**HanLP**是一系列模型与算法组成的NLP工具包，由大快搜索主导并完全开源，目标是普及自然语言处理在生产环境中的应用。**HanLP**具备功能完善、性能高效、架构清晰、语料时新、可自定义的特点。

**HanLP**提供下列功能：

* 中文分词
  + HMM-Bigram（速度与精度最佳平衡；一百兆内存）
    - [最短路分词](https://github.com/hankcs/HanLP#1-%E7%AC%AC%E4%B8%80%E4%B8%AAdemo)、[N-最短路分词](https://github.com/hankcs/HanLP#5-n-%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84%E5%88%86%E8%AF%8D)
  + 由字构词（侧重精度，可识别新词；适合NLP任务）
    - [感知机分词](https://github.com/hankcs/HanLP/wiki/%E7%BB%93%E6%9E%84%E5%8C%96%E6%84%9F%E7%9F%A5%E6%9C%BA%E6%A0%87%E6%B3%A8%E6%A1%86%E6%9E%B6)、[CRF分词](https://github.com/hankcs/HanLP#6-crf%E5%88%86%E8%AF%8D)
  + 词典分词（侧重速度，每秒数千万字符；省内存）
    - [极速词典分词](https://github.com/hankcs/HanLP#7-%E6%9E%81%E9%80%9F%E8%AF%8D%E5%85%B8%E5%88%86%E8%AF%8D)
  + 所有分词器都支持：
    - [索引全切分模式](https://github.com/hankcs/HanLP#4-%E7%B4%A2%E5%BC%95%E5%88%86%E8%AF%8D)
    - [用户自定义词典](https://github.com/hankcs/HanLP#8-%E7%94%A8%E6%88%B7%E8%87%AA%E5%AE%9A%E4%B9%89%E8%AF%8D%E5%85%B8)
    - [兼容繁体中文](https://github.com/hankcs/HanLP/blob/master/src/test/java/com/hankcs/demo/DemoPerceptronLexicalAnalyzer.java#L29)
    - [训练用户自己的领域模型](https://github.com/hankcs/HanLP/wiki)
* 词性标注
  + [HMM词性标注](https://github.com/hankcs/HanLP/blob/master/src/main/java/com/hankcs/hanlp/seg/Segment.java#L584)（速度快）
  + [感知机词性标注](https://github.com/hankcs/HanLP/wiki/%E7%BB%93%E6%9E%84%E5%8C%96%E6%84%9F%E7%9F%A5%E6%9C%BA%E6%A0%87%E6%B3%A8%E6%A1%86%E6%9E%B6)、[CRF词性标注](https://github.com/hankcs/HanLP/wiki/CRF%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90)（精度高）
* 命名实体识别
  + 基于HMM角色标注的命名实体识别 （速度快）
    - [中国人名识别](https://github.com/hankcs/HanLP#9-%E4%B8%AD%E5%9B%BD%E4%BA%BA%E5%90%8D%E8%AF%86%E5%88%AB)、[音译人名识别](https://github.com/hankcs/HanLP#10-%E9%9F%B3%E8%AF%91%E4%BA%BA%E5%90%8D%E8%AF%86%E5%88%AB)、[日本人名识别](https://github.com/hankcs/HanLP#11-%E6%97%A5%E6%9C%AC%E4%BA%BA%E5%90%8D%E8%AF%86%E5%88%AB)、[地名识别](https://github.com/hankcs/HanLP#12-%E5%9C%B0%E5%90%8D%E8%AF%86%E5%88%AB)、[实体机构名识别](https://github.com/hankcs/HanLP#13-%E6%9C%BA%E6%9E%84%E5%90%8D%E8%AF%86%E5%88%AB)
  + 基于线性模型的命名实体识别（精度高）
    - [感知机命名实体识别](https://github.com/hankcs/HanLP/wiki/%E7%BB%93%E6%9E%84%E5%8C%96%E6%84%9F%E7%9F%A5%E6%9C%BA%E6%A0%87%E6%B3%A8%E6%A1%86%E6%9E%B6)、[CRF命名实体识别](https://github.com/hankcs/HanLP/wiki/CRF%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90)
* 关键词提取
  + [TextRank关键词提取](https://github.com/hankcs/HanLP#14-%E5%85%B3%E9%94%AE%E8%AF%8D%E6%8F%90%E5%8F%96)
* 自动摘要
  + [TextRank自动摘要](https://github.com/hankcs/HanLP#15-%E8%87%AA%E5%8A%A8%E6%91%98%E8%A6%81)
* 短语提取
  + [基于互信息和左右信息熵的短语提取](https://github.com/hankcs/HanLP#16-%E7%9F%AD%E8%AF%AD%E6%8F%90%E5%8F%96)
* [拼音转换](https://github.com/hankcs/HanLP#17-%E6%8B%BC%E9%9F%B3%E8%BD%AC%E6%8D%A2)
  + 多音字、声母、韵母、声调
* [简繁转换](https://github.com/hankcs/HanLP#18-%E7%AE%80%E7%B9%81%E8%BD%AC%E6%8D%A2)
  + 简繁分歧词（简体、繁体、臺灣正體、香港繁體）
* [文本推荐](https://github.com/hankcs/HanLP#19-%E6%96%87%E6%9C%AC%E6%8E%A8%E8%8D%90)
  + 语义推荐、拼音推荐、字词推荐
* 依存句法分析
  + [基于神经网络的高性能依存句法分析器](https://github.com/hankcs/HanLP#21-%E4%BE%9D%E5%AD%98%E5%8F%A5%E6%B3%95%E5%88%86%E6%9E%90)
  + [MaxEnt依存句法分析](https://github.com/hankcs/HanLP/blob/master/src/main/java/com/hankcs/hanlp/dependency/MaxEntDependencyParser.java)
* [文本分类](https://github.com/hankcs/HanLP/wiki/%E6%96%87%E6%9C%AC%E5%88%86%E7%B1%BB%E4%B8%8E%E6%83%85%E6%84%9F%E5%88%86%E6%9E%90)
  + [情感分析](https://github.com/hankcs/HanLP/wiki/%E6%96%87%E6%9C%AC%E5%88%86%E7%B1%BB%E4%B8%8E%E6%83%85%E6%84%9F%E5%88%86%E6%9E%90#%E6%83%85%E6%84%9F%E5%88%86%E6%9E%90)
* [word2vec](https://github.com/hankcs/HanLP/wiki/word2vec)
  + 词向量训练、加载、词语相似度计算、语义运算、查询、KMeans聚类
  + 文档语义相似度计算
* [语料库工具](https://github.com/hankcs/HanLP/tree/master/src/main/java/com/hankcs/hanlp/corpus)
  + 默认模型训练自小型语料库，鼓励用户自行训练。所有模块提供训练接口，语料可参考[OpenCorpus](https://github.com/hankcs/OpenCorpus)。

在提供丰富功能的同时，**HanLP**内部模块坚持低耦合、模型坚持惰性加载、服务坚持静态提供、词典坚持明文发布，使用非常方便，同时自带一些语料处理工具，帮助用户训练自己的模型。

**中文分词做不好，人机自然语言交互当然难取得突破**

2017-06-28 18:56

中文分词是中文文本处理的一个基础步骤，也是中文人机自然语言交互的基础模块。不同于英文的是，中文句子中没有词的界限，因此在进行中文自然语言处理时，通常需要先进行分词，分词效果将直接影响词性、句法树等模块的效果。当然分词只是一个工具，场景不同，要求也不同。

在人机自然语言交互中，成熟的中文分词算法能够达到更好的自然语言处理效果，帮助计算机理解复杂的中文语言。竹间智能在构建中文自然语言对话系统时，结合语言学不断优化，训练出了一套具有较好分词效果的算法模型，为机器更好地理解中文自然语言奠定了基础。

在此，对于**中文分词方案、当前分词器存在的问题，以及中文分词需要考虑的因素及相关资源**，***竹间智能自然语言与深度学习小组***做了整理和总结，希望能为大家提供一些参考。

中文分词根据实现原理和特点，主要分为以下2个类别：

**1、基于词典分词算法**

也称字符串匹配分词算法。该算法是按照一定的策略将待匹配的字符串和一个已建立好的“充分大的”词典中的词进行匹配，若找到某个词条，则说明匹配成功，识别了该词。常见的基于词典的分词算法分为以下几种：**正向最大匹配法、逆向最大匹配法**和**双向匹配分词法**等。

基于词典的分词算法是应用最广泛、分词速度最快的。很长一段时间内研究者都在对基于字符串匹配方法进行优化，比如最大长度设定、字符串存储和查找方式以及对于词表的组织结构，比如采用TRIE索引树、哈希索引等。

**2、基于统计的机器学习算法**

这类目前常用的是算法是**HMM、CRF、SVM、深度学习**等算法，比如stanford、Hanlp分词工具是基于CRF算法。以CRF为例，基本思路是对汉字进行标注训练，不仅考虑了词语出现的频率，还考虑上下文，具备较好的学习能力，因此其对歧义词和未登录词的识别都具有良好的效果。

Nianwen Xue在其论文《Combining Classifiers for Chinese Word Segmentation》中首次提出对每个字符进行标注，通过机器学习算法训练分类器进行分词，在论文《Chinese word segmentation as character tagging》中较为详细地阐述了基于字标注的分词法。

常见的分词器都是使用**机器学习算法和词典相结合，一方面能够提高分词准确率，另一方面能够改善领域适应性。**

随着深度学习的兴起，也出现了**基于神经网络的分词器**，例如有人员尝试使用双向LSTM+CRF实现分词器，**其本质上是序列标注**，所以有通用性，命名实体识别等都可以使用该模型，据报道其分词器字符准确率可高达97.5%。算法框架的思路与论文《Neural Architectures for Named Entity Recognition》类似，利用该框架可以实现中文分词，如下图所示：



首先对语料进行字符嵌入，将得到的特征输入给双向LSTM，然后加一个CRF就得到标注结果。

**分词器当前存在问题：**

目前中文分词难点主要有三个：

**1、分词标准**：比如人名，在哈工大的标准中姓和名是分开的，但在Hanlp中是合在一起的。这需要根据不同的需求制定不同的分词标准。

**2、歧义**：对同一个待切分字符串存在多个分词结果。

歧义又分为组合型歧义、交集型歧义和真歧义三种类型。

1) 组合型歧义：分词是有不同的粒度的，指某个词条中的一部分也可以切分为一个独立的词条。比如“中华人民共和国”，粗粒度的分词就是“中华人民共和国”，细粒度的分词可能是“中华/人民/共和国”

2) 交集型歧义：在“郑州天和服装厂”中，“天和”是厂名，是一个专有词，“和服”也是一个词，它们共用了“和”字。

3) 真歧义：本身的语法和语义都没有问题, 即便采用人工切分也会产生同样的歧义，只有通过上下文的语义环境才能给出正确的切分结果。例如：对于句子“美国会通过对台售武法案”，既可以切分成“美国/会/通过对台售武法案”，又可以切分成“美/国会/通过对台售武法案”。

**一般在搜索引擎中**，构建索引时和查询时会使用不同的分词算法。**常用的方案是，在索引的时候使用细粒度的分词以保证召回，在查询的时候使用粗粒度的分词以保证精度。**

**3、新词**：也称未被词典收录的词，该问题的解决依赖于人们对分词技术和汉语语言结构的进一步认识。

另外，我们收集了如下**部分分词工具**，供参考：

中科院计算所NLPIR http://ictclas.nlpir.org/nlpir/

ansj分词器https://github.com/NLPchina/ansj\_seg

哈工大的LTP https://github.com/HIT-SCIR/ltp

清华大学THULAC https://github.com/thunlp/THULAC

斯坦福分词器https://nlp.stanford.edu/software/segmenter.shtml

Hanlp分词器https://github.com/hankcs/HanLP

结巴分词https://github.com/yanyiwu/cppjieba

KCWS分词器(字嵌入+Bi-LSTM+CRF) https://github.com/koth/kcws

ZParhttps://github.com/frcchang/zpar/releases

IKAnalyzer https://github.com/wks/ik-analyzer

**以及部分分词器的简单说明：**

**哈工大的分词器**：主页上给过调用接口，每秒请求的次数有限制。

**清华大学THULAC**：目前已经有Java、Python和C++版本，并且代码开源。

**斯坦福分词器**：作为众多斯坦福自然语言处理中的一个包，目前最新版本3.7.0， Java实现的CRF算法。可以直接使用训练好的模型，也提供训练模型接口。

**Hanlp分词**：求解的是最短路径。优点：开源、有人维护、可以解答。原始模型用的训练语料是人民日报的语料，当然如果你有足够的语料也可以自己训练。

**结巴分词工具**：基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)；采用了动态规划查找最大概率路径, 找出基于词频的最大切分组合；对于未登录词，采用了基于汉字成词能力的 HMM 模型，使用了 Viterbi 算法。**字嵌入+Bi-LSTM+CRF分词器**：本质上是序列标注，这个分词器用人民日报的80万语料，据说按照字符正确率评估标准能达到97.5%的准确率，各位感兴趣可以去看看。

**ZPar分词器**：新加坡科技设计大学开发的中文分词器，包括分词、词性标注和Parser，支持多语言，据说效果是公开的分词器中最好的，C++语言编写。

**关于速度：**

由于分词是基础组件，其性能也是关键的考量因素。通常，分词速度跟系统的软硬件环境有相关外，还与**词典的结构设计和算法复杂度**相关。比如我们之前跑过字嵌入+Bi-LSTM+CRF分词器，其速度相对较慢。另外，开源项目https://github.com/ysc/cws\_evaluation曾对多款分词器速度和效果进行过对比，可供大家参考。

# 什么是熵(Entropy)

简单来说，熵是表示物质系统状态的一种度量，用它老表征系统的无序程度。熵越大，系统越无序，意味着系统结构和运动的不确定和无规则；反之，，熵越小，系统越有序，意味着具有确定和有规则的运动状态。熵的中文意思是热量被温度除的商。负熵是物质系统有序化，组织化，复杂化状态的一种度量。

熵最早来原于**物理学**. 德国物理学家鲁道夫·克劳修斯首次提出熵的概念，用来表示任何一种能量在空间中分布的均匀程度，能量分布得越均匀，熵就越大。  
一滴墨水滴在清水中，部成了一杯淡蓝色溶液  
热水晾在空气中，热量会传到空气中，最后使得温度一致

更多的一些生活中的例子:  
熵力的一个例子是耳机线，我们将耳机线整理好放进口袋，下次再拿出来已经乱了。让耳机线乱掉的看不见的“力”就是熵力，耳机线喜欢变成更混乱。  
熵力另一个具体的例子是弹性力。一根弹簧的力，就是熵力。 胡克定律其实也是一种熵力的表现。  
万有引力也是熵力的一种(热烈讨论的话题)。  
浑水澄清[1]

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450265483975.png" \t "_blank)

于是从微观看，熵就表现了这个系统所处状态的**不确定性程度**。香农，描述一个信息系统的时候就借用了熵的概念，这里熵表示的是这个信息系统的**平均信息量**(\*\*\*\*平均不确定程度\*\*\*\*)。

# 最大熵模型

我们在投资时常常讲不要把所有的鸡蛋放在一个篮子里，这样可以降低风险。在信息处理中，这个原理同样适用。在数学上，这个原理称为最大熵原理(the maximum entropy principle)。

让我们看一个拼音转汉字的简单的例子。假如输入的拼音是"wang-xiao-bo"，利用语言模型，根据有限的上下文(比如前两个词)，我们能给出两个最常见的名字“王小波”和“王晓波 ”。至于要唯一确定是哪个名字就难了，即使利用较长的上下文也做不到。当然，我们知道如果通篇文章是介绍文学的，作家王小波的可能性就较大；而在讨论两岸关系时，台湾学者王晓波的可能性会较大。在上面的例子中，我们只需要综合两类不同的信息，即主题信息和上下文信息。虽然有不少凑合的办法，比如：分成成千上万种的不同的主题单独处理，或者对每种信息的作用加权平均等等，但都不能准确而圆满地解决问题，这样好比以前我们谈到的行星运动模型中的小圆套大圆打补丁的方法。在很多应用中，我们需要综合几十甚至上百种不同的信息，这种小圆套大圆的方法显然行不通。

数学上最漂亮的办法是最大熵(maximum entropy)模型，它相当于行星运动的椭圆模型。“最大熵”这个名词听起来很深奥，但是它的原理很简单，我们每天都在用。说白了，就是要保留全部的不确定性，将风险降到最小。

回到我们刚才谈到的拼音转汉字的例子，我们已知两种信息，第一，根据语言模型，wangxiao-bo可以被转换成王晓波和王小波；第二，根据主题，王小波是作家，《黄金时代》的作者等等，而王晓波是台湾研究两岸关系的学者。因此，我们就可以建立一个最大熵模型，同时满足这两种信息。现在的问题是，这样一个模型是否存在。匈牙利著名数学家、信息论最高奖香农奖得主希萨（Csiszar）证明，对任何一组不自相矛盾的信息，这个最大熵模型不仅存在，而且是唯一的。而且它们都有同一个非常简单的形式 -- 指数函数。下面公式是根据上下文（前两个词）和主题预测下一个词的最大熵模型，其中 w3 是要预测的词（王晓波或者王小波）w1 和 w2 是它的前两个字（比如说它们分别是“出版”，和“”），也就是其上下文的一个大致估计，subject 表示主题。

[640\_wx\_fmt=png&tp=webp&wxfrom=5.webp](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450278762003.jpg" \t "_blank)

我们看到，在上面的公式中，有几个参数 lambda 和 Z ，他们需要通过观测数据训练出来。最大熵模型在形式上是最漂亮的统计模型，而在实现上是最复杂的模型之一。

我们上次谈到用最大熵模型可以将各种信息综合在一起。我们留下一个问题没有回答，就是如何构造最大熵模型。我们已经所有的最大熵模型都是指数函数的形式，现在只需要确定指数函数的参数就可以了，这个过程称为模型的训练。

最原始的最大熵模型的训练方法是一种称为通用迭代算法 GIS(generalized iterative scaling) 的迭代 算法。GIS 的原理并不复杂，大致可以概括为以下几个步骤：

1. 假定第零次迭代的初始模型为等概率的均匀分布。
2. 用第 N 次迭代的模型来估算每种信息特征在训练数据中的分布，如果超过了实际的，就把相应的模型参数变小；否则，将它们便大。
3. 重复步骤 2 直到收敛。

GIS 最早是由 Darroch 和 Ratcliff 在七十年代提出的。但是，这两人没有能对这种算法的物理含义进行很好地解释。后来是由数学家希萨（Csiszar)解释清楚的，因此，人们在谈到这个算法时，总是同时引用 Darroch 和Ratcliff 以及希萨的两篇论文。GIS 算法每次迭代的时间都很长，需要迭代很多次才能收敛，而且不太稳定，即使在 64 位计算机上都会出现溢出。因此，在实际应用中很少有人真正使用 GIS。大家只是通过它来了解最大熵模型的算法。 八十年代，很有天才的孪生兄弟的达拉皮垂(Della Pietra)在 IBM 对 GIS 算法进行了两方面的改进，提出了改进迭代算法 IIS（improved iterative scaling）。这使得最大熵模型的训练时间缩短了一到两个数量级。这样最大熵模型才有可能变得实用。即使如此，在当时也只有 IBM 有条件是用最大熵模型。

由于最大熵模型在数学上十分完美，对科学家们有很大的诱惑力，因此不少研究者试图把自己的问题用一个类似最大熵的近似模型去套。谁知这一近似，最大熵模型就变得不完美了，结果可想而知，比打补丁的凑合的方法也好不了多少。于是，不少热心人又放弃了这种方法。第一个在实际信息处理应用中验证了最大熵模型的优势的，是宾夕法尼亚大学马库斯的另一个高徒原 IBM 现微软的研究员拉纳帕提(Adwait Ratnaparkhi)。拉纳帕提的聪明之处在于他没有对最大熵模型进行近似，而是找到了几个最适合用最大熵模型、而计算量相对不太大的自然语言处理问题，比如词性标注和句法分析。拉纳帕提成功地将上下文信息、词性（名词、动词和形容词等）、句子成分（主谓宾）通过最大熵模型结合起来，做出了当时世界上最好的词性标识系统和句法分析器。拉纳帕提的论文发表后让人们耳目一新。拉纳帕提的词性标注系统，至今仍然是使用单一方法最好的系统。科学家们从拉纳帕提的成就中，又看到了用最大熵模型解决复杂的文字信息处理的希望。

但是，最大熵模型的计算量仍然是个拦路虎。我在学校时花了很长时间考虑如何简化最大熵模型的计算量。终于有一天，我对我的导师说，我发现一种数学变换，可以将大部分最大熵模型的训练时间在 IIS 的基础上减少两个数量级。我在黑板上推导了一个多小时，他没有找出我的推导中的任何破绽，接着他又回去想了两天，然后告诉我我的算法是对的。从此，我们就建造了一些很大的最大熵模型。这些模型比修修补补的凑合的方法好不少。即使在我找到了快速训练算法以后，为了训练一个包含上下文信息，主题信息和语法信息的文法模型(language model)，我并行使用了20 台当时最快的 SUN 工作站，仍然计算了三个月。由此可见最大熵模型的复杂的一面。

最大熵模型，可以说是集简与繁于一体，形式简单，实现复杂。值得一提的是，在Google的很多产品中，比如机器翻译，都直接或间接地用到了最大熵模型。 讲到这里，读者也许会问，当年最早改进最大熵模型算法的达拉皮垂兄弟这些年难道没有做任何事吗？他们在九十年代初贾里尼克离开 IBM 后，也退出了学术界，而到在金融界大显身手。他们两人和很多 IBM 语音识别的同事一同到了一家当时还不大，但现在是世界上最成功对冲基金(hedge fund)公司----文艺复兴技术公司 (Renaissance Technologies)。我们知道，决定股票涨落的因素可能有几十甚至上百种，而最大熵方法恰恰能找到一个同时满足成千上万种不同条件的模型。达拉皮垂兄弟等科学家在那里，用于最大熵模型和其他一些先进的数学工具对股票预测，获得了巨大的成功。从该基金 1988 年创立至今，它的净回报率高达平均每年 34%。也就是说，如果 1988 年你在该基金投入一块钱，今天你能得到 200 块钱。这个业绩，远远超过股神巴菲特的旗舰公司伯克夏哈撒韦（Berkshire Hathaway)。同期，伯克夏哈撒韦的总回报是 16 倍。 值得一提的是，信息处理的很多数学手段，包括隐含马尔可夫模型、子波变换、贝叶斯网络等等，在华尔街多有直接的应用。由此可见，数学模型的作用。

# HMM（隐马尔可夫模型）

**隐马尔可夫模型**（Hidden Markov Model，HMM）是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析，例如模式识别。  
是在被建模的系统被认为是一个马尔可夫过程与未观测到的（隐藏的）的状态的统计马尔可夫模型。

下面用一个简单的例子来阐述：  
假设我手里有三个不同的骰子。第一个骰子是我们平常见的骰子（称这个骰子为D6），6个面，每个面（1，2，3，4，5，6）出现的概率是1/6。第二个骰子是个四面体（称这个骰子为D4），每个面（1，2，3，4）出现的概率是1/4。第三个骰子有八个面（称这个骰子为D8），每个面（1，2，3，4，5，6，7，8）出现的概率是1/8。

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450305163828.png" \t "_blank)

假设我们开始掷骰子，我们先从三个骰子里挑一个，挑到每一个骰子的概率都是1/3。然后我们掷骰子，得到一个数字，1，2，3，4，5，6，7，8中的一个。不停的重复上述过程，我们会得到一串数字，每个数字都是1，2，3，4，5，6，7，8中的一个。例如我们可能得到这么一串数字（掷骰子10次）：1 6 3 5 2 7 3 5 2 4  
这串数字叫做可见状态链。但是在隐马尔可夫模型中，我们不仅仅有这么一串可见状态链，还有一串隐含状态链。在这个例子里，这串隐含状态链就是你用的骰子的序列。比如，隐含状态链有可能是：D6 D8 D8 D6 D4 D8 D6 D6 D4 D8

一般来说，HMM中说到的马尔可夫链其实是指隐含状态链，因为隐含状态（骰子）之间存在转换概率（transition probability）。在我们这个例子里，D6的下一个状态是D4，D6，D8的概率都是1/3。D4，D8的下一个状态是D4，D6，D8的转换概率也都一样是1/3。这样设定是为了最开始容易说清楚，但是我们其实是可以随意设定转换概率的。比如，我们可以这样定义，D6后面不能接D4，D6后面是D6的概率是0.9，是D8的概率是0.1。这样就是一个新的HMM。

同样的，尽管可见状态之间没有转换概率，但是隐含状态和可见状态之间有一个概率叫做输出概率（emission probability）。就我们的例子来说，六面骰（D6）产生1的输出概率是1/6。产生2，3，4，5，6的概率也都是1/6。我们同样可以对输出概率进行其他定义。比如，我有一个被赌场动过手脚的六面骰子，掷出来是1的概率更大，是1/2，掷出来是2，3，4，5，6的概率是1/10。

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450321576527.png" \t "_blank)

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450339544712.png" \t "_blank)

其实对于HMM来说，如果提前知道所有隐含状态之间的转换概率和所有隐含状态到所有可见状态之间的输出概率，做模拟是相当容易的。但是应用HMM模型时候呢，往往是缺失了一部分信息的，有时候你知道骰子有几种，每种骰子是什么，但是不知道掷出来的骰子序列；有时候你只是看到了很多次掷骰子的结果，剩下的什么都不知道。如果应用算法去估计这些缺失的信息，就成了一个很重要的问题。这些算法我会在下面详细讲。

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××× 如果你只想看一个简单易懂的例子，就不需要往下看了。×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

说两句废话，答主认为呢，要了解一个算法，要做到以下两点：会其意，知其形。答主回答的，其实主要是第一点。但是这一点呢，恰恰是最重要，而且很多书上不会讲的。正如你在追一个姑娘，姑娘对你说“你什么都没做错！”你要是只看姑娘的表达形式呢，认为自己什么都没做错，显然就理解错了。你要理会姑娘的意思，“你赶紧给我道歉！”这样当你看到对应的表达形式呢，赶紧认错，跪地求饶就对了。数学也是一样，你要是不理解意思，光看公式，往往一头雾水。不过呢，数学的表达顶多也就是晦涩了点，姑娘的表达呢，有的时候就完全和本意相反。所以答主一直认为理解姑娘比理解数学难多了。

回到正题，和HMM模型相关的算法主要分为三类，分别解决三种问题：

\*\* 1）知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道每次掷出来的都是哪种骰子（隐含状态链）。\*\*  
这个问题呢，在语音识别领域呢，叫做解码问题。这个问题其实有两种解法，会给出两个不同的答案。每个答案都对，只不过这些答案的意义不一样。第一种解法求最大似然状态路径，说通俗点呢，就是我求一串骰子序列，这串骰子序列产生观测结果的概率最大。第二种解法呢，就不是求一组骰子序列了，而是求每次掷出的骰子分别是某种骰子的概率。比如说我看到结果后，我可以求得第一次掷骰子是D4的概率是0.5，D6的概率是0.3，D8的概率是0.2.第一种解法我会在下面说到，但是第二种解法我就不写在这里了，如果大家有兴趣，我们另开一个问题继续写吧。

**2）还是知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道掷出这个结果的概率。**   
看似这个问题意义不大，因为你掷出来的结果很多时候都对应了一个比较大的概率。问这个问题的目的呢，其实是检测观察到的结果和已知的模型是否吻合。如果很多次结果都对应了比较小的概率，那么就说明我们已知的模型很有可能是错的，有人偷偷把我们的骰子給换了。

**3）知道骰子有几种（隐含状态数量），不知道每种骰子是什么（转换概率），观测到很多次掷骰子的结果（可见状态链），我想反推出每种骰子是什么（转换概率）。**   
这个问题很重要，因为这是最常见的情况。很多时候我们只有可见结果，不知道HMM模型里的参数，我们需要从可见结果估计出这些参数，这是建模的一个必要步骤。

问题阐述完了，下面就开始说解法。（0号问题在上面没有提，只是作为解决上述问题的一个辅助）

**0.一个简单问题**  
其实这个问题实用价值不高。由于对下面较难的问题有帮助，所以先在这里提一下。 知道骰子有几种，每种骰子是什么，每次掷的都是什么骰子，根据掷骰子掷出的结果，求产生这个结果的概率。

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450350489511.png" \t "_blank)

解法无非就是概率相乘：

**1.看见不可见的，破解骰子序列** 这里我说的是第一种解法，解最大似然路径问题。 举例来说，我知道我有三个骰子，六面骰，四面骰，八面骰。我也知道我掷了十次的结果（1 6 3 5 2 7 3 5 2 4），我不知道每次用了那种骰子，我想知道最有可能的骰子序列。  
其实最简单而暴力的方法就是穷举所有可能的骰子序列，然后依照第零个问题的解法把每个序列对应的概率算出来。然后我们从里面把对应最大概率的序列挑出来就行了。如果马尔可夫链不长，当然可行。如果长的话，穷举的数量太大，就很难完成了。 另外一种很有名的算法叫做Viterbi algorithm. 要理解这个算法，我们先看几个简单的列子。 首先，如果我们只掷一次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450361262083.png" \t "_blank)

看到结果为1.对应的最大概率骰子序列就是D4，因为D4产生1的概率是1/4，高于1/6和1/8. 把这个情况拓展，我们掷两次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450369701425.png" \t "_blank)

结果为1，6.这时问题变得复杂起来，我们要计算三个值，分别是第二个骰子是D6，D4，D8的最大概率。显然，要取到最大概率，第一个骰子必须为D4。这时，第二个骰子取到D6的最大概率是

同样的，我们可以计算第二个骰子是D4或D8时的最大概率。我们发现，第二个骰子取到D6的概率最大。而使这个概率最大时，第一个骰子为D4。所以最大概率骰子序列就是D4 D6。 继续拓展，我们掷三次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450384543938.png" \t "_blank)

同样，我们计算第三个骰子分别是D6，D4，D8的最大概率。我们再次发现，要取到最大概率，第二个骰子必须为D6。这时，第三个骰子取到D4的最大概率是

同上，我们可以计算第三个骰子是D6或D8时的最大概率。我们发现，第三个骰子取到D4的概率最大。而使这个概率最大时，第二个骰子为D6，第一个骰子为D4。所以最大概率骰子序列就是D4 D6 D4。  
写到这里，大家应该看出点规律了。既然掷骰子一二三次可以算，掷多少次都可以以此类推。我们发现，我们要求最大概率骰子序列时要做这么几件事情。首先，不管序列多长，要从序列长度为1算起，算序列长度为1时取到每个骰子的最大概率。然后，逐渐增加长度，每增加一次长度，重新算一遍在这个长度下最后一个位置取到每个骰子的最大概率。因为上一个长度下的取到每个骰子的最大概率都算过了，重新计算的话其实不难。当我们算到最后一位时，就知道最后一位是哪个骰子的概率最大了。然后，我们要把对应这个最大概率的序列从后往前推出来。  
**2.谁动了我的骰子？** 比如说你怀疑自己的六面骰被赌场动过手脚了，有可能被换成另一种六面骰，这种六面骰掷出来是1的概率更大，是1/2，掷出来是2，3，4，5，6的概率是1/10。你怎么办么？答案很简单，算一算正常的三个骰子掷出一段序列的概率，再算一算不正常的六面骰和另外两个正常骰子掷出这段序列的概率。如果前者比后者小，你就要小心了。 比如说掷骰子的结果是：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450402987108.png" \t "_blank)

要算用正常的三个骰子掷出这个结果的概率，其实就是将所有可能情况的概率进行加和计算。同样，简单而暴力的方法就是把穷举所有的骰子序列，还是计算每个骰子序列对应的概率，但是这回，我们不挑最大值了，而是把所有算出来的概率相加，得到的总概率就是我们要求的结果。这个方法依然不能应用于太长的骰子序列（马尔可夫链）。 我们会应用一个和前一个问题类似的解法，只不过前一个问题关心的是概率最大值，这个问题关心的是概率之和。解决这个问题的算法叫做前向算法（forward algorithm）。 首先，如果我们只掷一次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450412515950.png" \t "_blank)

看到结果为1.产生这个结果的总概率可以按照如下计算，总概率为0.18：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450427046706.png" \t "_blank)

把这个情况拓展，我们掷两次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450442206677.png" \t "_blank)

看到结果为1，6.产生这个结果的总概率可以按照如下计算，总概率为0.05：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450450328560.png" \t "_blank)

继续拓展，我们掷三次骰子：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450469549473.png" \t "_blank)

看到结果为1，6，3.产生这个结果的总概率可以按照如下计算，总概率为0.03：

[image](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450499388728.png" \t "_blank)

同样的，我们一步一步的算，有多长算多长，再长的马尔可夫链总能算出来的。用同样的方法，也可以算出不正常的六面骰和另外两个正常骰子掷出这段序列的概率，然后我们比较一下这两个概率大小，就能知道你的骰子是不是被人换了。

# Viterbi algorithm

HMM（隐马尔可夫模型）是用来描述隐含未知参数的统计模型，举一个经典的例子：一个东京的朋友每天根据天气{下雨，天晴}决定当天的活动{公园散步,购物,清理房间}中的一种，我每天只能在twitter上看到她发的推“啊，我前天公园散步、昨天购物、今天清理房间了！”，那么我可以根据她发的推特推断东京这三天的天气。在这个例子里，显状态是活动，隐状态是天气。  
任何一个HMM都可以通过下列五元组来描述：

:param obs:观测序列

:param states:隐状态

:param start\_p:初始概率（隐状态）

:param trans\_p:转移概率（隐状态）

:param emit\_p: 发射概率 （隐状态表现为显状态的概率）

[6cbb8645gw1egs40a3bpmj208n09574o](https://link.jianshu.com?t=http:/images0.cnblogs.com/blog/133059/201507/161450524231771.jpg" \t "_blank)

伪码如下：

states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start\_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition\_probability = {

'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},

'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},

}

emission\_probability = {

'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},

'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},

}

求解最可能的天气  
求解最可能的隐状态序列是HMM的三个典型问题之一，通常用维特比算法解决。维特比算法就是求解HMM上的最短路径（-log(prob)，也即是最大概率）的算法。  
稍微用中文讲讲思路，很明显，第一天天晴还是下雨可以算出来：  
定义V[时间][今天天气] = 概率，注意今天天气指的是，前几天的天气都确定下来了（概率最大）今天天气是X的概率，这里的概率就是一个累乘的概率了。

因为第一天我的朋友去散步了，所以第一天下雨的概率V[第一天][下雨] = 初始概率[下雨] \* 发射概率[下雨][散步] = 0.6 \* 0.1 = 0.06，同理可得V[第一天][天晴] = 0.24 。从直觉上来看，因为第一天朋友出门了，她一般喜欢在天晴的时候散步，所以第一天天晴的概率比较大，数字与直觉统一了。

从第二天开始，对于每种天气Y，都有前一天天气是X的概率 \* X转移到Y的概率 \* Y天气下朋友进行这天这种活动的概率。因为前一天天气X有两种可能，所以Y的概率有两个，选取其分别解决三种问题中较大一个作为V[第二天][天气Y]的概率，同时将今天的天气加入到结果序列中

比较V[最后一天][下雨]和[最后一天][天晴]的概率，找出较大的哪一个对应的序列，就是最终结果。

算法的代码可以在github上看到，地址为：  
[https://github.com/hankcs/Viterbi](https://link.jianshu.com?t=https:/github.com/hankcs/Viterbi)  
运行完成后根据Viterbi得到结果：

Sunny Rainy Rainy

Viterbi被广泛应用到分词，词性标注等应用场景。

作者：Detailscool  
链接：https://www.jianshu.com/p/3a624de8b4c6  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。