Query 分词（切词）

分词指将一段连续的文本切成一个个独立且有意义的词汇，在文本召回中会对 Doc 文本内容分词以构建索引，并通过对查询词 Query 分词后去做检索。Query 分词在搜索中是一个基础信号，除了文本召回，在词权重、Term 同义改写中都会用到分词信号。

业界常用的分词方案有词典匹配和模型标注，其中，词典匹配需要解决新词发现（构建词典）、词匹配、以及路径消歧问题等，模型标注则是解决序列标注问题。此外分词粒度标准和分词评价指标是指导分词技术方向的重要因素。

在工业界，分词作为基础信号，对其性能要求较高，线上通常会采用轻量级的分词方案（词典 + CRF），且这种经典方案一般能达到较高技术指标，不断完善优化词典将决定分词效果的上限。此外，依赖分词的下游服务数量非常庞大，造成分词的维护成本较高，所以通常分词的基本流程完成搭建后，优化的重点就是新词发现以更新词典，考虑到 ROI，这个周期往往是半年到一年，

本章节将先介绍分词常用评价指标，然后对词典匹配和模型标注中的技术方案做介绍。

分词评价指标

和大多数任务一样，分词的评价指标主要是精确率（Precision）、召回率（Recall）和 F1 值，此外还有 OOV Recall Rate（未登录词召回率）和 IV Recall Rate（登录词召回率）。

精确率

- $\mathbf{Precision} = \frac{\mathbf{TP}}{\mathbf{TP}+\mathbf{FP}}$
- TP为预测为P且答案为P，即正确切分出的词语数目
- FP为预测是P且答案是N，即误切分出的词语数目

召回率

- $\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP}+\mathbf{FN}}$
- FN为预测是N且答案是P，即漏切分的词语数目

F1

$\mathbf{F1} = 2 \times \frac{\mathbf{Precision} \times \mathbf{Recall}}{\mathbf{Precision} + \mathbf{Recall}}$

例：

- 标准答案：分词 评价 指标 精确率
- 分词结果：分词 评价 指标 精确 率
- $\mathbf{Precision} = \frac{\mathbf{3}}{\mathbf{3}+\mathbf{2}} = \frac{3}{5}$
- $\mathbf{Recall} = \frac{\mathbf{3}}{\mathbf{3}+\mathbf{1}} = \frac{3}{4}$
- $\mathbf{F1} = \frac{2}{3}$

OOV Recall Rate

OOV（Out Of Vocabulary）“未登录词”，即新词或已知词典中不存在的词，其衡量了分词工具在识别未登录词方面的能力。计算方法：计算正确分词结果中所有未登录词的个数（作为分母），计算基于当前分词方法得到的分词结果中分词正确部分中未登录词的个数（作为分子）。

IV Recall Rate

IV（In Vocabulary）“登录词”，即已经存在在字典中的词，其衡量了词典中的词汇被正确召回的概率。分母和分子分别为所有正确分词结果中所有登录词的个数和基于当前分词方法得到的分词结果分词正确部分中登录词的个数。

词典匹配分词

最长匹配分词

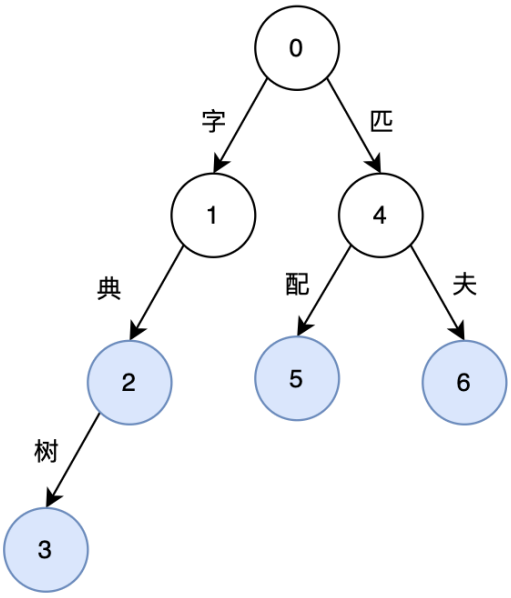
当词典确定时，则需要定义一套匹配规则完成对给定文本分词。考虑到越长的词表达的含义越准确和完整，如：“机器学习”通常作为一个整体，而不是切分为“机器”和“学习”，于是定义词越长则优先级越高，即在对给定文本遍历查词典的过程中，优先输出更长的词，这就是最长匹配分词。具体的：

- **正向最大匹配**：从左向右扫描文本，每次尝试匹配最长的词汇。如果词典中存在该词汇，则将其切分出来；否则，缩短一位继续匹配，直到找到词典中的词汇或只剩下一个字符为止。
- **逆向最大匹配**：与正向最大匹配相反，从右向左扫描文本进行匹配。这种方法在某些场景下可能比正向最大匹配更准确。
- **双向最大匹配**：结合正向和逆向最大匹配的优点，同时进行两次扫描，然后选择切分结果中词数较少的那个作为最终的分词结果。当词数也相同时，优先返回逆向最长匹配的结果。

最长匹配分词实现简单且运行速度快，但是对未登录词的处理能力有限，且对歧义词的消解不好，无法根据语境来解决歧义问题。

字典树和AC自动机

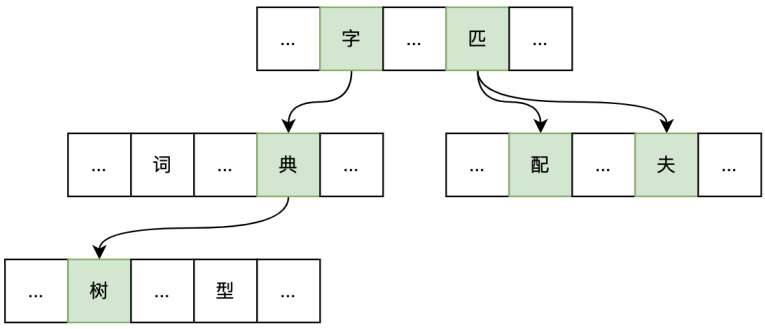
为了快速实现词匹配，字典树（Trie树、前缀树）作为多叉树结构拥有高效的搜索效率而被广泛使用。字典树将字符串视为从根节点到某个节点之间的一条路径，在检索过程中，只需沿着字典树的边进行匹配，直到找到对应的节点或确定不存在该字符串。字典树利用字符串的公共前缀来节省存储空间，具有相同前缀的字符串在字典树中共享相同的路径。字典树本质上是一个有限状态自动机，正常情况下，其插入、查找、删除的时间复杂度理论上仅与检索词长度有关，即：O(n)，n 为检索词长度。



如图，数字为路径编号，蓝色节点表示结尾，“字典树”的路径可以表示为0-1-2-3。

标准Trie

标准Trie的每个节点代表一个字符串的前缀，且由指针数组存储。以英文字典树为例，每个节点通常有27个指针，分别是26个英文字母和一个终止符，即每个节点指向下一个状态。由于要求链路中，每个数组节点逐层指向的每一层都是等长度的数组，会在实际应用中造成大量数组指针空置（公共前缀通常比节点数量少，即大多数节点没有子节点），另外对于中文来说，单字数量远超英文的26个字母，其内存的指数膨胀将极其恐怖，所以一个变通的方法是仅对根节点实现数组散列。



Double-Array Trie

Double-Array Trie 将Trie树结构压缩并存储在两个数组中（base 和 check），如此减少内存使用并提高查询速度。下面我们通过构建双数组字典树的方式来理解：

前置知识：

- DAT使用两个数组（base和check）来存储Trie树的状态和转移信息
- 每个状态通常用一个整数表示，这个整数是base和check数组的索引
- base数组的每个元素表示一个树节点，即状态
- check数组的每个元素表示某个状态的前驱状态
- 给定当前状态的下标 `s` 和输入字符的编码 `c`，下一个状态的下标 `t` 可以通过计算 `t = base[s] + c` 得到
- 使用check数组来验证状态转移的有效性，如果 `check[t] = s`，则表示从状态 `s` 到状态 `t` 的转移是有效的

- 当前状态 = `base[s]`
- 转移下标 = `base[s] + code(字符)`
- 转移基数 = 前驱节点转移基数
- 当转移下标触发冲突时（下标位置已被占用）：
    - 转移下标向后移到空位
    - 重新计算前驱转移基数 = 当前转移下标 - code(字符)
    - 遍历前驱节点下的子节点，检查其转移基数更新后是否引起冲突，是则返回上一步计算，并更新所有节点的转移基数

Base Array 构造：

1. 给定词：“字典树”、“匹配”、“匹夫”，以及字符编码code：[字-1，典-2，树-3，匹-4，配-5，夫-6]
2. 依次处理“字典树”、“匹配”每个词首字，然后处理每个词的第二个字...直到所有词的尾字：
    1. 初始化根节点的base转移基数为`1`，check值为`-1`
    2. 依次将“字”、“匹”写入数组，位置为`base[0] + code(字符)`，每个位置的转移基数为上一个状态的转移基数
        - “字”：
            - 当前状态 = 根节点状态 = base[0] = 1
            - 转移下标 = base[0] + code(字) = 2
            - 前驱转移基数 = base[0] = 1
            - 转移基数 = 前驱转移基数 = base[0] = 1
            - check = 前驱位置 = 0
        - “匹”：
            - 当前状态 = 根节点状态 = base[0] = 1
            - 转移下标 = base[0] + code(匹) = 5
            - 前驱转移基数 = base[0] = 1
            - 转移基数 = 前驱转移基数 = base[0] = 1
            - check = 前驱位置 = 0

			字			匹		
index	0	1	2	3	4	5	6	7
base	1		1			1		
check	-1		0			0		

    3. 依次将“典”、“配”、“夫”写入数组

- “字典”：
  - 当前状态 = “字”节点状态 = base[0] + code(字) = 2
  - 转移下标 = base[2] + code(典) = 1 + 2 = 3
  - 前驱转移基数 = base[2] = 1
  - 转移基数 = 前驱转移基数 = base[2] = 1
  - check = 前驱位置 = 2
- “匹配”：
  - 当前状态 = “匹”节点状态 = base[0] + code(匹) = 5
  - 转移下标 = base[5] + code(配) = 1 + 5 = 6
  - 前驱转移基数 = base[5] = 1
  - 转移基数 = 前驱转移基数 = base[5] = 1
  - check = 前驱位置 = 5
- “匹夫”：
  - 当前状态 = “匹”节点状态 = base[0] + code(匹) = 5
  - 转移下标 = base[5] + code(夫) = 1 + 6 = 7
  - 前驱转移基数 = base[5] = 1
  - 转移基数 = 前驱转移基数 = base[5] = 1
  - check = 前驱位置 = 5

			字	字典		匹	匹配	匹夫
index	0	1	2	3		5	6	7
base	1		1	1		1	1	1
check	-1		0	2		0	5	5

4. 将“树”写入数组
   - “字典树”：
     - 当前状态 = “字典”节点状态 = base[base[0] + code(字)] + code(典) = 3
     - 转移下标 = base[3] + code(树) = 1 + 3 = 4
     - 前驱转移基数 = base[3] = 1
     - 转移基数 = 前驱转移基数 = base[3] = 1
     - check = 前驱位置 = 3

			字	字典	字典树	匹	匹配	匹夫
index	0	1	2	3	4	5	6	7
base	1		1	1	1	1	1	1
check	-1		0	2	3	0	5	5

3. 检查节点，将叶子节点的转移基础设置为负数

AC自动机

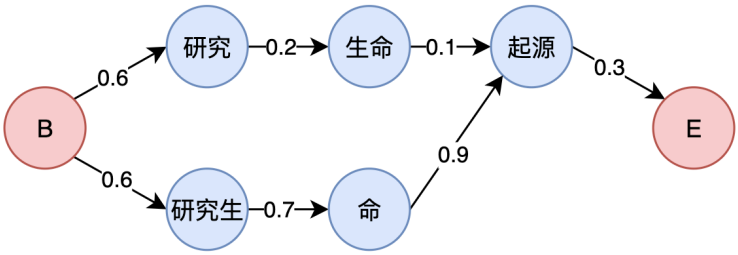
AC自动机结合了字典树（Trie）和KMP算法的思想，其中 KMP 算法是一种用于单模式字符串匹配的算法，其通过预处理模式串来加速匹配过程。AC自动机将多个模式串构建成一个具有状态转移功能的字典树，并在此基础上添加失败指针（fail pointer），使得在匹配失败时能够高效地转移到下一个可能的状态，从而避免了重复的匹配工作。

AC自动机的构建过程主要包括以下两个步骤：

1. **构建字典树**：将所有模式串插入到字典树中。字典树的每个节点代表一个字符，从根节点到任意节点的路径代表一个字符串的前缀
2. **创建失败指针**：失败指针指向字典树中的另一节点，当在某一节点上的字符匹配失败时，算法会通过失败指针跳转到另一节点继续匹配，而不是从头开始。失败指针的构建通常使用广度优先搜索（BFS）算法来实现

最短路径消歧

只依靠词典匹配分词的准确率不高，且无法解决歧义问题，如：“研究 生命 起源” 和 “研究生 命 起源”，词典只能找到所有可能的路径，但无法选择和判断哪种分词是正确的。为了解决这个问题，业界通常采用基于统计的方法，如CRF、HMM等统计模型进行消歧。



如上词图，从起点B到终点E的两条路径代表两种分词方式，语言模型的解码任务就是找到最合理的路径。假设我们已经从大规模语料库统计得到了词节点之间的距离，距离表示两个词之间的关联性，模型的任务就是找到最短的一条路径。

求解词图（由马尔科夫链构成）最短路径通常采用基于动态规划的维特比算法，维特比算法一般用来求解隐马尔科夫模型（HMM）过程最大后验证概率，我们在构建词图的时候会通过CRF/HMM语言模型获得状态转移概率和观测概率，维特比算法将从所有可能的隐藏状态路径中找到一条概率最大的路径，即最有可能产生观测序列的隐藏状态序列。这部分内容将在基于CRF的序列标注中介绍（解决词典分词OOV问题以及NER标注问题），此时我们只需了解工业界在分词路径消歧中通常采用CRF + 规则，规则一般考虑分词后词典命中量情况，另外会维护一个分词黑名单进行过滤。

粗/细粒度

分词粒度决定了召回索引的构建，会影响召回数量和相关性。常见的分词粒度有三种：粗粒度、细粒度、极细粒度（用在索引分词），在构建索引的使用一般要求三种粒度都有以保证召回率。

Query 侧分词

显而易见的，采用粗粒度分词召回 Doc 准确率高，但召回率低；细粒度分词能召回更多的 Doc，但是准确率低，需要处理分词后多个 token 间的关系（词权重、词紧密度等）。所以 Query 侧在分词的时候采用混合粒度，即同时做粗/细粒度分词。

例：Query = 运动裤

- 粗粒度分词：token = [运动裤]
- 细粒度分词：token = [运动, 裤]
- 相关性：
  - 粗粒度需要召回的 Doc 中包含`[运动裤]`，召回 Doc相关性高
  - 细粒度容易召回不相关结果，假设 Doc = “运动鞋搭配牛仔裤”，索引分词 = [运动鞋, **运动**, 鞋, 搭配, 牛仔裤, 牛仔, **裤**]，在细粒度分词`[运动，裤]`下会召回该不相关 Doc
- 召回率：
  - 假设 Doc = “运动长裤”，索引分词 = [**运动**, 长裤, 长, **裤**]，索引分词中不包含`[运动裤]`，该 Doc 无法在粗粒度下被召回，但是在细粒度分词中可被召回

索引侧分词

索引侧分词包括粗粒度、细粒度、极细粒度，其中极细粒度会在细粒度上做更进一步的切分，如 Doc = ”皮鞋“，粗细度的分词都是`[皮鞋]`，如果只对`[皮鞋]`做索引，那么当 Query = [鞋] 的时候将无法召回”皮鞋“，此时需要对 Doc 增加极细粒度分词 = [皮鞋, 皮, 鞋]。

词粒度判别

我们了解了分词的几种粒度，其中细粒度和极细粒度决定了文本倒排召回上限，那么如何判别粒度过粗词？以下是一些可以参考的特征：

- 紧密度
  - 对于两个单词a和b，$count(a)$、$count(b)$和$count(ab)$分别表示词a、词b和词ab在语料中出现的次数，则$PMI = \frac{count(ab)}{count(a)+count(b)}$。这个比值越大，说明$a$和$b$共同出现的概率越高，它们的紧密度也就越大
- 召回 Doc 增量
  - $Recall\ Increment = \frac{count(a\_b)+count(ab)}{count(ab)}$，其中$count(a_b)$表示词$ab$被拆开成$a$和$b$后召回 Doc 数量
- 语法规则
  - 形容词 + 名词，副词 + 动词等需要拆分
- 是否是专有名词
- 是否转义
  - 分词后召回 Doc 类别变化，如统计召回 Doc 类别分布的KL散度

有了上述特征即可通过规则或人工标注数据后训练模型来实现判别词粒度是否过粗。

新词发现

在互联网搜索领域中，尤其是 UGC 内容平台，在这个信息爆炸的时代每个周期都在不断涌现新词汇。这些未登录词中大部分是实体词，如 人名/作品/品牌/专有名词 等，这些未登陆词会被拆分成单字，导致召回结果不相关，所以需要建立一个新词发现流程，从现有的语料库中不断挖掘新词扩充词典。对于搜索平台来说，新词发现的挖掘流程如下：

语料库

- 用户查询短文本Query
- Query、Doc基于分词或NER模型预分词
- Doc中用户添加的标签（tag）
- 百科数据

模型判别

对于挖掘的候选新词，需要建立一个模型判断新词是否构成一个完整的词汇。在设计模型特征的时候，通常从文本特征、统计特征等考虑，以下是常用的特征：

特征
词长度
词中是否包含数字
词中是否包含英文字符
词的ngram命中词典数量
词作为Query被检索数量
词在Query中出现数量（日/周/月/季度/年）
词作为Query被检索后的消费特征（点击率等）
词在Doc中出现数量
词完整出现在Doc中的数量
词拆开出现在Doc中的数量
词作为Doc标签时命中Doc数量
词PMI（根据Query统计/根据Doc统计）
词左右熵（根据Query统计/根据Doc统计）
词所属类别、NER等模型预测信息
词完整和拆开后命中 Doc 的特征（Doc 类别、主题等属性特征）分布变化（KL散度）
词向量

训练样本的构建上，正样本一般是历史词典数据 + 人工标注新词数据；负样本除了标注数据也可以通过规则生成乱码假词，也可以基于统计特征反向构建负样本。 另外在模型选择上可以采用基于Wide & Deep的架构设计。

CRF序列标注

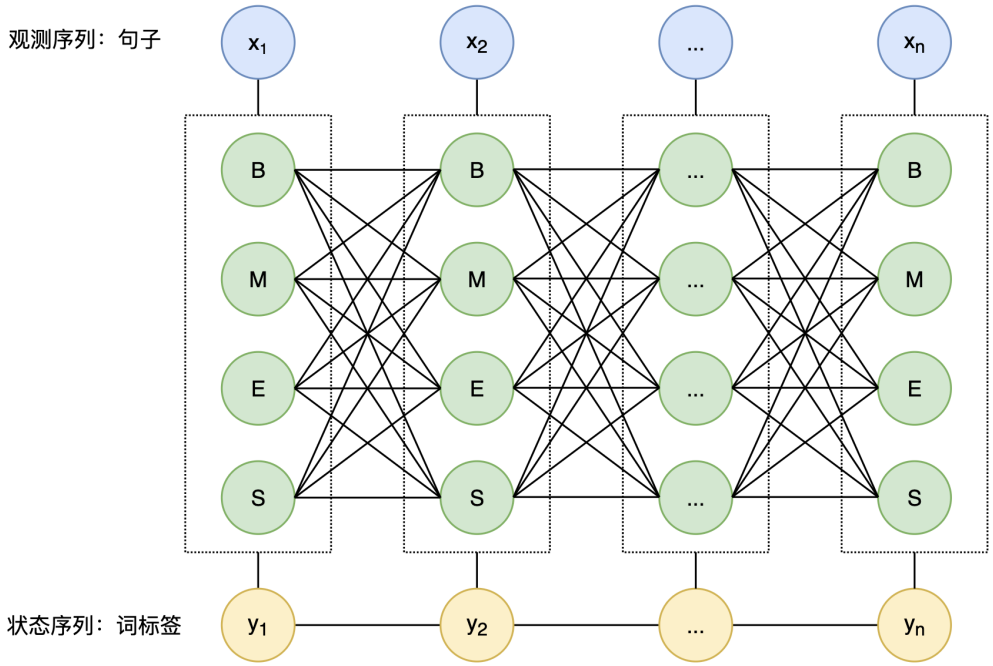
由于通过新词发现扩充词典的流程其迭代周期较长（需要时间积累且词典更新代价较高），线上会采用基于CRF的序列标注实现分词。

为保证分词效果，可以利用词典减小 CRF 解码的搜索空间（如基于字典树匹配构造分词候选，并基于候选直接生成CRF候选标签），并可通过规则约束一步保证 CRF 序列标注的准确率和健壮性（如在型输出标注结果后，可进一步通过粗/细粒度词典控制模型分词粒度，保证Query侧分词和索引侧分词一致性）。

序列标注是指给定文本序列$x = x_1x_2...x_n$，对每个字打上标签$y = y_1y_2...y_n$，对于分词而言，最通用的标签为：B（Begin）、M（Middle）、E（End）、S（Single），分别表示代表词开头、词中间、词结尾、单字成词。

既然可以理解成对字的分类问题，那么很自然的可以想到用DNN对文本序列建模，然后接全连接层用softmax激活。然而，分词序列标注的每个标签有前后依赖性，比如：标签B后只能接M和E，而softmax没有考虑上下文的关联性。于是，我们利用条件随机场（CRF）建模相邻标签之间的关系。

CRF分词可建图如下：



即输入有n个字，每个字有4种标签，CRF序列标注则是在k^n种序列可能中找到正确的一条路径。 CRF是一种判别式模型，它直接对条件概率$P(Y||X)$进行建模，其中X是输入的文本序列（观测序列），Y是对应的标签序列（状态序列）。 特别的，与生成式模型（如隐马尔可夫模型HMM）不同，CRF不需要对联合概率分布$P(X,Y)$进行建模。

$$P(Y||X) = \frac{1}{Z(X)} \exp\left(\sum_{i=1}^{m} \theta_t f_t(y_{i-1}, y_i, X, i) + \sum_{i=1}^{n} \theta_s f_s(y_i, X, i)\right) \quad (1)$$

其中，$Z(X) = \sum_Y exp\left(\sum_{i,m} \theta_t f_t\left(y_{i-1}, y_i, X, i\right) + \sum_{i,n} \theta_s f_s\left(y_i, X, i\right)\right)$

- $Z(X)$是归一化因子，确保概率和为1，需要对所有可能的路径的打分进行指数求和
- θ_t和θ_s是模型参，数
- $f_t(y_{i-1}, y_i, X, i)$是转移特征函数，表示从状态y_{y-1}转移到状态y_i的特征
- $f_s(y_i, X, i)$是状态特征函数，表示位置i处状态y_i的特征

搜索分词中，一个合格的CRF，至少需要百万级的标注数据，CRF根据标注的训练数据和特征模版生成对应特征函数（转移特征函数、状态特征函数），继而通过学习算法在模型空间中找到最拟合训练数据的模型。待CRF模型完成训练确定该马尔科夫随机场的分布后，对数据文本序列解码则需要通过维特比算法找到全局最优解（序列最优路径）。

目前业界通用的CRF工具包是CRF++，其接受纯文本语料，对于中文分词而言，输入的变量仅需字符即可达到较高的准召。此外，BERT + CRF也是业内常用的分词方案。

总结

综上，本章介绍了业内搜索领域分词的常见技术方案，一个好的分词系统需要有一个结合下游应用场景的好的分词标注规范，在这个基础上才能明确分词的优化方向。此外，词粒度与召回能力紧密相关，决定了召回结果的召回量和相关性，如何在其中找到平衡点是词粒度设计的重点。互联网平台不断在汲取、创造新知识，表达方式也在不断发生变化，新词发现流程则是跟紧信息爆炸的时代的重要依靠，分词系统需要不断的维护才能保证技术指标维持在较高水准。

参考文献

1. An Efficient Digital Search Algorithm by Using a Double-Array Structure
2. Conditional random fields: Probabilistic models for segmenting and labeling sequence data
3. Word-Context Character Embeddings for Chinese Word Segmentation
4. Adversarial Multi-Criteria Learning for Chinese Word Segmentation
5. State-of-the-art Chinese Word Segmentation with Bi-LSTMs