自然语言处理基础

# 隐马尔可夫模型

## 原理

### 两个基本假设

1. 齐次马尔可夫性

任一时刻的状态只与前一时刻的状态有关。注意：在计算最佳状态序列时都使用同一个状态转移概率。

1. 观测独立性假设

观测值之间相互独立，这个假设与朴素贝叶斯模型是一样的。

1. 两个假设导致的缺陷

两个基本假设就是隐马尔可夫模型解决序列标注问题时的缺陷，首先任一时刻的隐含状态只能与前一时刻相关，其次认为各观测值之间相互独立。按照何晗的《自然语言处理入门》第4章所述，HMM应用于分词的效果并不是特别理想，特别是很多在词典中的词没有被正确识别，也就是说HMM对于分词消岐并不理想，作者的结论是HMM的分词效果甚至不如词典分词。因此，学习隐马尔可夫模型就是在为学习条件随机场模型打基础，我们主要学习的是隐马尔可夫模型预测时的维特比算法。同时，HanLP的人名、地名、机构名识别使用了基于隐马的角色标注，因此，学习隐马三要素以及训练过程对于理解HanLP识别人名、地名、机构名的原理有帮助。

### 三要素与训练

1. 初始状态概率向量PI

以分词为例，就是统计状态B、M、E、S的出现频率。其中B表示分词后词语的首字，M表示分词后词语的非首非尾字（如果词语字数>=3），E表示词语的尾字，S表示单字成词。比如“北京/ 信息/ 科技/ 大学”，对应的状态序列为“BEBEBEBE”。以状态B为例说明计算频率的方法，统计状态B在训练语料中的出现次数#B，统计训练语料中所有状态总数(#B+#M+#E+#S)，则PI(B)=#B/(#B+#M+#E+#S)。按照类似的步骤可以统计出另外3种状态的频率。各状态频率统计完成后，我们可以得到下边这样的初始状态概率向量。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 状态符号 | B | M | E | S |
| 频率值 | PI(B) | PI(M) | PI(E) | PI(S) |

1. 观测概率（发射概率）矩阵B

依然以分词为例，发射概率就是一个状态可能对应的汉字的概率分布，就是下边的概率矩阵。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | B | M | E | S |
| 字1 |  |  |  |  |
| 字2 |  |  |  |  |
| … |  |  |  |  |
| 大 | B(大->B) |  |  |  |
| … |  |  |  |  |
| 字m |  |  |  |  |

以“B->大”这个元素值的计算为例说明观测概率矩阵中各元素的计算方法。我们需要统计出“大”被标记为“B”的次数，记为#(大->B)，以及#(大->B)+#(大->M)+ #(大->E) +#(大->S)，于是B(大->B)= #(大->B)/( #(大->B)+#(大->M)+ #(大->E) +#(大->S))。

1. 转移概率矩阵A

转移概率矩阵顾名思义，就是说从一种转态到另一种状态的概率，就是下边的概率矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | B | M | E | S |
| B |  | A(B->M) |  |  |
| M |  |  |  |  |
| E |  |  |  |  |
| S |  |  |  |  |

以元素“B->M”的计算为例说明状态转移概率矩阵中各元素值计算。我们需要统计出状态“B”后边接状态“M”的次数，记为#(B->M)，以及#(B->B)+ #(B->M)+ #(B->E)+ #(B->S)，于是A(B->M)= #(B->M)/( #(B->B)+ #(B->M)+ #(B->E)+ #(B->S))。

1. 有监督训练

有监督训练过程就是在给定训练语料的情况下，统计上述三个概率矩阵各元素的值。

### 维特比预测

1. 依据动态规划原理，最优路径应具有的性质是什么？

如果最优路径在时刻通过结点，那么这一路径从结点到终点的部分路径，对于从到的所有可能部分路径来说是最优的。

1. 简述维特比算法

依据动态规划原理，我们只需从时刻开始，递推地计算在时刻状态为的各条部分路径的最大概率，直至得到时刻状态为的各条路径的最大概率。时刻的最大概率即为最优路径的概率，最优路径的终结点也同时得到。之后，为了找出最优路径的各个结点，从终结点开始，由后向前逐步求得结点，得到最优路径。这就是维特比算法。

1. 维特比算法

首先导入两个变量和。**定义在时刻状态为的所有单个路径中概率最大值为**

这里表示取到的某一个状态，状态有种。

由定义可得变量的递推公式：

定义在时刻状态为的所有单个路径中概率最大的路径的第个**结点为**

维特比算法：

|  |
| --- |
| 输入：模型和观测  输出：最优路径   1. 初始化 2. 递推   对   1. 终止 2. 最优路径回溯   对  求得最优路径 |

**例：**假设有4个盒子，每个盒子里都装有红白两种颜色的球，盒子里的红白球数由下表列出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 盒子编号 | 1 | 2 | 3 | 4 |
| 红球数 | 5 | 3 | 6 | 8 |
| 白球数 | 5 | 7 | 4 | 2 |

按照下面的方法抽球，产生一个球的颜色的观测序列：

开始，从4个盒子里以等概率随机选取1个盒子，从这个盒子里随机抽取出1个球，记录其颜色后，放回；

然后，从当前盒子随机转移到下一个盒子，规则是：如果当前盒子是盒子1，那么下一盒子一定是盒子2，如果当前盒子是盒子2或3，那么分别以概率0.4和0.6转移到左边或右边的盒子，如果当前是盒子4，那么各以0.5的概率停留在盒子4或转移到盒子3；确定转移的盒子后，再从这个盒子里随机抽出1个球，记录其颜色；放回；

如此下去，重复进行5次，得到一个球的颜色的观测序列：O={红，红，白，白，红}。

在这个过程中，观察者只能观测到球的颜色的序列，观察不到球是从哪个盒子取出的，即观测不到盒子的序列。

**问题1：为什么说这个例子是一个隐马尔可夫模型？**

在这个例子中有两个随机序列，一个盒子的序列（状态序列），一个是球的颜色的观测序列（观测序列）。前者是隐藏的，只有后者是可观测的，这是一个隐马尔可夫模型的例子。

**问题2：根据所给条件，写出状态集合、观测集合、序列长度以及模型的三要素。**

盒子对应状态，状态的集合是Q={盒子1，盒子2，盒子3，盒子4}，N=4

球的颜色对应观测，观测的集合是V={红，白}，M=2

状态序列和观测序列长度T=5。

由于是等概率随机选取1个盒子，因此，初始状态概率分布为：。

根据从当前盒子随机转移到下一个盒子的规则，我们可以得到状态转移概率分布为

根据盒子里装球情况的表，我们可以得到观测概率分布为

注意：这个例子直接给出了三要素矩阵的各元素值，而不是通过训练集统计得到。

**问题3：假设模型三要素为，其中**

**，，**

**设T=3，O=(红，白，红)，试用维特比算法求解最优路径。**

解：

1. 初始化
2. 递推

其中：

其中：

其中：

其中：

其中：

其中：

1. 终止
2. 最优路径回溯

求得最优路径。

## 分词示例

## 应用论文汇总

张华平-基于角色标注的中国人名自动识别研究

余鸿魁-基于层叠隐马尔可夫模型的中文命名实体识别

余鸿魁-基于角色标注的中文机构名识别

# 最大熵模型

## 原理

### 基本思想

最大熵方法是E.T.Jaynes于1957年提出的，1992年首次被Della Pietra运用于自然语言处理建模，并于1996年提出了一种利用最大熵方法将上下文中多信源特征集成起来建立语言模型的方法。最大熵原理指出，当我们需要对一个随机事件的概率分布进行预测时，我们的预测应当满足全部已知的条件，而对未知的情况不要做任何主观假设。在这种情况下，概率分布最均匀，预测的风险最小。因为这时概率分布的信息熵最大，所以人们把这种模型称为“最大熵模型”。

设随机过程所有的输出值构成有限集，对于每个输出，其产生均受上下文信息的影响和约束。已知与有关的所有上下文信息组成的集合为，则模型的目标是：给定上下文，计算输出的条件概率，。

进行最大熵分类的步骤是建立一个模型，捕捉单独的联合特征的频率，而不必作任何无根据的假设。这里的联合特征指的是输入与输出构成的特征对。举一个例子将有助于说明这一原则。

假设有一个词义消岐任务，要求从10个可能词义（标签从A-J）中为给定的某个词word找出一个最合适的词义，已知（1）词义A被选出的频率55%；（2）单词“up”出现在word附近上下文中的频率为10%；（3）当“up”出现在word附近上下文时有80%的可能性使用词意A或C。则下边的联合概率分布（注：这里只截取出有关“up”的一部分）是否合适？为什么？

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 预测词义  up出现？ | A | B | C | D | E | F | G | H | I | J |
| +up | 5.1% | 0.25% | 2.9% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% |
| -up | 49.9% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% |

其中+up表示word上下文中出现了单词“up”，-up表示word上下文中没有出现单词“up”。每1列表示词word可能具有的词义，这里有A-J共10种词义。表格中的值表示的是p(word词义|给定上下文中是否出现词up)。显然表格第A列概率之和为55%，满足被告知的条件（1）。表格第1行所有概率之和为10%，满足了被告知的条件（2）。表格第1行，A和C两列概率之和占该行概率总和的80%，满足了被告知的条件（3）。此外其他概率值基本相等，呈现均匀分布。

纵观这个例子，模型被限制在已知分布上的同时熵最大。特别的，对于每个联合特征（也就是被告知的条件），最大熵模型在训练集上计算该特征的“经验频率”（也就是被告知条件中的概率），然后搜索能使熵最大的分布。

给出最大熵模型的定义：假设满足所有约束条件的模型集合为

定义在条件概率分布上的条件熵为

则模型集合中条件熵最大的模型称为最大熵模型。

其中，表示某1个特征函数，描述输入和输出之间的某一个事实。其定义是

是特征函数关于经验分布的期望值，

表示训练数据中样本出现的次数，表示训练数据样本总数。

是特征函数关于模型与经验分布的期望值，

表示训练数据中输入特征集出现的次数。表示训练数据样本总数。

### 模型训练

1. **IIS数值计算方法训练最大熵模型**

|  |
| --- |
| 输入：特征函数，经验分布，模型  输出：最优参数值，最优模型   1. 对所有，取初值 2. 对每一： 3. ，   对于一个确定的训练集，为一个常数，因此，   1. 更新值： 2. 如果不是所有都收敛，重复步（2） |

上边的算法中省略了最大迭代次数的限制。

下边的代码演示IIS算法应用于词性标注

1. **BFGS数值计算方法训练最大熵模型**

|  |
| --- |
| 输入：特征函数，经验分布，目标函数，梯度，精度要求。  输出：最优参数值，最优模型   1. 选定初始点，取为正定对称矩阵，置 2. 计算，若，则停止计算，得，否则转（3） 3. 由，求出 4. 一维搜索：求使得 5. 置 6. 计算，若，则停止计算，得，否则按下式求出，其中， 7. 置，转（3） |

1. **DFP数值计算方法训练最大熵模型**

|  |
| --- |
| 输入：特征函数，经验分布，目标函数，梯度，精度要求。  输出：最优参数值，最优模型   1. 选定初始点，取为正定对称矩阵，置 2. 计算，若，则停止计算，得，否则转（3） 3. 置 4. 一维搜索：求使得 5. 置 6. 计算，若，则停止计算，得，否则按下式求出，其中， 7. 置，转（3） |

1. **各训练算法的比较**

Steven Bird等在《Natual Language Processing with Python》一书中指出，当训练最大熵模型时，应避免使用广义迭代缩放GIS或改进的迭代缩放IIS，这两种都比共轭梯度CG和BFGS优化方法慢很多。

### 模型预测

通过引入拉格朗日乘子，可以将最大熵模型定义所描述的约束最优化原始问题转换为无约束最优化的对偶问题，求解得到的极值就是最大熵模型的预测表达式。

其中，

这里，为输入，为输出，为权值向量，为任意实值特征函数。

## 词性标注示例

这里，我们用IIS实现的最大熵模型训练一个词性标注器。

|  |
| --- |
| # encoding: utf-8  """  最大熵词形标注示例  """  from collections import defaultdict  import math  import codecs  class MaxEnt(object):      def \_\_init\_\_(self):          # 存储特征函数fi(x,y)关于经验分布p(x,y)的期望值 Ep\_(fi)          self.feats = defaultdict(int)          # 文本经过特征抽取后建成的训练集每一项用一个事件表示(词性标记,特征1,特征2,特征3)，          self.trainset = []          # 词性标记集，表示Y可取的值的集合，比如名词、动词、形容词...          self.labels = set()      def generate\_events(self, text, train\_flag=False):          """          text: 以空格分隔的文本          train\_flag: train\_flag=True时，要求text必须标注好词语词性          功能：对输入的一句话，抽取出事件列表          """          # 定义事件序列，注意输入的1行文本可以抽取出多个事件，事件序列称为数据集或许更合适          event\_li = []          # 分词，要求输入的字符串中词语之间以空白分隔，对于训练集在词语后还要加词性标记，格式为word/pos          word\_li = text.split()          # 分离词语和词性          if train\_flag:              # 对于含有词性标记的词语序列，分离词语和词性：[(word1,pos1),(word2,pos2),...]              word\_li = [tuple(w.split(u'/')) for w in word\_li if len(w.split(u'/')) == 2]          else:              # 对于没有词性标记的词语序列，添加词性"x\_pos"：[(word1,'x\_pos'),(word2,'x\_pos'),...]              word\_li = [(w, u'x\_pos') for w in word\_li]          # 为词语序列添加头元素和尾元素，便于后续抽取事件          word\_li = [(u'pre1', u'pre1\_pos')] + word\_li + [(u'pro1', u'pro1\_pos')]          # 遍历中心词抽取1个事件，每个事件由1个词性标记和多个特征项构成          for i in range(1, len(word\_li) - 1):              # 特征函数1 中心词              fea\_1 = word\_li[i][0]              # 特征函数2 前一个词              fea\_2 = word\_li[i - 1][0]              # 特征函数3 后一个词              fea\_3 = word\_li[i + 1][0]              # 词性y              y = word\_li[i][1]              # 构建1个事件，注意1个事件由3个特征项构成，同一事件中的3个特征项共享1个输出标记y              # 因此1个事件对应3个特征函数或者叫做联合特征：f1(fea\_1,y),f2(fea\_2,y),f3(fea\_3,y)              # 注意这里并没有区分3个特征的先后顺序              fields = [y, fea\_3, fea\_2, fea\_1]              # 将事件添加到事件序列              event\_li.append(fields)          # 返回事件序列          return event\_li      def load\_data(self, file):          """          file: 训练文件路径和文件名          功能：读取文本数据构建训练集          """          with codecs.open('../data/199801.txt', 'rb', 'utf8', 'ignore') as infile:              for line\_ser, line in enumerate(infile):                  if line\_ser >= 100:                      break                  line = line.strip()                  if line:                      # 抽取事件序列                      events\_li = self.generate\_events(line, train\_flag=True)                      # 遍历每个事件，对每个事件构建3个特征函数或者叫做联合特征fi(x,y)                      for event in events\_li:                          # 第1列是输出词性标记y                          label = event[0]                          # 更新词性标记                          self.labels.add(label)                          # 对联合特征(词性标记，特征)计数，以便后续计算P(X,Y)的经验分布                          # 注意一个联合特征为(y,x)的组合，x表示一个特征值，并且丢失掉了x的位置信息                          for f in set(event[1:]):                              self.feats[(label, f)] += 1                          # 将事件添加到训练集                          self.trainset.append(event)      def \_initparams(self):          """          功能：根据load\_data方法加载进来的事件集（训练集）计算Ep\_(fi)，并且给联合特征(y,x)编号          """          # 训练集中抽取的事件总数          self.N = len(self.trainset)          # 1个事件对应的特征数量，取值为遍历训练集中每个事件抽取出的最大特征数量          self.M = max([len(record) - 1 for record in self.trainset])          # 计算每一种特征函数fi(x,y)关于经验分布P\_(X,Y)的期望值，指的是《统计学习方法》P82页最下边的式子          self.ep\_ = [0.0] \* len(self.feats)          # 计算Ep\_(fi)，并且用self.feats存储联合特征的索引          for i, union\_feature in enumerate(self.feats):              # 第i个特征函数的经验值=联合特征(y,x)出现次数/训练集中事件总数。              self.ep\_[i] = float(self.feats[union\_feature]) / float(self.N)              # 记录特征函数索引，经过这一步后self.feats中记录的是特征函数的索引值，以便后续预测时读取特征函数对应的权值              self.feats[union\_feature] = i          # 权重序列初始化为全0，每一个特征函数都对应一个权重值，就是所求的系数          self.w = [0.0] \* len(self.feats)          self.lastw = self.w      def Ep(self):          """          计算Ep(fi) = sum (x,y) [p\_(x)\*p(y|X)fi(x,y)] 即《统计学习方法》P83页最上边的式子          此过程涉及调用模型预测方法self.calprob，预测输出p(y|X)          经验分布p\_(x)刚好等于 1/训练集样本数，实际它的公式应该为p\_(x)=v(X=x)/N          """          # 期望值的个数等于特征函数的个数          ep = [0.0] \* len(self.feats)          # 遍历每个事件          for record in self.trainset:              # 分离出特征集              features = record[1:]              # 计算特征集的预测输出P(y|X)              prob\_li = self.calprob(features)              # 遍历每一个特征值              for x in features:                  # 遍历每个标记和其对应的概率值                  for prob, y in prob\_li:                      # 由标记和1个特征构建出1个特征函数fi(y,x)                      if (y, x) in self.feats:                          # 读取出特征函数的索引值                          idx = self.feats[(y, x)]                          # p\_(x)的经验分布计算 p(x)=1/N                          ep[idx] += prob \* (1.0 / self.N)          return ep      def \_convergence(self, lastw, w):          """          lastw:上一次迭代的系数向量          w: 本次迭代的系数向量          返回：True表示已经收敛；False表示未收敛          """          for w1, w2 in zip(lastw, w):              if abs(w1 - w2) >= 0.01:                  return False          return True      #  训练过程      def train(self, max\_iter=1000):          """          最大熵分词模型训练过程          :param max\_iter: 模型训练的最大迭代次数          :return:无          """          # 初始化模型参数          self.\_initparams()          # 开始迭代          for i in range(max\_iter):              print('iter %d ...' % (i + 1))              # 计算特征函数关于模型P(Y|X)和经验分布P(X)的期望值序列，每个特征函数对应一个值              ep = self.Ep()              # 保存当前的权值向量              self.lastw = self.w[:]              # 计算delta并更新权值向量              for i, w in enumerate(self.w):                  # 计算delta                  delta = 1.0 / self.M \* math.log(self.ep\_[i] / ep[i])                  # 更新权值                  self.w[i] += delta              # 检查是否收敛              if self.\_convergence(self.lastw, self.w):                  break        def probwgt(self, features, label):          """          features: 特征集          label: 候选标记          返回: P(y|X)的概率值          功能：给定(X,y)计算条件概率P(y|X)          """          wgt = 0.0          # 遍历每个特征f          for f in features:              # 如果训练集中见过特征函数(y,f)则读出权重              if (label, f) in self.feats:                  # 累加权重                  wgt += self.w[self.feats[(label, f)]]          # 取exp(权重和)作为预测值，这里没有进行归一化          return math.exp(wgt)      # 最大熵模型的预测函数P(y|x)，输入为1个事件中的特征x，输出为标记集中每种标记对应的归一化的概率P(y1|x),P(y2|x),...      # 返回[(p1, y1),(p2,y2),...]      def calprob(self, features):          """          features: 特征集          返回：预测为标记集self.labels中每种标记的概率          功能：给定特征集X,输出预测为标记集中每种标记的概率P(y|X)          """          # 计算标记集中每种标记的条件概率          wgts = [(self.probwgt(features, l), l) for l in self.labels]          # 使每个概率值相加后等于1          Z = sum([w for w, l in wgts])          prob = [(w / Z, l) for w, l in wgts]          return prob      def predict(self, text):          """          text: 输入的经过分词但未经词性标注的文本          返回：词性标注后的文本          功能：对经过分词的文本进行词性标注          """          word\_li = []          events\_li = self.generate\_events(text)          for features in events\_li:              prob = self.calprob(features)              prob.sort(reverse=True)              word\_li.append(u'%s/%s' % (features[1], prob[0]))          return word\_li  if \_\_name\_\_ == "\_\_main\_\_":      maxent = MaxEnt()      maxent.load\_data("../data/199801.txt")      maxent.train(50)      word\_li = maxent.predict("中国 政府 将 继续 坚持 奉行 独立自主 的 和平 外交 政策 。")      print(word\_li) |

## 应用论文汇总

最大熵模型除了可以应用于词性标注，也同样可以应用于句法分析、文本分类、问题回答等应用场景。

## 实现库

### HanLP实现

* GIS训练算法<https://github.com/hankcs/MaxEnt>
* IIS训练算法<https://github.com/hankcs/maxent_iis>

### 张乐实现

* <https://github.com/lzhang10/maxent>

### NLTK实现

* <http://www.nltk.org/api/nltk.classify.html#nltk.classify.maxent.MaxentClassifier>
* <http://www.nltk.org/_modules/nltk/classify/maxent.html#MaxentClassifier>

NLTK实现了'GIS', 'IIS', 'MEGAM', 'TADM'4种学习算法，'MEGAM'需要安装额外库。

使用示例

|  |
| --- |
| *# coding:utf-8 """ NLTK的最大熵模型实现词性标注 """* **import** nltk **import** codecs   *# 事件生成器，1个分词后的句子可以产生多个事件* **def** generate\_events(word\_li):  events = []  *# 为词语序列添加头元素和尾元素，便于后续抽取事件* word\_li = [(**u'pre1'**, **u'pre1\_pos'**)] + word\_li + [(**u'pro1'**, **u'pro1\_pos'**)]   *# 每个中心词抽取1个event，每个event由1个词性标记和多个特征项构成* **for** i **in** range(1, len(word\_li) - 1):  *# 定义特征词典* features\_dict = dict()  *# 特征函数a 中心词* features\_dict[**'fea\_1'**] = word\_li[i][0]  *# 特征函数b 前一个词* features\_dict[**'fea\_2'**] = fea\_2 = word\_li[i - 1][0]  *# 特征函数d 下一个词* features\_dict[**'fea\_4'**] = word\_li[i + 1][0]  *# 标记* label = word\_li[i][1]  *# 添加一个事件* events.append((features\_dict, label))   **return** events   *# 加载数据，生成事件集，返回* **def** load\_data(file\_name):  data\_set = []  **with** codecs.open(file\_name, **'rb'**, **'utf8'**, **'ignore'**) **as** infile:  **for** line\_ser, line **in** enumerate(infile):  **if** line\_ser >= 100:  **break** line = line.strip()  **if** line:  word\_li = line.split()  word\_li = [tuple(w.split(**u'/'**)) **for** w **in** word\_li **if** len(w.split(**u'/'**)) == 2]  *# 生成事件并更新到data\_set* data\_set.extend(generate\_events(word\_li))  print(**"抽取出 %d 个事件"** % len(data\_set))  **return** data\_set   *# 标注词性* **def** pos\_tag(classifier, line):  new\_word\_li = []  word\_li = line.split()  word\_li = [(w, **u'x\_pos'**) **for** w **in** word\_li]  events\_li = generate\_events(word\_li)  **for** i, (features, label) **in** enumerate(events\_li):  predict\_pos = classifier.classify(features)  new\_word\_li.append(word\_li[i][0]+**"/"**+predict\_pos)  **return** new\_word\_li   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  *# 抽取特征构建训练和测试集* data\_set = load\_data(**'data/199801.txt'**)  train\_size = int(len(data\_set)\*0.8)  test\_size = int(len(data\_set)\*0.2)  train\_set = data\_set[:train\_size]  test\_set = data\_set[-test\_size:]  print(**"训练集事件数="**, len(train\_set))  print(**"测试集事件数="**, len(test\_set))  *# GIS学习算法的最大熵模型* classifier\_gis = nltk.classify.maxent.MaxentClassifier.train(train\_set, trace=2, algorithm=**'gis'**, max\_iter=10)  print(**"GIS模型准确率= "**, nltk.classify.accuracy(classifier\_gis, test\_set))  print(**"GIS模型测试："**)  print(pos\_tag(classifier\_gis, **"中国 政府 将 继续 坚持 奉行 独立自主 的 和平 外交 政策 。"**))  *# IIS学习算法的最大熵模型* classifier\_iis = nltk.classify.maxent.MaxentClassifier.train(train\_set, trace=2, algorithm=**'iis'**, max\_iter=10)  print(**"IIS模型准确率= "**, nltk.classify.accuracy(classifier\_iis, test\_set))  print(**"IIS模型测试："**)  print(pos\_tag(classifier\_iis, **"中国 政府 将 继续 坚持 奉行 独立自主 的 和平 外交 政策 。"**)) |

运行结果

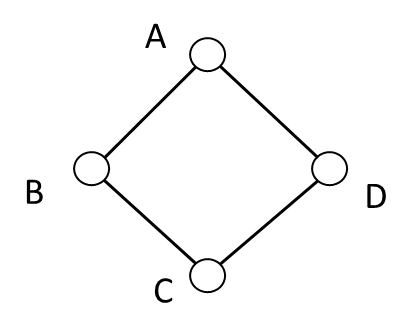
|  |
| --- |
| 抽取出 6389 个事件  训练集事件数= 5111  测试集事件数= 1277  ==> Training (10 iterations)  GIS模型准确率= 0.6664056382145653  GIS模型测试：  ['中国/ns', '政府/n', '将/d', '继续/v', '坚持/v', '奉行/v', '独立自主/i', '的/u', '和平/n', '外交/n', '政策/n', '。/w']  ==> Training (10 iterations)  IIS模型准确率= 0.653876272513704  IIS模型测试：  ['中国/ns', '政府/n', '将/d', '继续/v', '坚持/v', '奉行/v', '独立自主/i', '的/u', '和平/n', '外交/n', '政策/n', '。/w'] |

# 条件随机场模型

## 数学原理

### 概率无向图模型——马尔可夫网络

（1）图的概率意义

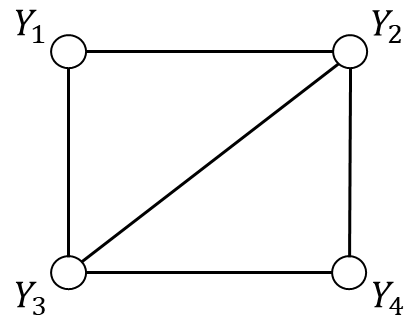


上图就是一个马尔可夫网络，图中的每个节点表示一个随机变量，相连节点表示两个随机变量相关，不相连的节点表示两个随机变量独立，于是可以写出下边的两个等式

（2）团：一个图中的任何两个节点之间都有边相连时，这个图就是一个团。

（3）因子函数（势函数或者能量函数）：定义在团c上的势函数。

（4）图的概率计算



一个图的所有随机变量的联合概率分布等于图中所有团上的势函数的乘积，即。

### HMM等生成模型存在的问题

（一）计算繁琐

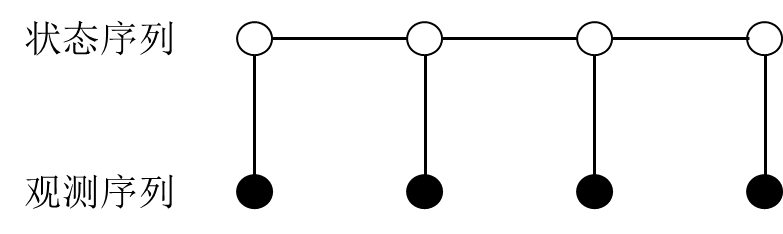
1. HMM是生成模型，需要先计算联合概率分布也就是发射概率矩阵。

2. 观测值是符号也就是离散值，因此不适用取值连续的观测。

3.必须计算每个状态发射所有可能观测符号的概率。

（二）观测符号是原子的

每个观测符号的概率与其前后的观测符号无关，如下图所示



（三）很难加入字词顺序以外的特征

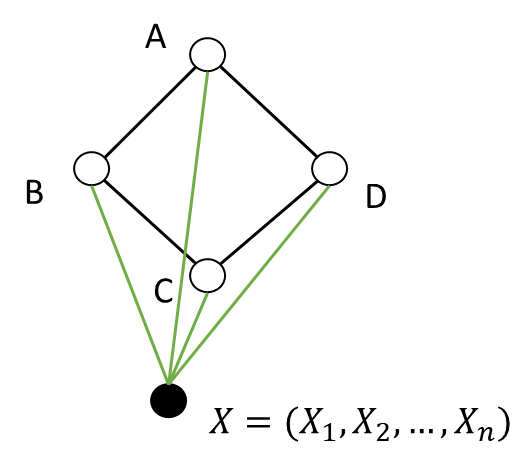
以实体识别为例，我们除了使用词语串作为特征以外，无法加入词性串特征。

（四）标注偏置问题

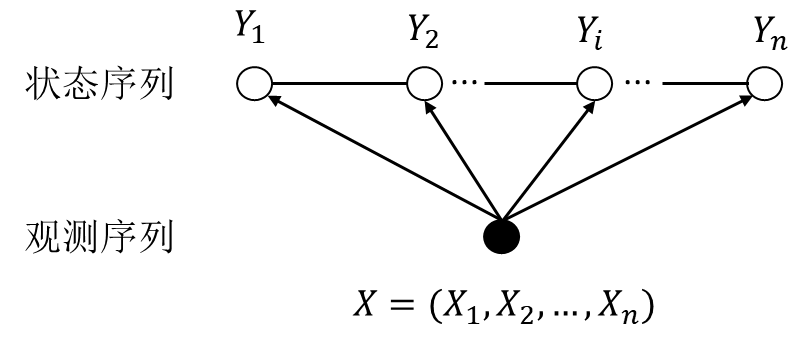
假设有两个相似的观测序列ri和ro，它们的状态序列分别为12和34，如果训练集中ri出现的次数远大于ro出现的次数，则预测观测序列ro对应的状态序列时，很容易将状态序列错误的预测为12。

### 条件随机场的定义

条件随机场就是在马尔可夫网络中加入输入节点，如下图所示



在自然语言处理中，使用的是线性链条件随机场，如下图所示



仿照着马尔可夫网络的联合概率分布公式可以写出条件随机场的条件概率分布公式，也就是模型预测公式

这里的就是输入的一个观测序列，就是对应的一个状态序列。表示序列中的位置序号，表示特征函数的序号，表示特征函数的序号。从3种序号来看的值取决于观测序列中每个位置（对求和）的两类特征函数（两类特征函数）各自拥有的所有特征函数（对求和，对求和）值。和就是模型中需要训练的未知数，表示对应特征函数的重要性，和越大表示特征在训练语料中出现频次越大，重要性越大。为归一化因子，它确保计算出的数值在0到1之间。和就是势函数也就是特征函数，它的取值为0或1。注意特征函数的自变量，说明该特征函数是定义在边（团）上的特征函数，是**局部**特征函数。注意特征函数的自变量，说明该特征函数是定义在结点上的特征函数，也是**局部**特征函数。从特征函数的所有自变量可以看出，该特征函数值取决于当前的序列位置，当前序列位置的状态，前一序列位置的状态，输入的序列。从特征函数的所有自变量可以看出，该特征函数值取决于当前的序列位置，当前序列位置的状态，输入的序列。

下面通过一个例子来理解上边的公式

例：设有一标注问题：输入观测序列为，输出标记序列为，取之于。假设特征，和对应的权值为，如下：

**对给定的观测序列，求标记序列为的非规范化条件概率。**

解：

为了计算列出下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

上表中只有绿色高亮的元素才有值，其他元素都为0，因此

为了计算列出下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

上表中只有绿色高亮的元素才有值，其他元素都为0，因此

### 简化形式的定义

其中为全局特征向量，一旦输入的观测序列和预测序列确定后，就成为有具体元素值的特征向量。是一个维的特征向量，即，的值等于特征函数的数量。是每个特征重要性的权重信息向量，显示。这里为什么说是全局特征向量呢，原因是它的每一个元素都是对整个序列来求值的，即，显然特征函数的定义最多只能涉及两个状态，当然最少也可以涉及一个状态。注意是整个观测序列。

## 特点

## 基本实践

### CRFsuite

### sklearn-CRFsuite

本节介绍基于sklearn-crfsuite实现一个简单的实体识别系统。

（1）准备数据

这里使用NLTK的CoNLL2002西班牙语语料。

|  |
| --- |
| # 查看训练和测试数据，这里数据使用NLTK中CoNLL2002的西班牙语料  print("文件ID", nltk.corpus.conll2002.fileids())  # 读出训练句子序列  train\_sents = list(nltk.corpus.conll2002.iob\_sents('esp.train'))  # 读出测试句子序列  test\_sents = list(nltk.corpus.conll2002.iob\_sents('esp.testb'))  # 1个句子由单词序列构成，每个的单词是一个三元组（词形，词性，实体标记）  print("打印训练集第1个句子", train\_sents[0]) |

（2）抽取特征

抽取特征的工作使用函数实现，这具有典型NLTK风格。通常条件随机场库（比如crf++）使用模板文件+语料生成特征。

|  |
| --- |
| # 这种定义一个函数来抽取特征的编码方式非常符合NLTK风格  # 给定一个句子的单词元组序列和当前词位置，返回针对该当前词的特征字典  def word2features(sent, i):  word = sent[i][0]  postag = sent[i][1]  features = {  'bias': 1.0,  'word.lower()': word.lower(),  'word[-3:]': word[-3:],  'word[-2:]': word[-2:],  'word.isupper()': word.isupper(),  'word.istitle()': word.istitle(),  'word.isdigit()': word.isdigit(),  'postag': postag,  'postag[:2]': postag[:2],  }  if i > 0:  word1 = sent[i-1][0]  postag1 = sent[i-1][1]  features.update({  '-1:word.lower()': word1.lower(),  '-1:word.istitle()': word1.istitle(),  '-1:word.isupper()': word1.isupper(),  '-1:postag': postag1,  '-1:postag[:2]': postag1[:2],  })  else:  features['BOS'] = True  if i < len(sent)-1:  word1 = sent[i+1][0]  postag1 = sent[i+1][1]  features.update({  '+1:word.lower()': word1.lower(),  '+1:word.istitle()': word1.istitle(),  '+1:word.isupper()': word1.isupper(),  '+1:postag': postag1,  '+1:postag[:2]': postag1[:2],  })  else:  features['EOS'] = True  return features  # 对一个句子中的每个单词抽取一个特征字典，返回所有单词的特征构成的特征序列  def sent2features(sent):  return [word2features(sent, i) for i in range(len(sent))]  # 返回一个句子（单词序列）的实体标记序列  def sent2labels(sent):  return [label for token, postag, label in sent] |

（3）训练

|  |
| --- |
| # 抽取训练数据每个单词的特征字典，构成特征字典序列X\_train  X\_train = [sent2features(s) for s in train\_sents]  # 训练数据每个特征字典对应的标记构成训练标记序列y\_train  y\_train = [sent2labels(s) for s in train\_sents]  # 抽取测试数据每个单词的特征字典，构成特征字典序列X\_test  X\_test = [sent2features(s) for s in test\_sents]  # 测试数据每个特征字典对应的标记构成测试标记序列y\_test  y\_test = [sent2labels(s) for s in test\_sents] |

（4）评测

|  |
| --- |
| y\_pred = crf.predict(X\_test)  labels = list(crf.classes\_)  # group B and I results  sorted\_labels = sorted(  labels,  key=lambda name: (name[1:], name[0])  )  # 从评测函数的输入参数可以看出：  # 实际标记序列y\_test与预测标记序列y\_pred必须在相同位置上必须对应的是同一个词语，  # 即分词一致  print(metrics.flat\_classification\_report(  y\_test, y\_pred, labels=sorted\_labels, digits=3  )) |

官网还介绍了如何输出重要的特征函数，这里就不再介绍了。

### CRF++

CRF++是一款简单的、可定制的、开源的CRF实现工具。它可用来进行分词和序列标注。CRF++是一款相对通用的工具，可用来处理各种NLP任务，如**命名实体识别、信息抽取、块提取**等。

#### 特征

（1）可以通过模板定义特征集

（2）使用C++的STL编写

（3）实现**LBFGS快速训练算法**，实现**拟牛顿法**解决数值型数据的训练问题。

（4）内存占用小

（5）输出top-n最优标注结果

（6）输出边缘概率分布

（7）Singe-best MIRA训练

（8）源码开源

#### 下载

2013年2月发布最新版代码。提供源码包和可执行程序包。

下载地址：<https://code.google.com/p/crfpp/downloads/list>。

#### 安装

（1）环境要求：gcc3.0及以上

（2）安装步骤：

|  |
| --- |
| $ ./configure  $ make  $ sudo make install |

#### 用法

1. 训练和测试文件的格式

（1）行的要求：每行一个token。多个token组成一个句子，句子之间以空行分隔。

（2）列的要求：每个token可以有多个列，但是各行列数必须一致，各列之间以空白分隔。每一列代表一种词法信息标记，比如第1列通常填字（词）形串，第2列填词性串等。

（3）标记列要求：标记列必须为最后一列。

示例：

|  |
| --- |
| He PRP B-NP  reckons VBZ B-VP  the DT B-NP  current JJ I-NP  account NN I-NP  deficit NN I-NP  will MD B-VP  narrow VB I-VP  to TO B-PP  only RB B-NP  # # I-NP  1.8 CD I-NP  billion CD I-NP  in IN B-PP  September NNP B-NP  . . O  He PRP B-NP  reckons VBZ B-VP  .. |

2. 编写特征模板

CRF++是一种通用序列标注工具，因此必须事先定义自己的模板，该模板用于描述训练和测试数据所要抽取的特征。

（1）基础模板和宏

模板文件中的每一行表示一种模板，一个模板的宏%x[row, col]用来指定输入数据中某个位置的一个token。row表示相对于当前token的行索引，col表示相对于当前token的列索引。

|  |
| --- |
| Input: Data  He PRP B-NP  reckons VBZ B-VP  the DT B-NP << CURRENT TOKEN  current JJ I-NP  account NN I-NP |

（2）模板类型

模板类型的标识在每行模板的行首，分别用字母“U”或字母“B”标识。其中字母“U”表示unigram模板，字母“B”表示bigram模板。需要注意的是unigram和bigram是针对输出类别标记的，而不是token。

下边举一个unigram的示例：

假设给定模板"U01:%x[0,1]"，和下边的一段tokens，

|  |
| --- |
| He PRP B-NP  reckons VBZ B-VP  the DT B-NP << CURRENT TOKEN  current JJ I-NP  account NN I-NP |

CRF++自动生成下边的特征函数集

|  |
| --- |
| func1 = if (output = B-NP and feature="U01:DT") return 1 else return 0  func2 = if (output = I-NP and feature="U01:DT") return 1 else return 0  func3 = if (output = O and feature="U01:DT") return 1 else return 0 |

显然，对于1个unigram模板，生成的特征函数数量为L\*N，其中L表示预测标记集合的长度，N表示对应于该模板展开的唯一token串的种类数。

bigram模板会将当前token和前一token组合起来展开，1个bigram模板生成的特征函数数量为L\*L\*N，其中L表示预测标记集合的长度，N表示对应于该模板展开的唯一token串的种类数。需要注意bigram模板会使训练和预测时间变长。

（3）模板标识

有时不同模板会展开成相同的token串，为了区分这一情况引入模板的编号。举例如下，输入tokens段为

|  |
| --- |
| The DT B-NP  pen NN I-NP  is VB B-VP << CURRENT TOKEN  a DT B-NP |

将"%x[-2,1]" 和"%x[1,1]"应用到该输入token段，都会展成"DT"这个值，但是它们来自不同的模板。为了区分两个值，在两个模板前加入模板表示号如下所示

|  |
| --- |
| U**01**:%x[-2,1]  U**02**:%x[1,1] |

（4）示例

下边列出一个完整的模板文件示例

|  |
| --- |
| # Unigram  U00:%x[-2,0]  U01:%x[-1,0]  U02:%x[0,0]  U03:%x[1,0]  U04:%x[2,0]  U05:%x[-1,0]/%x[0,0]  U06:%x[0,0]/%x[1,0]  U10:%x[-2,1]  U11:%x[-1,1]  U12:%x[0,1]q  U13:%x[1,1]  U14:%x[2,1]  U15:%x[-2,1]/%x[-1,1]  U16:%x[-1,1]/%x[0,1]  U17:%x[0,1]/%x[1,1]  U18:%x[1,1]/%x[2,1]  U20:%x[-2,1]/%x[-1,1]/%x[0,1]  U21:%x[-1,1]/%x[0,1]/%x[1,1]  U22:%x[0,1]/%x[1,1]/%x[2,1]  # Bigram  B |

3. 训练（编码问题）

（1）训练命令

|  |
| --- |
| $ **crf\_learn -t template\_file train\_file model\_file**  $ crf\_learn -f 3 -c 1.5 template\_file train\_file model\_file |

（2）命令参数

-a CRF-L2|CRF-L1：设置正则化方法，默认值为CRF-L2。一般情况下，L2效果优于L1，但是L1模型的非零特征远小于L2。

-c 浮点数：控制模型过拟合与欠拟合的程度，该值会对系统效果有显著的影响。该值越大，模型的过拟合程度越大。

-f 整数：特征选择的阈值，默认值为1。那些出现频次低于该阈值的特征将被删除。

-p 整数：对于多核CPU可以使用多线程完成模型训练，该参数指定线程数量。

（3）控制台输出信息

iter：当前迭代的次数

terr：token错误率

serr：sentence错误率

obj：当前模型对象值，当其收敛到一个固定值后，迭代停止

diff：本次迭代与前次迭代对象值的差

4. 预测（解码问题）

（1）预测命令

|  |
| --- |
| $ **crf\_test -m model\_file test\_files ... >result.txt** |

（2）命令参数

-v 1：预测结果中给出top1标记的概率

-v 2：预测结果中给出所有标记的概率

-n 整数：预测结果中给出topn标记的概率

#### 示例

在CRF++工具中的example目录中提供了4个示例，分别为基本名词块识别（basenp），组块分析（chunking），日本命名实体识别（JapaneseNE），分词（seg）。可以直接运行下边的命令进行训练和预测

|  |
| --- |
| $ crf\_learn template train model  $ crf\_test -m model test |

#### 提示

（1）CRF++的输入数据格式与YamCha工具

<http://chasen.org/~taku/software/yamcha/>

的输入数据格式相同，可以用同一份评测数据在两种工具上运行，对比CRF与SVM的识别效果。

（2）CRF++输出数据格式与CoNLL2002评测数据格式一致，可以用perl脚本

<https://www.aflat.org/conll2000/chunking/output.html>

评测系统效果，该脚本将给出所有标记类的F值。

#### 参考

（1）J. Lafferty, A. McCallum, and F. Pereira. **Conditional random fields: Probabilistic models for segmenting and labeling sequence data**, In Proc. of ICML, pp.282-289, 2001

（2）F. Sha and F. Pereira. **Shallow parsing with conditional random fields**, In Proc. of HLT/NAACL 2003

# HanLP词法分析

## 分词

## 词性标注

## 命名实体识别

## HanLP的n元语法维特比分词器

1. 人名识别默认打开。
2. 地名识别是否打开对精度影响不大，因此初期可以打开地名识别。
3. 机构名打开后，精度下降很大，因此在初期不做基于HMM的机构名识别。而暂时只用词表匹配法。

### 分词器配置变量

分词器的相关配置定义在Config.java类中，这里我们将分词相关的所有配置变量列于下表

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名称 | 变量名 | 默认值 | 备注 |
| 索引分词的词语最小长度 | indexMode | 0 | 整数值，为0时表示不进行索引全切分 |
| 是否识别中国人名 | nameRecognize | true |  |
| 是否识别音译人名 | translatedNameRecognize | true |  |
| 是否识别日本人名 | japaneseNameRecognize | false |  |
| 是否识别地名 | placeRecognize | false |  |
| 是否识别机构 | organizationRecognize | false |  |
| 是否加载用户词典 | useCustomDictionary | false |  |
| 用户词典是否为高优先级 | forceCustomDictionary | false |  |
| 是否进行词性标注 | speechTagging | false |  |
| 命名实体识别是否至少有一项被激活 | ner | true |  |
| 是否计算词语偏移量 | offset | false |  |
| 是否识别数字和量词 | numberQuantifierRecognize | false |  |
| 并行分词的线程数 | threadNumber | 1 |  |

这种配置类什么时候实例化呢，不用想肯定是分词开始前就会实例化，拿HanLP类中的ViterbiSegment分词类举例。该类的继承关系用如下图所示：



由继承关系图可以看到，只要实例化ViterbiSegment则首先会执行Segment()初始化，在该方法中实例化分词器配置对象config。这些配置变量都是公有变量，因此可以在ViterbiSegment类实例化为对象后直接在外部修改。那么什么时候来使用这些配置变量呢，当然是在分词的时候，具体是哪个类的哪个方法呢，当然是ViterbiSegment类的List<Term> segSentence(char[] sentence)方法。

### 词典的使用条件和先后顺序

我们知道了词典配置变量使用的位置后，就可以确定每个词典的使用条件了以及每个词典的使用顺序

1. 词语粗分

（1）构建词图

对应方法为void generateWordNet(final WordNet wordNetStorage)，在此方法中系统使用CoreNatureDictionary.txt文件切分出所有可能的分词路径。此时如果配置变量useCustomDictionary为true，则将CustomDictionary.txt中的词也考虑进来，说明CustomDictionary.txt优先级会高。另外大家可以看到CoreNatureDictionary.txt实际上也充当了隐马词性标注的发射矩阵，里边某些多词性词也列出了词性序列以及各词性对应的频次。

（2）用户定制词典干预

如果配置变量useCustomDictionary为true，即需要使用CustomDictionary.txt进行干预，则执行下边对应的方法，否则跳过该步骤。用户词典干预根据是否进行全切分有两种不同方法：当配置变量indexMode>0时，即系统处于全切分模式时，对应方法为

List<Vertex> combineByCustomDictionary(List<Vertex> vertexList, DoubleArrayTrie<CoreDictionary.Attribute> dat, final WordNet wordNetAll)，

如果indexMode=0，即系统处于普通分词模式，对应方法为

List<Vertex> combineByCustomDictionary(List<Vertex> vertexList, DoubleArrayTrie<CoreDictionary.Attribute> dat)。

从调用的方法我们不难看出，全切分时系统会根据CustomDictionary.txt添加分词路径。而普通切分时，系统会根据CustomDictionary.txt合并路径。这也就是为什么有的时候明明已经在CustomDictionary.txt中添加了新词却不生效的原因，因为一旦根据CoreNatureDictionary.txt构建了词图就不会再有新的路径插到已有分词路径中间，此时就去查找并修改CoreNatureDictionary.txt中的相关字或词吧。

（3）维特比选择最优路径

对应方法为List<Vertex> viterbi(WordNet wordNet)，至此就得到了一个粗分的分词结果。需要注意HanLP的Viterbi分词只是用viterbi方法求解最优路径，并不是隐马。

3. 数字识别

如果配置变量numberQuantifierRecognize为true，则在粗分结果的基础上进行数字合并操作，否则直接跳过该步。对应方法为

void mergeNumberQuantifier(List<Vertex> termList, WordNet wordNetAll, Config config)。

4. 实体识别

配置变量ner为true时，则需要进行各种实体的识别，继续向下执行。需要注意该变量受其他实体识别变量影响，只要其他任意实体配置变量为true，则ner就会为true。如果ner为false，则跳过下边各项实体识别继续词性标注环节。

（1）中国人名识别

执行此步，配置变量nameRecognize必须为true。调用方法为

PersonRecognition.recognition(vertexList, wordNetOptimum, wordNetAll)。人名使用隐马，因此有转移矩阵nr.tr.txt和发射矩阵nr.txt。由于HanLP不提供训练语料，我们自己也很难得到有角色标注的语料，因此我们一般只修改nr.txt文件，删除nr.txt.bin文件后生效。

（2）音译人名识别

执行此步，配置变量translatedNameRecognize必须为true。调用方法为

TranslatedPersonRecognition.recognition(vertexList, wordNetOptimum, wordNetAll)。需要注意音译人名的识别没有用隐马，就是匹配分词法。涉及到的词典为nrf.txt，如果用户修改该词典，则需要删除nrf.txt.trie.dat使其生效。

（3）日本人名识别

执行此步，配置变量japaneseNameRecognize必须为true。调用方法为

JapanesePersonRecognition.recognition(vertexList, wordNetOptimum, wordNetAll)。需要注意日本人名的识别没有用隐马，就是匹配分词法。涉及到的词典为nrj.txt，如果用户修改该词典，则需要删除nrj.txt.trie.dat和nrj.txt.value.dat使其生效。

（4）地名识别

执行此步，配置变量placeRecognize必须为true。调用方法为

PlaceRecognition.recognition(vertexList, wordNetOptimum, wordNetAll)。地名使用隐马，因此有转移矩阵ns.tr.txt和发射矩阵ns.txt。由于HanLP不提供训练语料，我们自己也很难得到有角色标注的语料，因此我们一般只修改ns.txt文件，删除ns.txt.bin文件后生效。

（5）机构名识别

执行此步，配置变量organizationRecognize必须为true。调用方法为

OrganizationRecognition.recognition(vertexList, wordNetOptimum, wordNetAll)。注意这里在调用机构名识别之前先进行了一次识别，也就是层叠隐马，而人名和地名的识别就是普通的隐马。机构名的识别使用层叠隐马，涉及的文件有转移矩阵nt.tr.txt和发射矩阵nt.txt。由于HanLP不提供训练语料，我们自己也很难得到有角色标注的语料，因此我们一般只修改nt.txt文件，删除ns.txt.bin文件后生效。机构名的识别需要人名地名识别具有较高准确率。

|  |
| --- |
| 其他没有在系统中使用的词典有  机构名词典.txt  全国地名大全.txt  人名词典.txt  上海地名.txt  现代汉语补充词库.txt  这些词典是对系统中的词典的更新记录，如果你添加了新的人名、地名、机构名可以在这里添加保存。  另外，如果需要添加人名、地名、机构名可以直接在CoreNatureDictionary.txt中添加，最好是3字以上实体，如果要去掉错误识别的命名实体可以直接在相应的nr.txt，ns.txt，nt.txt中添加。 |

5. 索引全切分

如果配置变量indexMode为true则使用方法

List<Term> decorateResultForIndexMode(List<Vertex> vertexList, WordNet wordNetAll)执行全切分。

6. 词性标注

如果配置变量speechTagging为true则进行隐马词性标注，对应方法为

void speechTagging(List<Vertex> vertexList)。

词性标注使用了隐马，因此涉及到词典CoreNatureDictionary.tr.txt。同样因为我们没有14年的人民日报语料，因此这一部分一般也不修改。需要注意speechTagging默认为false，指的是不进行隐马的词性标注，但不表示系统不进行词性标注，此时系统使用最大概率的词性作为词语的词性，因此CoreNatureDictionary.txt是很重要的。

### 多线程分词

HanLP的ViterbiSegment分词器类是支持多线程的，线程数量由配置变量threadNumber决定的，该变量默认为1。HanLP作者说ViterbiSegmet分词效率最高的原因肯定也有ViterbiSegment分词器支持多线程分词这个因素。另外由于ViterbiSegment分词器内部所具有的相关命名实体功能，因此这些命名实体识别的效率也会很高。在哪里实现的多线程分词呢，在Segment类的List<Term> seg(String text)这个方法中实现的，需要注意HanLP的多线程分词指的是一次输入了一个长文本，而不是一次处理多个输入文本。

### 基于角色标注的中国人名识别

在HanLP中，基于角色标注识别了中国人名。首先系统利用隐马尔可夫模型标注每个词语的角色，之后利用最大模式匹配法对角色序列进行匹配，匹配上模式的即为人名。理论指导文章为：《基于角色标注的中国人名自动识别研究》，网页地址为：<http://www.hankcs.com/nlp/chinese-name-recognition-in-actual-hmm-viterbi-role-labeling.html>

#### 人名角色简表

所有与人名有关的文本信息被称为角色，比如人名的前词，后词，首字，尾字等。以“馆/内/陈列/周/恩/来/和/邓/颖/超生/前/使用/过/的/物品/。”为例说明其中词语的角色序列。这句话的角色序列为“馆/A内/A 陈列/K周/B恩/C来/D和/M邓/B颖/C超生/V前/A使用/A过/A的/A物品/A。/A”。

不同的实体有不同的角色标注，实际就是特征，这些特征需要有语言学的知识，实际上就是你的阅读量，通过你大量阅读总结经验，比如姓氏可以作为名字的一个特征（张、王、李、赵），常用地名的后缀可以作为一个特征（省、市、区、县），机构名表处所的尾字可以作为一个特征（局、处、所、院）。

下表总结了人名识别中所有角色种类及其标记：

|  |  |  |
| --- | --- | --- |
| 标记 | 意义 | 例子 |
| B | 姓氏 | 张华平先生 |
| C | 双名的首字 | 张华平先生 |
| D | 双名的末字 | 张华平先生 |
| E | 单名 | 张浩说：“我是一个好人” |
| F | 前缀 | 老刘、小李 |
| G | 后缀 | 王总、刘老、肖氏、吴妈、叶帅 |
| K | 人名的上文 | 又来到于洪洋的家。 |
| L | 人名的下文 | 新华社记者黄文摄 |
| M | 两个中国人名之间的成分 | 编剧邵钧林和稽道青说 |
| U | 人名的上文和姓成词 | 这里有关天培的壮烈 |
| V | 人名的末字和下文成词 | 龚学平等领导, 邓颖超生前 |
| X | 姓与双名的首字成词 | 王国维、 |
| Y | 姓与单名成词 | 高峰、汪洋 |
| Z | 双名本身成词 | 张朝阳 |
| A | 以上之外其他的角色 |  |

#### 怎样通过角色序列识别出人名？

基于角色序列的人名识别就是一个简单的模式最大匹配问题。人名模式的识别集为**{ BBCD, BBE, BBZ, BCD, BEE,BE,BG,BXD,BZ,CD,EE,FB, Y,XD}**。例句“馆/内/陈列/周/恩/来/和/邓/颖/超生/前/使用/过/的/物品/。”对应的T#为：“AAKBCDMBC*V*AAAAAA”。V分裂处理后，最终的角色序列为：“AAKBCDMBC*DL*AAAAAA”。模式最大匹配后，我们识别出的人名是：“周恩来”和“邓颖超”。

#### 隐马的三要素

观察值序列：词1 词2 … 词n

隐变量序列：角色1 角色2 … 角色n

1. **初始概率分布**

初始概率分布即为各种角色的概率分布，如下表所示

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | C | D | E | F | G | K | L | M | U | V | X | Y | Z | A |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **状态转移概率分布**

状态转移概率分布即为各角色之间转移的概率分布，如下表所示

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | B | C | D | E | F | G | K | L | M | U | V | X | Y | Z | A |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| K |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| L |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| M |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| U |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Y |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **观测概率分布**

观测概率分布即各角色发射出各词语的概率分布，如下表所示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 词1 | 词2 | … | 词|V| |
| B |  |  | … |  |
| C |  |  | … |  |
| D |  |  | … |  |
| E |  |  | … |  |
| F |  |  | … |  |
| G |  |  | … |  |
| K |  |  | … |  |
| L |  |  | … |  |
| M |  |  | … |  |
| U |  |  | … |  |
| V |  |  | … |  |
| X |  |  | … |  |
| Y |  |  | … |  |
| Z |  |  | … |  |
| A |  |  | … |  |

词|V|表示训练集词汇表长度。

#### badcase分类整理

1. **核心词典中的单字姓氏人名引起误识别**

核心词典中的单字人名姓氏影响

[1] 日本人为什么会推崇降臣王**铎**的书法？

[2] 徐**峥**小陶虹夫妇《信中国》，致敬汶川地震救灾英雄，缅怀遇难同胞

[3] **嵇**境雷——新时代最具收藏价值的书画名家作品展

解决方法：去掉核心词典中的单字姓氏，将没有识别全的名字放在人工干预表中。核心词典中最好不要放单名姓氏。

1. **人名、地名混淆引起的误识别**

[1] **马嵬驿**和白鹿原旅游景区的游客为什么日益锐减？

[2] **韩美今**启动“超级雷霆”联合军演，8架F22战机参演

[3] 安徽**毛坦厂**镇各地学生就读 逾万家长陪读

解决方法：核心词典中进行地名干预。

1. **双名本身成词、姓与双名首字成词、姓与单名成词、人名末字和下文成词、人名上文和姓成词**

[1] 房价**高全怪**炒房客 炒房客：不炒房我们炒什么？

[2] 开发**商抢地**还抢人！南阳房地产市场要开挂！

[3] 水磨镇涅槃重生 工业重镇成功转型**康养**旅游景区

解决方法：

1. 将人名词典中的成词人名标记为忽略，这样的词语也应该放到核心词典中。
2. HanLP发射矩阵词典nr.txt中收录单字姓氏619个。袁义达在《中国的三大姓氏是如何统计出来的》文献中指出：当代中国100个常见姓氏中，集中了全国人口的87%，根据这一数据我们只保留nr.txt中的100个常见词语的姓氏角色，其他词语去掉其姓氏角色状态。过滤后，nr.txt中具有姓氏角色的单字共计97个。列于下表：

|  |
| --- |
| 丁 万 乔 于 任 何 余 侯 傅 冯 刘 卢 史 叶 吕 吴 周 唐 夏 姚  姜 孔 孙 孟 宋 尹 崔 常 康 廖 张 彭 徐 戴 方 易 曹 曾 朱 李  杜 杨 林 梁 武 段 毛 江 汤 汪 沈 潘 熊 王 田 白 石 秦 程 罗  胡 苏 范 萧 董 蒋 薛 袁 许 谢 谭 贺 贾 赖 赵 邓 邱 邵 邹 郑  郝 郭 金 钟 钱 阎 陆 陈 雷 韩 顾 马 高 魏 黄 黎 龚 |

对于非常用姓氏构成的人名，我们人工干预那些3字以上的人名，其他的人名不做识别。

1. 双字人名模式过滤

我们对87个误识别的双字人名的角色序列进行了统计，人名和其对应角色列于下表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 奥鹏 | EE | 谷寿 | EE | 交安 | EE | 青创 | EE | 温阳 | EE |
| 小司 | FC | 龟峰 | EE | 于练 | BC | 青洋 | EE | 吴公 | BC |
| 霸屏 | EE | 过么 | EE | 京云 | EE | 屈光 | EE | 武文 | BC |
| 白蝶 | BE | 函谷 | EE | 于右 | BC | 任城 | BE | 袭黔 | EE |
| 小田 | FB | 韩美 | BC | 康养 | BC | 任君 | BC | 弦流 | EE |
| 星繁 | EE | 叶绿 | BC | 抗凝 | EE | 沙鹰 | EE | 小麦 | FC |
| 超超 | EE | 寒香 | EE | 原牧 | EE | 赏雪 | EE | 小密 | FE |
| 晨峰 | EE | 何会 | BC | 蓝德 | EE | 射导 | EE |  |  |
| 薛家 | BC | 何有 | BC | 浪淘 | EE | 沈师 | BC |  |  |
| 电竞 | EE | 易疆 | BE | 老多 | FC | 施策 | EE |  |  |
| 方宅 | BE | 胡赛 | BC | 霑梅 | EE | 熟宣 | EE |  |  |
| 非遗 | EE | 易旺 | BC | 黎坪 | BE | 帅铃 | EE |  |  |
| 枫示 | EE | 音悦 | EE | 张图 | BC | 苏城 | BE |  |  |
| 高带 | BC | 黄安 | BC | 毛体 | BC | 唐口 | BC |  |  |
| 何造 | BC | 黄貂 | BE | 郑医 | BC | 唐冶 | BC |  |  |
| 高屋 | BE | 黄鹤 | BC | 周起 | BC | 塘峪 | EE |  |  |
| 燕舞 | EE | 鹰爸 | EE | 周周 | BC | 添香 | EE |  |  |
| 龚滩 | BE | 鹰跃 | EE | 钱会 | BC | 汪山 | BE |  |  |
| 杨飘 | BC | 影添 | EE | 茜溪 | EE | 王洞 | BC |  |  |
| 杨絮 | BC | 江静 | BE | 秦报 | BE | 唯逢 | EE |  |  |

各角色序列频次分布如下

|  |  |
| --- | --- |
| EE | 41 |
| BC | 29 |
| BE | 12 |
| FC | 3 |
| FB | 1 |
| FE | 1 |

目前只保留下边的双字（词）角色模式

|  |  |
| --- | --- |
| 模式串 | 示例 |
| BG | 王总 |
| BZ | 张朝阳 |
| FB | 老张 |

1. 三字人名模式过滤

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 韩美今 | BCD | 才子多 | XD | 古树群 | XD | 吉祥新 | XD |
| 高全怪 | BCD | 陈仓米 | XD | 古思特 | XD | 江景房 | XD |
| 华十大 | BCD | 成人礼 | XD | 关东军 | XD | 金龙盘 | XD |
| 巴马香 | XD | 杜蕾斯 | XD | 胡里山 | XD | 康复为 | XD |
| 白音昌 | XD | 宫川英 | XD | 花田酒 | XD | 来华政 | XD |

从上表可以看出，识别出的姓与双名首字成词的角色构成的人名基本都被识别错了。

目前只保留下边的双字（词）角色模式

|  |  |
| --- | --- |
| 模式串 | 示例 |
| BBE | 刘张虎 |
| BCD | 张华平 |
| BEE | 林华华 |

1. **双名末字为动词**

[1] 著名书法家**张海为**自己立“卧薪尝胆状”

解决方法：将误识别的人名标注为忽略。如果后续这种人名较多了，则将“为”字作为双名末字的角色去掉。

1. **对联、诗歌等文学作品中由姓氏引发的人名误识别**

[1] 上联：**常在花**间走，如何对下联？

[2] 上联：**江中鱼**东游西藏，下联怎么对？

[3] 楷书《**石涧记**》

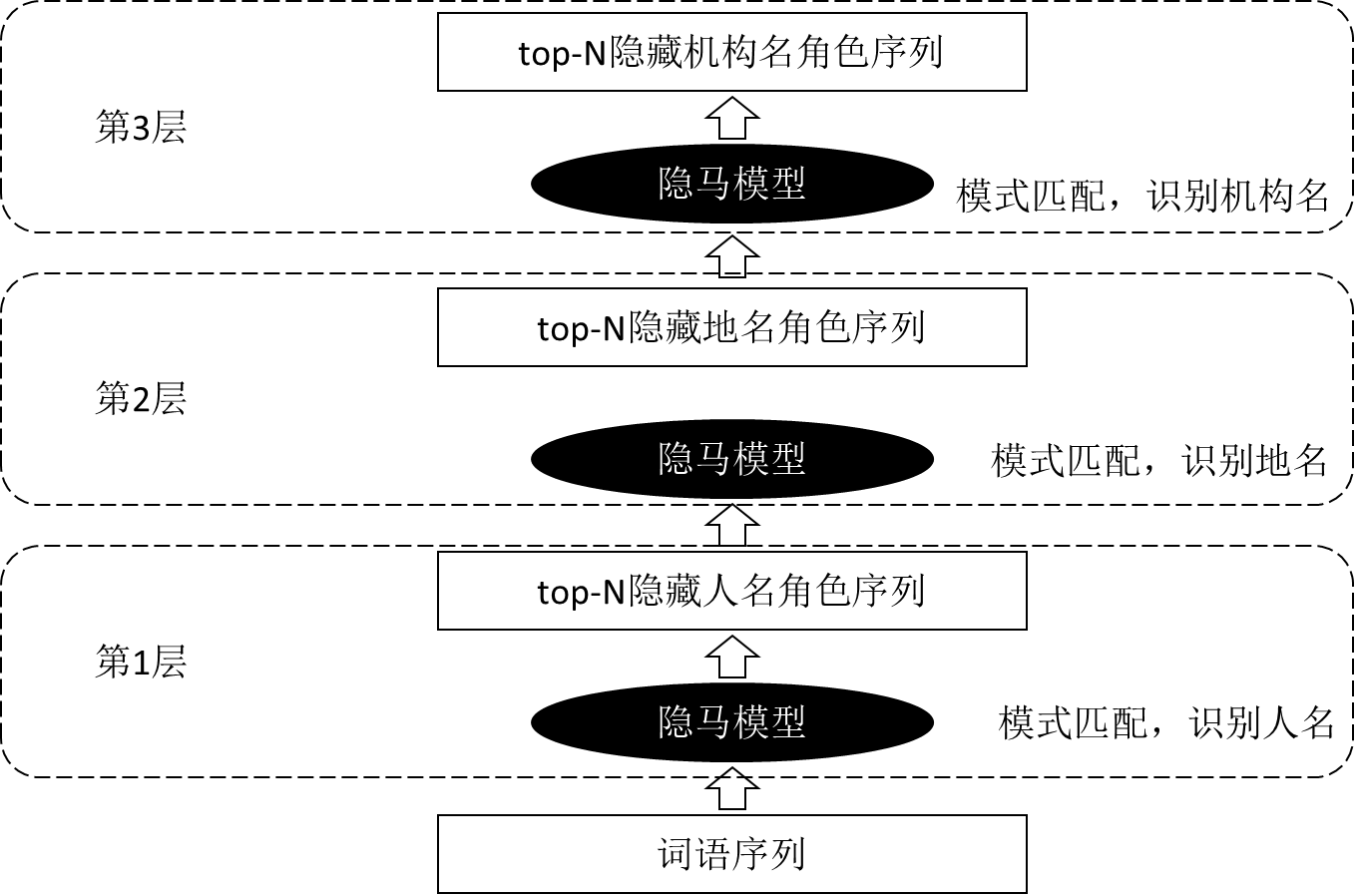
解决方法：目前只能发现后标注为忽略。

### 基于角色标注的中国地名识别

理论指导文章为：《基于层叠隐马尔可夫模型的中文命名实体识别》，网页地址为：

<http://www.hankcs.com/nlp/ner/place-names-to-identify-actual-hmm-viterbi-role-labeling.html>。

1. **层叠隐马**



如图所示，层叠隐马就是训练三个隐马模型，每个模型标注一种实体，三个模型采用级联形式连接。

1. **地名角色简表**

|  |  |  |
| --- | --- | --- |
| 角色 | 意义 | 例子 |
| A | 地名的上文 | 我/来到/中/关/园 |
| B | 地名的下文 | 刘家村/和/下岸村/相邻 |
| C | 中国地名的首部 | 石/河/子/乡 |
| D | 中国地名的中部 | 石/河/子/乡 |
| F | 中国地名的末部 | 石/河/子/乡 |
| G | 中国地名的后缀 | 海/淀区 |
| X | 连接词 | 刘家村/和/下岸村/相邻 |
| Z | 其他非名成分 |  |

### 基于角色标注的中国机构名识别

机构名的理论指导文章为：《基于角色标注的中文机构名识别》，网页地址为：

<http://www.hankcs.com/nlp/ner/place-name-recognition-model-of-the-stacked-hmm-viterbi-role-labeling.html>。

1. 机构名角色标注简表

|  |  |  |
| --- | --- | --- |
| 角色 | 意义 | 例子 |
| A | 上文 | *参与*亚太经合组织的活动 |
| B | 下文 | 中央电视台*报道* |
| X | 连接词 | 北京电视台*和*天津电视台 |
| C | 特征词的一般性前缀 | 北京*电影*学院 |
| F | 特征词的译名性前缀 | 美国*摩托罗拉*公司 |
| G | 特征词的地名性前缀 | 交通银行*北京*分行 |
| H | 特征词的机构名前缀 | *中共中央*顾问委员会 |
| I | 特征词的特殊性前缀 | *中央*电视台 |
| J | 特征词的简称性前缀 | *巴*政府 |
| D | 机构名的特征词 | 国务院侨务*办公室* |
| Z | 非机构名成份 |  |

## HanLP词法分析工具——训练用户领域分词、词性标注、实体识别模型

## 流行自然语言处理工具效果对比

# 医疗领域实体识别

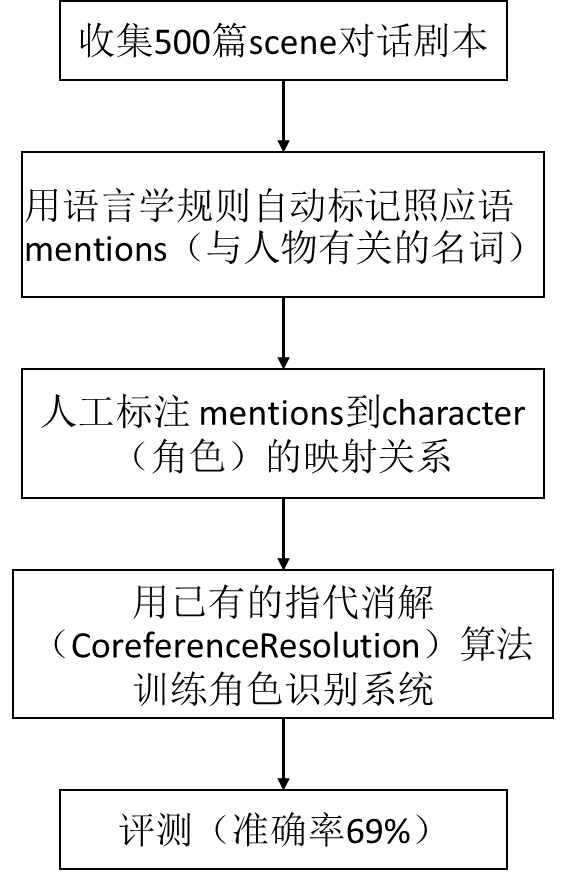
## 调研

## 药品名识别

## 疾病名识别

# 角色识别

## 角色识别系统构建流程



## 指代消解

指代一般分为两种情况：回指（anaphora，也称为指示性指代）和共指（co-reference，也称同指）。所谓回指，是指当前的指示语与上下文出现的词、短语或句子（句群）存在密切的语义关联性；共指则主要是指两个名词（包括代名词、名词短语）指向真实世界中的同一参照体。共指关系可以独立于上下文存在。例如，“美国总统”和“布什”就是共指。

指代消解是指将文本中指称同一对象的所有表达形式关联起来。包括具有指代关系的指示语和先行语，以及具有同指关系的词语。分析文档中实体之间的指代关系，同一个实体在所分析的篇章中，可能有多种不同的指代方法，指代的解析就是将不同的指代连接到同一实体上。