计概率P(Wn)最大，则是S的终点词;

(5)从Wn开始，按照从右到左顺序，依次将每个词的最佳左邻词输出，即S的分词结果。

https://blog.csdn.net/chase1998/article/details/83247192

输入句子：结合成分子时

第一步：遍历整句话，找出全部候选词和在语料库中的词频

候选词	词频
结	0.0037%
结合	0.0353%
合	0.0049%
合成	0.0006%
成	0.0423%
成分	0.0023%
分	0.0312%
分子	0.0038%
子	0.0010%
时	0.1043%

输入句子：结合成分子时

第二步：找出全部左邻词

候选词	备选左邻词
结	
结合	
合	结
合成	结
成	结合、合
成分	结合、合
分	合成、成
分子	合成、成
子	成分、分
时	分子、子

输入句子：结合成分子时

第三步：计算累计概率，选出最佳左邻词

候选词	累计概率	备选左邻词（前趋词）	最佳左邻词
结	0.0037%	NULL	NULL
结合	0.0353%	NULL	NULL
合	0.0049×0.0037 =0.00001813%	结0.0037	结
合成	0.0006×0.0037 =0.00000222%	结0.0037	结
成	0.0423×0.0353 =0.00149319%	结合0.0353、 合0.00001813	结合
成分	0.0023×0.0353 =0.00008119%	结合0.0353、 合0.00001813	结合
分	0.0312×0.00149319 =0.0000465875%	合成0.00000222、 成0.00149319	成
分子	0.0038×0.00149319 =0.00000567412%	合成0.00000222、 成0.00149319	成
子	0.0010×0.00008119 =0.00000008119%	成分0.00008119、 分0.0000465875	成分
时	0.1043×0.00000567412=0.000000591811%	分子0.00000567412、子0.00000008119	分子

候选词	词频
结	0.0037%
结合	0.0353%
合	0.0049%
合成	0.0006%
成	0.0423%
成分	0.0023%
分	0.0312%
分子	0.0038%
子	0.0010%
时	0.1043%

第四步：输出结果

结合/成/分子/时

【注】概率比较低，接近于零，不便于在机器上表示。
可采取两侧取负对数的方法。
教材：P104

CODE

```python
def find_word_in_dict(s):
    """
    在字典中查找候选词
    :param s: 输入句子
    :return: 返回字典==>"词:词频|候选左邻词1/候选左邻词2"
    """
    freq_dict = load_dict()
    result = {}
    for index in range(0, len(s)):  # 遍历所有字
        for wordLen in range(0, len(s) - index):  # 遍历该字的所有可能词
            seg_word = s[index:index + wordLen + 1]
            if seg_word in freq_dict.keys():
                # 找到候选词，找其左邻词
                left_words = ''
                for word_len in range(index, 0, -1):  # 在之前的候选词列表里找左邻词（最大词长开始）
                    for k in result.keys():
                        if s[index - word_len:index] == k.split('-')[1]:
                            left_words += str(index - word_len) + '-' + s[index - word_len:index] + '/'
                # 返回候选词及其语料库词频和候选左邻词
                result[str(index) + '-' + seg_word] = freq_dict[seg_word] + '|' + left_words
    return result


def cl_probability(words_dict):
    """
    计算累加概率并选择最佳左邻词
    :param words_dict: "词:词频|候选左邻词1/候选左邻词2"
    :return:返回新字典==>"词:累计概率|最佳左邻词"
    """
    for k, v in words_dict.items():
        freq = v.split('|')[0][:-1]
        left_words = v.split('|')[1]
        if left_words == '':
            continue
        else:
            left_word = left_words.split("/")
            max_left_p = 0.0
            which_word = ''
            for num in range(0, len(left_word) - 1):
                curr_left_word_p = float(words_dict[left_word[num]].split('|')[0][:-1])
                if curr_left_word_p > max_left_p:  # 比较当前左邻词的累计概率
                    max_left_p = curr_left_word_p
                    which_word = left_word[num]
            curr_max_p = float(max_left_p) * float(freq)
            # 用最大累计概率替换原来的词频,用最佳左邻词替换候选左邻词
            words_dict[k] = v.replace(freq, str(curr_max_p)).replace(left_words, which_word)
    return words_dict
```

```python
def seg(sentence):
    """
    接收输入，调用函数并输出
    :param sentence:
    :return: 分词后的句子
    """
    words_dict = find_word_in_dict(sentence)
    best_words_dict = cl_probability(words_dict)
    seg_line = ''
    keys = list(best_words_dict.keys())
    key = keys[-1]
    while key != '':
        seg_line = key.split('-')[1] + '/ ' + seg_line
        key = best_words_dict[key].split('|')[1]
    return seg_line


if __name__ == '__main__':
    print("概率最大化分词-请输入待切分句子：")
    input_str = input()
    print(seg(input_str))
```

```
概率最大化分词-请输入待切分句子：
结合成分子时
结合/ 成/ 分子/ 时/
```

4.与词性标注结合的分词方法

基本思想:

将分词和词性标注结合起来,
利用丰富的词类信息对分词决策提供帮助,
并且在标注过程中又反过来对分词进行校验、调整,
从而极大的提高切分的准确度

(7.6.3 HMM词性标注法)

算法举例:

对于“他俩谈恋爱是从头年元月开始的”，有两种切分方式:

（1）...是|从头|年|元月|...

（2）...是|从|头年|元月|...

虽然 “从头”、“年”的词频之积 大于 “从”、“头年”的词频之积，

但是 词性序列“动词+副词+时间两次+时间词”的概率

远小于 词性序列“动词+介词+时间词+时间词”的概率，

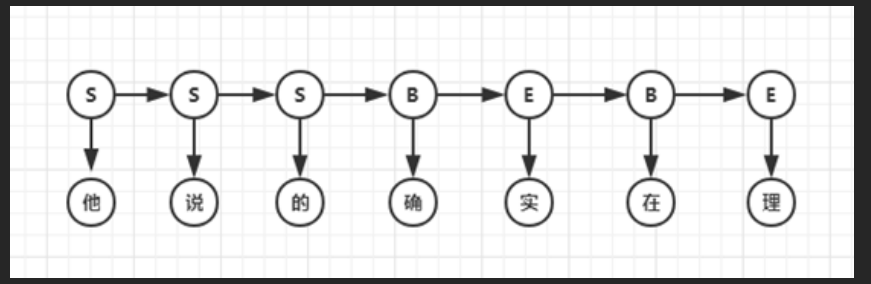
所以选择切分方式2作为结果。

5.基于互现信息的分词方法

从形式上看，词是一个稳定的字组合，
相邻的字同时出现的次数越多，就越可能构成一个词，
可以对语料中相邻的各字组合的频度进行统计，计算他们的互现信息。

互现信息体现了汉字间结合关系的紧密程度。
当紧密程度高于某一阈值时，
便可以认为是一个词。

6.基于字分类的分词方法



将分词过程看做是字的分类问题，
每个字在构造一个特定的词语时都占据一个确定的构词位置（词位）

汉语句子作为输入，**“BEMS”序列串**作为输出，然后进行切词，进而得到输入句子的划分。

B：起始字
M：中间字
E：结束字
S：单字成词

例句：小明硕士毕业于中国科学院计算所
序列：B E B E B M E B E B M E B E S
切词：B E / B E / B M E / B E / B M E / B E / S
进而：小明/硕士/毕业于/中国/科学院/计算/所

基于HMM的分词方法

https://blog.csdn.net/riverflowrand/article/details/50057323

观测序列 O：小明硕士毕业于中国科学院计算所

状态序列 S：BEBEBMEBEBMEBES

初始状态矩阵 π：句子的第一个字属于{B,E,M,S}这四种状态的概率

状态转移矩阵 A：如果前一个字位置是B，那么后一个字位置为BEMS的概率各是多少

观测概率矩阵 B：在状态B的条件下，观察值为“耀”的概率，取对数后是-10.460

	P
B	-0.263
E	-3.14e+100
M	-3.14e+100
S	-1.465

初始状态矩阵 π

	B	E	M	S
B	-3.14e+100	-0.511	-0.916	-3.14e+100
E	-0.590	-3.14e+100	-3.14e+100	-0.809
M	-3.14e+100	-0.333	-1.260	-3.14e+100
S	-0.721	-3.14e+100	-3.14e+100	-0.666

状态转移矩阵 A

	耀	涉	谈	伊	洞	…
B	-10.460	-8.766	-8.039	-7.683	-8.669	…
E	-9.267	-9.096	-8.436	-10.224	-8.366	…
M	-8.476	-10.560	-8.345	-8.022	-9.548	…
S	-10.006	-10.523	-15.269	-17.215	-8.370	…

观测概率矩阵 B

用Viterbi算法求解中文分词问题

		小	明	硕	士	毕	业	于	中	国	科	学	院	计	算	所
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
E	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
M	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
S	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

二维数组 weight[4][15]，4是状态数(0:B,1:E,2:M,3:S)，15是输入句子的字数。
比如 weight[0][2] 代表 状态B的条件下，出现'硕'这个字的可能性。

二维数组 path[4][15]，4是状态数(0:B,1:E,2:M,3:S)，15是输入句子的字数。
比如 path[0][2] 代表 weight[0][2]取到最大时，前一个字的状态，
比如 path[0][2] = 1, 则代表 weight[0][2]取到最大时，前一个字(也就是明)的状态是E。

记录前一个字的状态是为了使用viterbi算法计算完整个 weight[4][15] 之后，
能对输入句子从右向左地回溯回来，找出对应的状态序列。

确定边界条件和路径回溯

weight[4][15] 和 path[4][15] 计算完毕后，**确定边界条件和路径回溯**

边界条件如下：
对于每个句子，最后一个字的状态只可能是 E 或者 S，不可能是 M 或者 B。
所以在本文的例子中我们只需要比较 weight[1(E)][14] 和 weight[3(S)][14] 的大小即可。

在本例中：
weight[1][14] = -102.492;　weight[3][14] = -101.632;　所以 S > E，路径回溯的起点是 path[3][14]。

回溯的路径是:　SEBEMBEBEMBEBEB
倒序一下就是:　BE/BE/BME/BE/BME/BE/S
切词结果就是:　小明/硕士/毕业于/中国/科学院/计算/所

HMM中文分词

发射概率矩阵 B

```
cnt=0
for key,value in emit_P.items():
    cnt += 1
    if cnt>2:
        break
    print("{}:{}".format(key,value))
```

```
B:{'一': -3.6544978750449433, '丁': -8.125041941842026,
33, '三': -5.932085850549891, '上': -5.739552583325728,
9, '丐': -9.985251083961709, '丑': -10.200388187382178,
47, '丘': -8.88385596143092, '丙': -10.895131537474946,
5, '丝': -8.859361493580714, '丞': -10.149094892994627,
8, '严': -6.980651489152666, '並': -7.383967650611162,
```

状态转移矩阵 A

```
print(json.dumps(trans_P, indent=4,
```

```
{
    "B":{
        "E":-0.51082562376599,
        "M":-0.916290731874155
    },
    "E":{
        "B":-0.5897149736854513,
        "S":-0.8085250474669937
    },
    "M":{
        "E":-0.33344856811948514,
        "M":-1.2603623820268226
    },
    "S":{
        "B":-0.7211965654669841,
        "S":-0.6658631448798212
    }
}
```

初始状态矩阵 π

```
print(json.dumps(start_P, indent=4,
```

```
{
    "B":-0.26268660809250016,
    "E":-3.14e+100,
    "M":-3.14e+100,
    "S":-1.4652633398537678
}
```

```
def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]  # tabular
    path = {}
    for y in states:  # init
        V[0][y] = start_p[y] + emit_p[y].get(obs[0], MIN_FLOAT)
        path[y] = [y]
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}
        for y in states:
            em_p = emit_p[y].get(obs[t], MIN_FLOAT)
            (prob, state) = max(
                [(V[t - 1][y0] + trans_p[y0].get(y, MIN_FLOAT) + em_p, y0) for y0 in PrevStatus[y]])
            V[t][y] = prob
            newpath[y] = path[state] + [y]
        path = newpath

    (prob, state) = max((V[len(obs) - 1][y], y) for y in 'ES')

    return (prob, path[state])
```

```
cut("小明硕士毕业于中国科学院计算所")
```

```
[]
-101.63238958952303
['B', 'E', 'B', 'E', 'B', 'M', 'E', 'B', 'E', 'B', 'M', 'E', 'B', 'E', 'S']
['小明']
['小明', '硕士']
['小明', '硕士', '毕业于']
['小明', '硕士', '毕业于', '中国']
['小明', '硕士', '毕业于', '中国', '科学院']
['小明', '硕士', '毕业于', '中国', '科学院', '计算']
['小明', '硕士', '毕业于', '中国', '科学院', '计算', '所']
['小明', '硕士', '毕业于', '中国', '科学院', '计算', '所']

['小明', '硕士', '毕业于', '中国', '科学院', '计算', '所']
```

https://www.cnblogs.com/xlturing/p/8467033.html

https://github.com/xlturing/machine-learning-journey/tree/master/seg_hmm

7.基于实例的汉语分词方法

在训练语料中存放事先切分好的汉字串，
为以后输入的待切分句子提供可靠的实例。

在分词的时候，根据输入句子和训练语料，
找到所有切分片段的实例和可能的词汇，
依据某些优化原则和概率信息寻求最优词序列。

各种分词方法对语言学资源的利用情况

序号	分词方法	词典	语料库
1	最大匹配法	√	
2	最少分词法	√	
3	最大概率法	√	√
4	基于HMM的分词方法 与词性相结合	√	√
5	基于互现信息的分词方法		√
6	基于字分类的分词方法		√

7.3 中文姓名识别

命名实体识别：人名、机构名、地址。

7.3.1 基于规则的方法

7.3.2 基于统计的方法

7.3.1 基于规则的方法

建立姓名识别规则

序号	条件	识别结果	举例
1	“老/小” + 姓氏用字	是	小王、老李
2	姓氏用字 + “工/总”	是	张工、陈总
3	数词 + 可做量词的姓氏用字	否	一周、第七章
4	“多/各” + “方/项/章/段”等	否	多方筹备、各项准备
5	只能或几乎总是用作姓氏的词 + 双字词	是	罗胜利、陈建国
6	只能或几乎总是用作姓氏的词 + 单字 + “标点符号/的/了/是/动词/非单字词”	是	瑞金医院的陈柯 主治医师毛羽说
7	只能或几乎总是用作姓氏的词 + 单字 + “标点符号/的/了/是/在/动词”的单字	是	顾筑胜、吴俊洲

7.3.2 基于统计的方法

建立人名识别的统计模型

“李袁沁明不明白”候选姓名：李袁、李袁沁、袁沁、袁沁明

选择结果中概率最大的做姓名

p(姓名|李袁) = p(X|李) p(M|袁) p(XM)
p(姓名|李袁沁) = p(X|李) p(M|袁) p(M|沁) p(XMM)
p(姓名|袁沁) = p(X|袁) p(M|沁) p(XM)
p(姓名|袁沁明) = p(X|袁) p(M|沁) p(M|明) p(XMM)

X：姓，M：名，p(XM)单姓单名概率，p(XMM)单姓双名概率

对比两种方法

	基于规则的方法	基于统计的方法
特点	利用语言规则来进行人名识别	仅从字、词本身来考虑， 通过计算字、词做人名用的概率来实现
优点	准确率较高	占用的资源少、速度快、效率高
缺点	① 系统庞大复杂，耗费资源多，效率不高 ② 很难列举所有规则 ③ 规则之间往往会顾此失彼，产生冲突	① 准确率较低 ② 合理性、科学性及所用统计源的可靠性、代表性、合理性难以保证 ③ 搜集合理的有代表性的统计源的工作本身也较难。

《基于角色标注的中国人名自动识别研究》

张华平　刘群　中国科学院计算技术研究所　《计算机学报》　2004年01期：http://www.cnki.com.cn/Article/CJFDTotal-JSJX200401009.htm

根据在人名识别中的作用，采取Viterbi算法对切词结果进行角色标注，
在角色序列的基础上，进行模式最大匹配，最终实现中国人名的识别。
通过对 16M字节真实语料库的封闭与开放测试，该方法取得了接近 98%的召回率。

CODE：HMM——NER

```python
def train(self, corpus_path):
    """函数说明： 训练HMM模型，得到模型参数pi,A,B"""
    with open(corpus_path, mode='r', encoding='utf-8') as fr:
        lines = fr.readlines()
    print('开始训练数据：')
    for i in tqdm(range(len(lines))):
        if len(lines[i]) == 1:
            continue
        else:
            cur_char, cur_tag = lines[i].split()
            self.B[self.tag2idx[cur_tag]][ord(cur_char)] += 1
            if len(lines[i - 1]) == 1:
                self.pi[self.tag2idx[cur_tag]] += 1
                continue
            pre_char, pre_tag = lines[i - 1].split()
            self.A[self.tag2idx[pre_tag]][self.tag2idx[cur_tag]] += 1
    self.pi[self.pi == 0] = self.epsilon  # 防止数据下溢,对数据进行对数归一化
    self.pi = np.log(self.pi) - np.log(np.sum(self.pi))
    self.A[self.A == 0] = self.epsilon
    self.A = np.log(self.A) - np.log(np.sum(self.A, axis=1, keepdims=True))
    self.B[self.B == 0] = self.epsilon
    self.B = np.log(self.B) - np.log(np.sum(self.B, axis=1, keepdims=True))
    np.savetxt('pi', self.pi)
    np.savetxt('A', self.A)
    np.savetxt('B', self.B)
    print('训练完毕！')
```

```python
def viterbi(self, Obs):
    """
    函数说明： 使用viterbi算法进行解码
    Parameter：Obs - 要解码的文本string
    Return:path - 最可能的隐状态路径

    """
    T = len(Obs)
    delta = np.zeros((T, self.n_tag))  # shape: 观测文本数量*7
    psi = np.zeros((T, self.n_tag))  # shape: 观测文本数量*7
    delta[0] = self.pi[:] + self.B[:, ord(Obs[0])]  # 初始化
    for i in range(1, T):
        temp = delta[i - 1].reshape(self.n_tag, -1) + self.A
        delta[i] = np.max(temp, axis=0)
        delta[i] = delta[i, :] + self.B[:, ord(Obs[i])]
        psi[i] = np.argmax(temp, axis=0)
    path = np.zeros(T)
    path[T - 1] = np.argmax(delta[T - 1])
    for i in range(T - 2, -1, -1):  # 回溯
        path[i] = int(psi[i + 1][int(path[i + 1])])
    return path
```

```
model.predict("张明是河北大学的一名学生")
```

```
张B-PER_|明I-PER_|是O_|河B-ORG_|北I-ORG_|大I-ORG_|学I-ORG_|的O_|一O_|名O_|学O_|生O_|
```

7.4 汉语自动分词系统的评价

准确率(Accuracy)、精确率（Precision）、召回率（Recall）、综合评价指标（F1-Measure）

真实结果	预测结果	
	正例	反例
正例	TP（真正例）	FN（假反例）
反例	FP（假正例）	TN（真反例）

真正例TP（True positive）：　将正例预测为正例（正确）
假反例FN（False negative）：　将正例预测为反例（错误）
假正例FP（False positive）：　将反例预测为正例（错误）
真反例TN（True negative）：　将反例预测为反例（正确）

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

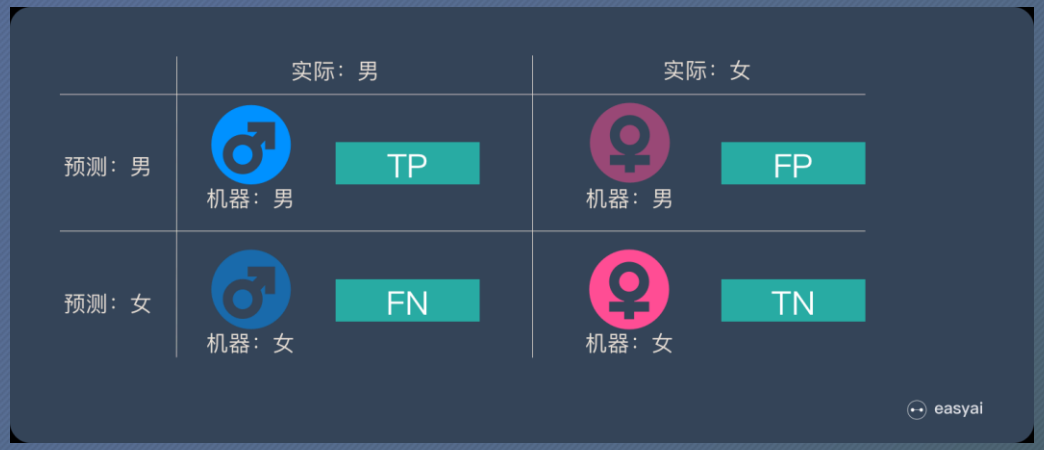
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}Measure = \frac{2 * P * R}{P + R}$$

7.4 汉语自动分词系统的评价

TP = 3； FN = 2； FP =1； TN = 4



https://easyai.tech/ai-definition/accuracy-precision-recall-f1-roc-auc/

准确率 – Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

预测正确的结果 / 总样本

TP = 3； FN = 2； FP =1； TN = 4

$$Accuracy = \frac{3+4}{3+4+1+2} = 0.7$$

精确率（查准率）- Precision

$$Precision = \frac{TP}{TP + FP}$$

实际为正的样本 / 预测为正的样本

精准率 =TP/(TP+FP)

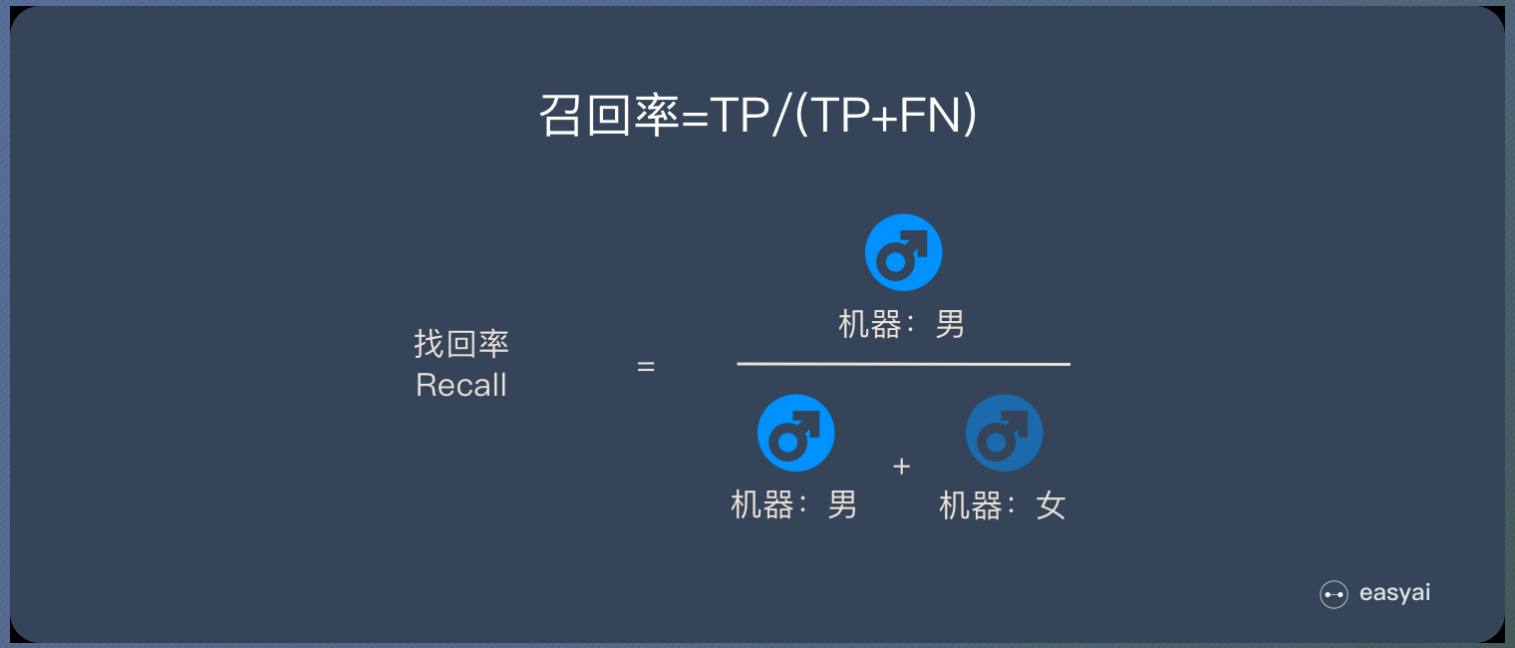
TP = 3； FN = 2； FP =1； TN = 4

$$Precision = \frac{3}{3+1} = 0.75$$

召回率（查全率）- Recall

$$Recall = \frac{TP}{TP + FN}$$

预测为正样本 / 实际为正的样本



TP = 3； FN = 2； FP =1； TN = 4

$$Recall = \frac{3}{3+2} = 0.6$$

F1分数

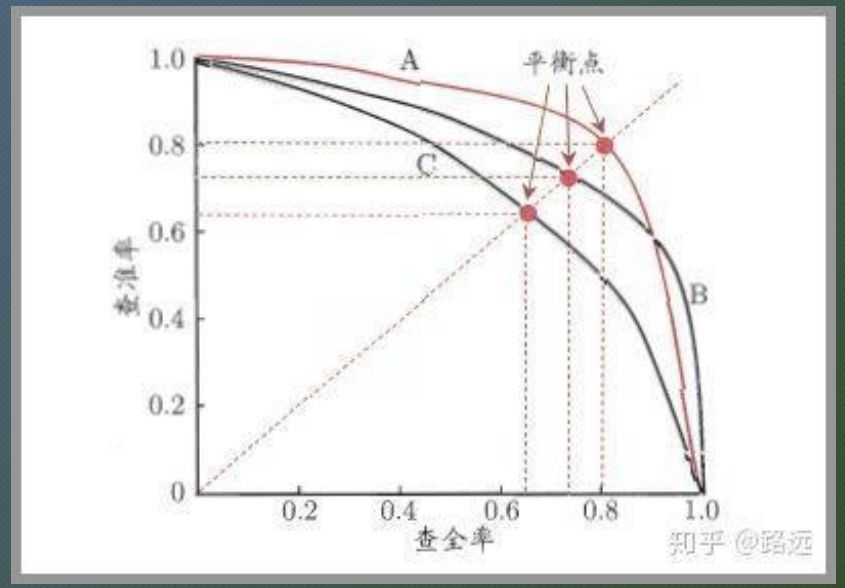
$$F1\text{-}Measure = \frac{2 * P * R}{P + R}$$

精确率（Precision）和召回率（Recall）的关系是「矛盾」关系。为综合两者表现，在两者之间找一个平衡点。

F1=(2×Precision×Recall)/（Precision+Recall)

$$\text{F1 分数} = \frac{2 \times \text{精确率} \times \text{召回率}}{\text{精确率} + \text{召回率}}$$

easyai



$$F1\text{-}Measure = \frac{2*0.75*0.6}{0.75+0.6} = 0.67$$

http://sighan.cs.uchicago.edu/

SIGHAN -汉字特别兴趣小组 国际计算语言学会（ACL）

Special Interest Group for Chinese Language Processing of the Association for Computational Linguistics

国际中文分词评测

International Chinese Word Segmentation Bakeoff

- 第一届2003年，日本札幌举行（Bakeoff 2003）
- 第二届2005年，韩国济州岛举行（Bakeoff 2005）
- 第三届2006年，澳大利亚悉尼举行（Bakeoff 2006）加入了中文命名实体识别评测
- SIGHAN-4
- SIGHAN-5
- SIGHAN-6
- SIGHAN-7 2013 Nagoya名古屋，Japan
- SIGHAN-8 2015 Beijing北京，China
- SIGHAN-9 2017 Taipei台北，China

2010年第一届CIPS-SIGHAN联合会议 北京

2012年第二届CIPS-SIGHAN联合会议 天津

2014年第三届CIPS-SIGHAN联合会议 武汉

7.5 英语形态还原

形态还原：

英语 动词的时态、名词的复数、形容词的比较级最高级 等形态

还原成 词干形式

```
from nltk.stem import PorterStemmer # 波特词干算法

stemmer = PorterStemmer()
print(stemmer.stem('working'))
print(stemmer.stem('worked'))
print(stemmer.stem('buses'))
```

```
work
work
buse
```

```
from nltk.stem import WordNetLemmatizer # 单词变体还原

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize('working'))
print(lemmatizer.lemmatize('working', pos="v"))
print(lemmatizer.lemmatize('worked'))
print(lemmatizer.lemmatize('worked', pos="v"))
print(lemmatizer.lemmatize('buses'))
```

```
working
work
worked
work
bus
```

7.6 词性标注（Part-of-Speech tagging 或 POS tagging)

定义:
根据一个词在某个特定句子中的上下文，为这个词标注正确的词性。

方法:
基于规则的方法
基于统计的方法

7.6 词性标注

7.6.1 词性标记集

7.6.2 基于规则的词性标注方法

7.6.3 基于统计的词性标注方法

7.6.1 词性标记集 POS Tagset

英文：

- Brown标注集、
- Penn Tree Bank标注集、
- CLAWS c5标注集

中文：

- 《PFR人民日报标注语料库》词性编码表
- 《现代汉语语料库加工规范—词语切分与词性标注》词性标记
- 计算所 ICTCLAS 3.0汉语词性标记集
- HanLP词性标注集
- 结巴分词中出现的类型

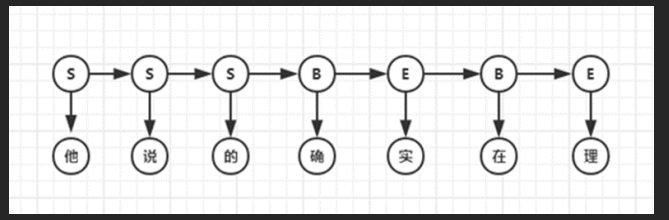
7.6.2 基于规则的词性标注方法

基本思想:

按兼类词（拥有多种可能词性的词）搭配关系和上下文语境建立词性消歧规则。

早期的规则一般由人工编写。
随着语料库规模的逐步增大，人工方式显然不现实，
于是提出基于机器学习的规则自动提取方法。

7.6.3 基于统计的词性标注方法



使用 HMM 自动分词 + 词性标注

自动分词：

给定一个字的序列，找出最可能的标签序列。

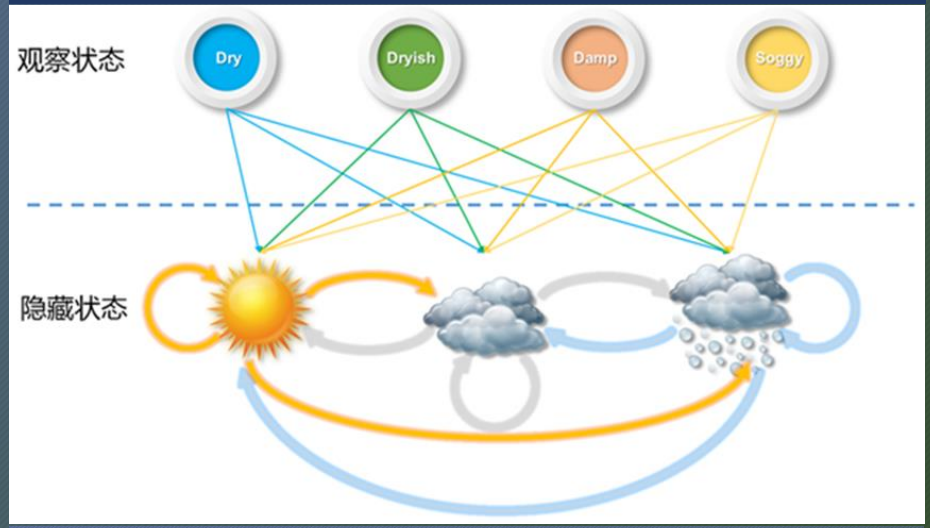
观察状态– 句子；隐藏状态 – 标签序列

词性标注：

给定一个词的序列，找出最可能的词性序列。

观察状态– 词语；隐藏状态 – 词性标记

观察状态、隐藏状态
初始状态概率、状态转移概率、发射概率



2. 给定模型和观测序列，如何找到最匹配的状态序列?
   - 如何根据观测序列推断出隐藏的模型状态?
   - 找到一个隐藏状态的序列使得这个序列产生一个可观察状态序列的概率最大(解码)
   - 维特比算法(Viterbi Algorithm)

http://www.hankcs.com/nlp/hmm-and-segmentation-tagging-named-entity-recognition.html

HMM词性标注

```python
def viterbi(obs, states, start_p, trans_p, emit_p):
    """
    :param obs: 可见序列
    :param states: 隐状态
    :param start_p: 开始概率
    :param trans_p: 转换概率
    :param emit_p: 发射概率
    :return: 序列+概率
    """
    path = {}
    V = [{}]  # 记录第几次的概率
    for state in states:
        V[0][state] = start_p[state] * emit_p[state].get(obs[0], 0)
        path[state] = [state]
    for n in range(1, len(obs)):
        V.append({})
        newpath = {}
        for k in states:
            pp,pat=max([(V[n - 1][j] * trans_p[j].get(k,0) * emit_p[k].get(obs[n], 0) ,j )for j in states])
            V[n][k] = pp
            newpath[k] = path[pat] + [k]
            # path[k] = path[pat] + [k]#不能提起变，，后面迭代好会用到！
        path=newpath
    (prob, state) = max([(V[len(obs) - 1][y], y) for y in states])
    return prob, path[state]
```

```python
test_strs=[u"你们 站立 在",
           u"我 站 在 北京 天安门 上 大声 歌唱",
           u"请 大家 坐下 喝茶",
           u"你 的 名字 是 什么",
           u"今天 天气 特别 好"]

for li in range(0,len(test_strs)):
    test_strs[li]=test_strs[li].split()
for li in test_strs:
    p,out_list=viterbi(li,state_list,start_c,trans_c,emit_c)
    print(li)
    print(out_list)
    print('--------------------------------------------------')
```

```
['你们', '站立', '在']
['r', 'v', 'p']
--------------------------------------------------
['我', '站', '在', '北京', '天安门', '上', '大声', '歌唱']
['r', 'v', 'p', 'ns', 'ns', 'f', 'd', 'v']
--------------------------------------------------
['请', '大家', '坐下', '喝茶']
['v', 'r', 'v', 'v']
--------------------------------------------------
['你', '的', '名字', '是', '什么']
['r', 'u', 'n', 'v', 'r']
--------------------------------------------------
['今天', '天气', '特别', '好']
['t', 'n', 'd', 'a']
--------------------------------------------------
```

基于HMM的分词，词性标注，命名实体识别

liuhuanyong Update README.md 84f11c7 on 14 Apr 2018 2 commits

CRF	创建NLP项目	2 years ago
HMM	创建NLP项目	2 years ago
README.md	Update README.md	2 years ago
__init__.py	创建NLP项目	2 years ago
huannlp.py	创建NLP项目	2 years ago
test.py	创建NLP项目	2 years ago

README.md

HuanNLP

self implement of NLP toolkit 个人实现NLP汉语自然语言处理组件，提供基于HMM与CRF的分词，词性标注，命名实体识别接口，提供基于CRF的依存句法接口。

使用简介

引入

import nlp nlp = huannlp.HuanNLP('HMM') 或者 nlp = huannlp.HuanNLP('CRF') text = "刘焕勇硕士毕业于北京语言大学，目前在中国科学院软件研究所工作"

https://github.com/liuhuanyong/HuanNLP

前沿发展

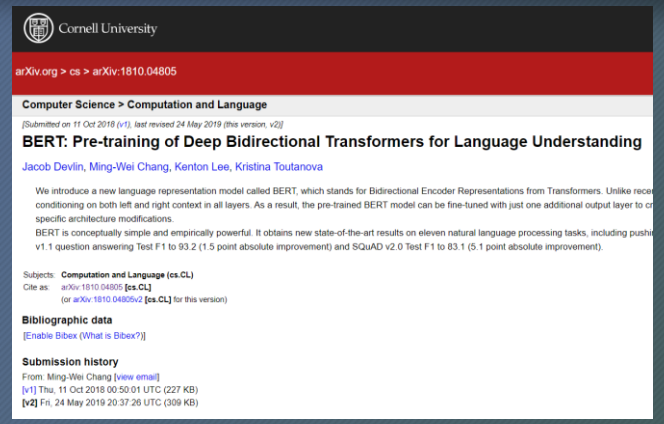
最大熵马尔科夫模型 Maximum Entropy Markov Model， MEMM
条件随机场 conditional random field，CRF
LSTM的全称是Long Short Term Memory，它是RNN（Recurrent Neural Network）的一种

HMM，CRF，BiLSTM，**BiLSTM+CRF**

https://zhuanlan.zhihu.com/p/100969186?utm_source=com.gzqwkj.cshu

早期方法	概率模型方法	深度学习方法	近期研究方向
基于规则 基于字典	HMM MEMM CRF	RNN-CRF CNN-CRF LSTM-CRF	Attention Transfer Learning Semi-Supervised Low-Resource ⋮

https://zhuanlan.zhihu.com/p/112340282



https://arxiv.org/abs/1810.04805

CAN-NER: Convolutional Attention Network for Chinese Named Entity Recognition

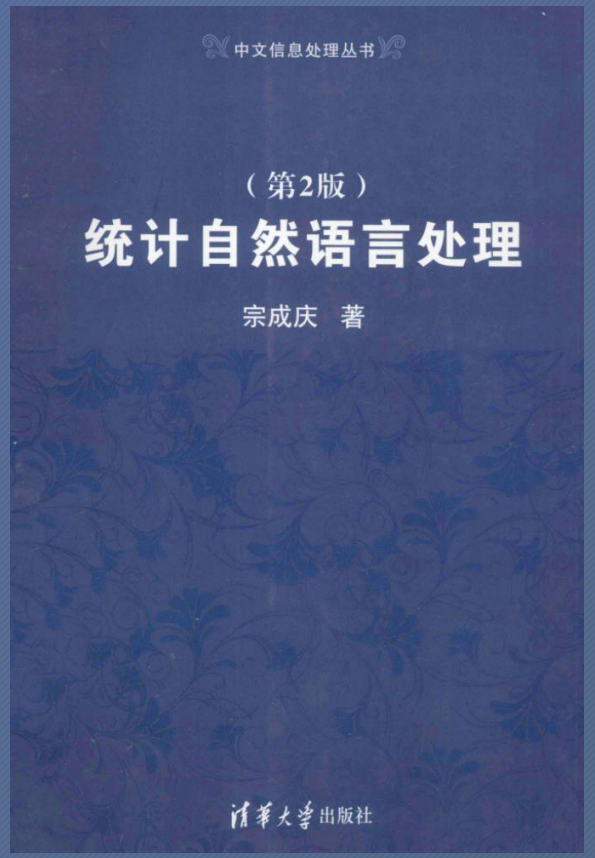
Yuying Zhu*
Nankai University
Tianjin, China
yuyzhu@mail.nankai.edu.cn

Guoxin Wang
Microsoft Research Asia
Beijing, China
guow@microsoft.com

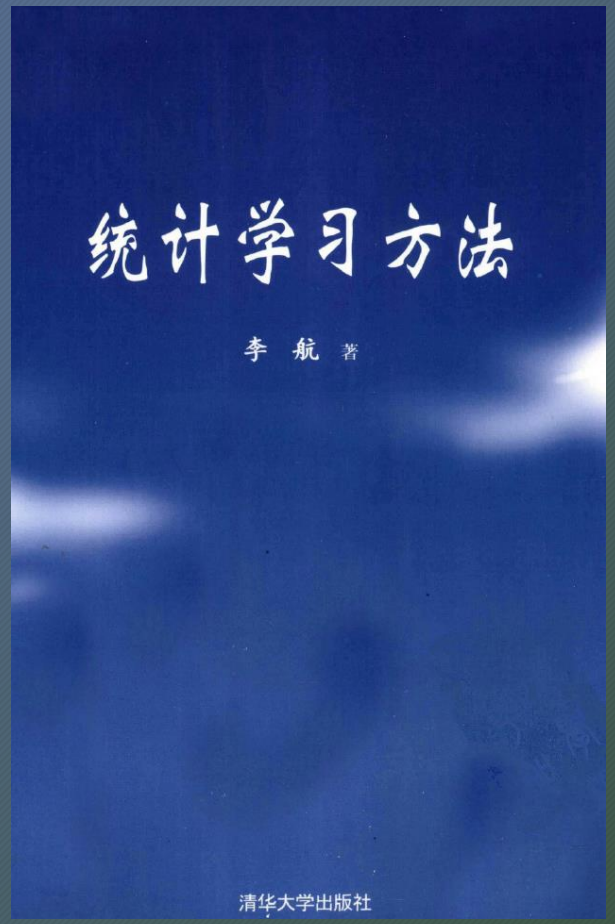
https://www.aclweb.org/anthology/N19-1342.pdf

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com



- 第7章 自动分词、命名实体识别与词性标注
- 第8章 句法分析
- 第9章 语义分析
- 第10章 篇章分析
- 第11章 统计机器翻译
- 第12章 语音翻译
- 第13章 文本分类与情感分类
- 第14章 信息检索与问答系统
- 第15章 自动文摘与信息抽取
- 第16章 口语信息处理与人机对话系统



THE END