





相反的策略：**1. User Agent**: 利用访问网站浏览器发布的 Request Headers 信息中的 User-Agent 信息，判断用户使用何种方式浏览。**应对策略**: 利用 Python 的 Request 库允许用户自定义请求头信息的手段，在请求头信息中将 User-Agent 的值为访问浏览器的请求头标识，从而规避反爬虫机制。

```
headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11; rv:52.0) Gecko/20100101 Firefox/52.0", "Referer": "http://132.2.4.101:8080/"}
print(reqs.get(url, headers=headers))
```

2. **IP/账号访问次数/频率**: 通过限制特定 IP 地址/账号访问频率和次数进行反爬。其本质在于判断浏览行为是否是人类行为。**应对手段**: 1)构造 IP 代理池，然后每次访问时随机选择代理 (有相关服务，通过这种提供免费 IP 的网站来提供代理服务) 2)每次请求行为间隔一段时间 3)注册多个账号以稀释数据集中度 挑战: 账号本身的成本问题、账号被查封的危险
3. **验证码**: 通过各类验证码，判断浏览器属于人类还是机器(从简单的字符识别到 复杂的逻辑推理)**应对手段**: 1)简单的字符识别: 基于机器学习与模式识别相关技术 2)复杂的逻辑推理: 人工辅助编辑 (focus: 验证码是一把双刃剑，在抵御爬虫的同时也在危害用户体验)
4. **动态网页**: 从网页的 url 加载网页的源代码之后，会在浏览器里执行 JavaScript 脚本 (网页内容由脚本加载，而直接抓取只能得到空白页面，此情况在去爬去在线数据库的视频文件时尤为常见)**应对手段**: 核心思路: 模拟浏览器请求 + 使用审查元素窗口 ajax 请求，如知道需要抓取包含数据信息的 json 文件
5. **蜜罐技术**: 网页上会故意留下一些人类看不到或绝对不会点击的链接，由于爬虫会从源代码中获取内容，所以爬虫可能会访问这样的链接，而网站发现有人 IP 访问这个链接，立刻永久封禁访问者，从而难以继续抓取 **应对手段**: 核心思路: 干扰爬虫蜜罐，通过工具库判断网页上的隐含元素，使爬虫避开这些元素，可以部分回避蜜罐的诱导。
6. **用户权限限制**: 不同类型/级别的用户给予不同的内容权限，VIP、SVIP、红钻、绿钻、各种钻…… **应对手段**: 无账号，就可以变强

7. **其他的反爬虫策略**: 不同的网页结构: 每个相同类型的页面的源代码格式均不相同，同样也是双刃剑: 增加爬取难度的同时降低用户浏览体验 (遭反感诟病) + 多模态的呈现方式: 文字转为图像或视频 **应对策略**: OCR、语音识别、图像/视频标注技术

## 网页文本处理

### 一基于字符匹配的方法 (机械式分类)

正向最大匹配分词 (FMM) 双向最大匹配分词 (RMM) 双向最大匹配分词 (BM: FMM+RMM) 最少切分词 (最短路径分词) ASM(d,m,d) 表示匹配方向，+1 为正向，-1 为逆向；a: 每次匹配失败后减/增字符数，+1 为增字，-1 为减字；m: 最小/最大匹配数，+1 为最大匹配，-1 为最小匹配。例如，ASM (+, -, +) 即正向最大匹配分词 (FMM 方法) ； 双向匹配最大匹配数实用案例

FMM: 从左至右尽可能查找最长的词 分词速度较快，但错误率较高

RMM: 从右至左尽可能查找最长的词 统计发现 RMM 分词效果更好 (由于汉语中偏正结构较多，若从后向前匹配，可以适当地避免空白问题)

BM: 综合比较 FMM 与 RMM 两种方法的切分结果，从而选择正确的切分 有助于识别分词中的交叉歧义

最少切分词方法: 使每个字中的词数最少，即 等价于在有向图中搜索最短路径的问题。 将每个字元视作节点，每个词形构成一条边，边权重可视为 0，也可视做词频决定 (尽量取高词频) ，边权重高/频率高后，实际上可视为基于统计的方法的近似 N-最短路径法: 保留 N 条最短路径，以提供更多分词方案 (求一个图中两点的第短 N 路径: Dijkstra (O(n^2))

基于巴配分词方法的优劣 + 优点: 效率高，显现性好 + 缺点: 对词典的依赖性，维护高质量词典需要较大的开支，永远难以应对新生词汇 + 词权重/重要性往往对结果不产生影响

### 二基于统计的分词方法

$$P(w|c) = \max\{P(w_a|c),P(w_b|c),...,P(w_k|c)\}$$

一般化过程: 1. 建立统计语言模型 2. 对句子按不同方案进行分词 3. 计算不同分词方案的概率，选出概率最大的分词结果

理论上，基于统计的分词方法可以不需要词典，但实际应用中第 2 步可以采 用机械式分词方法进行分词，以获得候选的结果集合，既发挥巴配分词分词速度快、效率高的特点，又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

N-gram 模型

N=1，一元文法模型 (最大概率模型)， $P(w) = P(w_1)P(w_2)...P(w_n)$

N=2，Bigram 模型， $P(w) = P(w_1)P(w_2|w_1)...P(w_n|w_{n-1})$

以 Bigram 模型为例

- 首先，构造训练语料库，计算所有的  $C(w_0)$  与  $C(w_1|w_0)$ 。
- 其次，对于每一个可能的分词序列  $w_i$ ，计算以下公式： $P(w) = P(w_1)P(w_2|w_1)...P(w_n|w_{n-1})$
- 最后，返回最大的  $P(w)$  所对应的分词序列作为结果。

其中， $C(w_{i-1}|w_i)$ 指词序列  $w_{i-1}$ 在语料库中出现的次数， $C(w_{i-1}|w_i)$  指词序列  $w_{i-1}$ 在语料库中出现的次数。

当N=1时，N-gram模型退化为一元文法模型，此时词与词之间是独立的。

$$P(W) = P(w_1, w_2, \dots, w_n) \approx P(w_1) \times P(w_2) \times \dots \times P(w_n)$$

$$P(w_i) = \frac{w_i \text{在语料库中的出现次数}N}{\text{语料库中的总词数}N}$$

独立性假设：一元文法

- 基于统计文法模型的优势 + 优点: 减轻了对词典的依赖性 + 然而，这种依赖并非完全消除，取决于性能与效率的平衡
- 如果训练数据机械分词 (匹配合适)，则效率提升但依赖词典 + 如果减少对词典的依赖，则需要更多地遍历潜在的可能 (解空间巨大) 1. 缺点: 依然只有数据中词频的统计对于表示生词或专业词汇不好 + 冷门领域的稀有词汇往往难以准确标注 + 导致标注数据存在偏差 (Bias) 的影响
- 三 **隐马尔可夫模型 (HMM)**
- 隐马尔可夫模型的五个核心要素: 两个集合、三个矩阵、两个集合 + 隐藏集合 (字符集合)、隐藏状态集合 (BEMS)
- 三个矩阵: + 初始状态概率矩阵: 第一个字属于某种隐含状态 (BMS) 的概率 + 隐含状态转移概率矩阵: 各种隐含状态 (各种标签) 之间的转移概率 + 观测状态概率矩阵: 从隐含状态 (标签) 到观测值 (字符) 的转移概率

常用词: Stopwords, 指文档中频繁出现或对实定义影响不大的词语，去掉用原因: + 重复率很高，会造成索引中的倒排表很长，影响查询性能。 + 对最后结果的排序没什么贡献，反而可能产生干扰。

常用的停用词识别方法 + 较为成熟的停用词识别方法有: 文本频率、词频统计、统计计算。 + 更为复杂的算法将统计结合与句法或内容分析。

现代搜索引擎的趋势是逐渐减少对停用词的使用。 + 现代搜索引擎更注重利用语言的统计特性来处理常见词问题。 + 采用压缩技术，降低停用词的存储冗余 +。 + 引入词项压缩，将高频词的影响降至最低。 + 索引去除技术，低于权重的词项将被排除

**归一化处理**: 归一化/词频化，指将单词出现的特殊形式的过程，可以有效降低词频的数量并减少歧义

词干提取: Stemming, 指去除单词前缀/后缀，获得词根的过程。常见的前后缀词有 “复数形式”、“过去分词”、“进行时”等

词形还原: Lemmatization, 指基于词典，将单词的复杂形式转变成最基础的形式。并不是简单地将从后缀去掉，而是会根据词典将单词进行转换

词干提取与词形还原的相同点: 1)目标一致。两者的目标均为将词的不同形态简化或归并为基础形式。2)结果交叉。两者不是互斥关系，其结果有部分交叉。3)方法类似。目前两者主流方法均是借助语料库中的规则词典来实现。

词干提取与词形还原的不同点 1)在原理上，词干提取采用“偷窃”，而词形还原采用“转变”。 2)在复杂性上，词形还原需考虑词缀转化、词性识别等，更为复杂。 3)在实现上，词干提取主要利用规则变化，而词形还原更依赖于词典。 4)在结果上，词干提取不一定能够完整还原词，而词形还原一定是完整单词。

基本干词提取方法: Porter Stemming 基本流程: 1. 去除单词的复数形式 2. 去掉 -ed (ly) 或 -ing (ly) 等后缀 3. 将 -y 改为 -i 4. 处理双重后缀，如 -ization 等 5. 处理 -full, -ness 等后缀 6. 处理 -ant, -ence 等后缀 7. 去掉最后的 -e 和 -ll

网页索引			
倒排表 (Posting List): 文档 ID 列表，列举出项在哪些文档中出现			
关键词 (key)	Document	Term	
优点	维护简单	构造、维护成本高	
缺点	搜索时间长	搜索快	

倒排表三步走: 1.检索每篇文章，获得对，并写入索引表 2. 对临时表中的项进行排序 3. 遍历临时索引，对于相同项的文档 ID 进行合并

基于倒排表的优缺点: 如果两个倒排表的长度分别为 x 和 y，则合并共需 O(x+y)次操作，同时扫描两个倒排表。所需时间与倒排记录的数量呈线性关系

优化改进: 1. AND 是合并操作，先处理文档率较小的，再处理大的 2. 任意位置的布列查寻: 首先，获得所有词的文档频率 + 其次，正向最大匹配分词 (FMM) 双向最大匹配分词 (RMM) 双向最大匹配分词 (BM: FMM+RMM) 最少切分词 (最短路径分词) ASM(d,m,d) 表示匹配方向，+1 为正向，-1 为逆向；a: 每次匹配失败后减/增字符数，+1 为增字，-1 为减字；m: 最小/最大匹配数，+1 为最大匹配，-1 为最小匹配。例如，ASM (+, -, +) 即正向最大匹配分词 (FMM 方法) ； 双向匹配最大匹配数实用案例

FMM: 从左至右尽可能查找最长的词 分词速度较快，但错误率较高

RMM: 从右至左尽可能查找最长的词 统计发现 RMM 分词效果更好 (由于汉语中偏正结构较多，若从后向前匹配，可以适当地避免空白问题)

BM: 综合比较 FMM 与 RMM 两种方法的切分结果，从而选择正确的切分 有助于识别分词中的交叉歧义

最少切分词方法: 使每个字中的词数最少，即 等价于在有向图中搜索最短路径的问题。 将每个字元视作节点，每个词形构成一条边，边权重可视为 0，也可视做词频决定 (尽量取高词频) ，边权重高/频率高后，实际上可视为基于统计的方法的近似 N-最短路径法: 保留 N 条最短路径，以提供更多分词方案 (求一个图中两点的第短 N 路径: Dijkstra (O(n^2))

扩展性优势: 1) 除了文档 ID，搜索引擎往往在倒排表中加入更多元素 (词频/权重、词项类型等) 2) 除了基本的词查询之外，倒排表还可能包含更多需求 (词组查询、词频统计、文档区域)

词频倒排表 (词频-倒排表模型) 二元词索引 (问题: 词汇总迅速增长): 将文档中每个连续词对看成一个词组，构建面向二元词的倒排表，并处理两个词 (或多个词) 构成的短语查询 更长的短语查询: 可以分成多个短语查询来处理 (如果采用二元词索引拼接的方式，对于长词查询返回的结果，不能确定其中是否真正包含最原始的短语词语) 2) 位置信息索引: 在记录词频的同时，记录它们已在文档中出现的位置，可以达成更广泛的范围。对于短语查询，仍采用合并算法 (AND)，查找符合的文档，当然不只是简单判断两个词是否出现在同一文档中，还需要检查它们出现的位置信息是否符合要求，并且该方法能够用于邻近搜索 (例如，间隔 k 个词)

索引存储: 词典与倒排表分开存储 (可以支持并行、分布式查询) 词汇总表存储: 顺序、哈希、B/B+树、Trie 树 (前缀树)

**索引压缩**

(节省磁盘空间 (M)，提高效率 (内存利用或数据传输速度) 前提: 快速的解压缩算法，目前的变长统计算法效率比较高。) 意义: 1) 对词典: 压缩得足够小，可以直接放入内存，提升效率 2) 对倒排记录表: 减少所需的磁盘盘空间，可以更多移入内存，并且可以减少从磁盘读取倒排表所需的时间

词典库存储方式: 定长存储 (词典本身设置定长为 20 字节，另外需要记录词典频率和指向倒排表各 4 字节)，但是空间会浪费 (英文中平均的词典词长度为 8 个字节)

压缩倒排表: 将词典库中每一字符串 (在词典字符串中，每个词项平均长度 8 个字节 + 其次，词项文档频率与倒排表指向各 4 字节不变 + 最后，词项指向符 3 字节)

词典库存储: 单一字符在词典上需要占用较多额外空间，通过为每 k 个词项存储一个指针，来减少指针的总数量，需要额外 1 个字节用于表示词项长度。例如，当块的大小 k=4 时，不采用按块存储时，每个指针花费 3 字节，共需 12 字节 + 采用按块存储时，只需要花费 3+4\*1=7 字节

在搜索上的问题: 二分查找只能在块外进行，块内采用线性查找方式 (随着 k 上升，线性查找的增多，效率更低) 改进: 前端编码 (Front Coding)

采用前缀代替具体 ID——再进一步: 可变长度编码 (第一个存数据，其它存前缀)

先写前缀，并分配 bit 作为延续位

如果 G<128，则采用前一位延续位为 1 + 7 位有效二进制编码的格式

如果 G=128，则为对低 8 位 (然后，然后再采取前缀算法) 对高位位进行编码，最后一个字节 (8bit) 的延续位为 1，其他字节延续位为 0。

214577 的二进制为 1101,0001,1001,10001，VB 模型为 00001110,00011100,11001011

**查询和评价**

查询: 用户输入关键词或短语

评价: FN: Positive or Negative, 表示查询与样本的关系 (和结果是否一致)

TF: 词频，表示统计判断的正确性和 (和结果是否一致)

四种评价的含义:

- TF: True Positive, 真正例数，正确识别正例，真真正正
- FP: False Positive, 假正例数，错误识别正例为真，假阳性
- FN: False Negative, 假负例数，错误识别负例为真，假阴性
- TN: True Negative, 真正例数，正确识别负例，真真正正

	TP	FP
实际为真	10	10
实际为假	10	10

针对某一检索问题，各个算法分别给出检索结果中的 Top N 个文档，将这些结果汇总起来并进行人工标注，从而得到一个相关的文档池 (虽然它实际上仍然无法得到全部相关文档，因此并不能得到召回率的结果，但是，它可以比较各个算法的相对优。N 通常取 50-200)

**F 值**:

$F(\text{measure})$ ，即精确率与召回率的调和平均数

$F = \frac{1}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P+R}$

通常情况下，我们取  $\alpha=0.5$  (即两者同等重要)

此时，可得最大的 F 值，即  $F=2P \times R$

面向单查询的基本评价指标: 1) **准确率 (Precision)**: 指检索出的文档中，相关文档所占比例。也即查准率，计算公式为  $TP/(TP+FP)$  2) **召回率** (Recall): 指所有相关文档中，被检索出的部分的比例，也称查全率。计算公式为  $TP/(TP+FN)$

召回率的近似计算: 缓冲池 (Pooling)

针对某一检索问题，各个算法分别给出检索结果中的 Top N 个文档，将这些结果汇总起来并进行人工标注，从而得到一个相关的文档池 (虽然它实际上仍然无法得到全部相关文档，因此并不能得到召回率的结果，但是，它可以比较各个算法的相对优。N 通常取 50-200)

**F 值**:

$F(\text{measure})$ ，即精确率与召回率的调和平均数

$F = \frac{1}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P+R}$

通常情况下，我们取  $\alpha=0.5$  (即两者同等重要)

此时，可得最大的 F 值，即  $F=2P \times R$

如何基于 N 维特征向量 ROC 曲线做出好坏 + 如果线条 A 离原点更近，则线条 A 对应的算法效果更好 + 如果两条线发生重合，则可认为两条算法效果相当

- 计算 AUC，AUC 越高效果越好
  - Area under curve, 曲线下方面积
  - 与面积成正比近似计算
- 另外，当使用非 A 面积时，可使用平衡点计算
  - 平衡点即 Precision = Recall 点，值越高越好

P-R 曲线 ROC 曲线的选择

- P-R 曲线与 ROC 曲线的选择
  - ROC 曲线能真正反映评价，更为全面，而 P-R 曲线只考虑正确例
  - “关注某些关注点正例，如果数据中特定正例 (被检索)，P-R 曲线是个好选择
  - 单份数据正例样本比例失调时，P-R 曲线更合适
  - 当数据中大量正例时，负例数据仅占多数数据时，P-R 曲线不合适，使用 ROC 曲线更合适
- P-R 曲线受分布影响大，多数数据且正比例不一时 ROC 曲线最合适
- P-R 曲线两个指标各自针对正样本，而 Precision 只针对样本，而 ROC 曲线更全面

**P@N**, 即 Precision@N, 指前 N 个检索结果文档的准确率: 如果相关文档数小于 N, P@N 的理论上限必小于 1

**R@N**, 即 Recall@N, 指前 N 个检索结果找回的相关文档比例: 1) 由于返回结果有限，Recall@N 值，甚至其理论上限往往都远小于 1 2) 理论上限为 N/相关文档数，即使通过 Pooling 加以控制仍然较小

**R-Precision** 与 P@N 相关文档信息

未读读 A: 查询共有 6 个相关文档，检索返回了 3 篇相关信息，位置分别第 1，第 2，第 5，第 10，第 20 位，则  $AP = \frac{1}{6} (1 + \frac{2}{2} + \frac{3}{5} + \frac{4}{10} + \frac{5}{20} + \frac{6}{20}) = 0.5$

读读 A: 事先知道文档按相关性排序，则只需要计算 1 个位置即可，计算在返回前 6 个 (第 1~6) 中，10%，20%，10% 对应 1 个位置上文档相关性高，即 R-Precision: 只计算前 6 个位置相关性行为， $AP = \frac{1}{6} (1 + \frac{2}{2} + \frac{3}{5} + \frac{4}{10} + \frac{5}{20} + \frac{6}{20})$ ，偏向那些快速返回结果的文档，没有考虑召回率等其他的情况 (不推荐)

累计增益 CG: 位置 1 到 p 的检索结果相关度之和

$DCG_p = rel_1 + \sum_{i=2}^p \frac{rel_i}{log_2(i)}$  (ppt 用第一个) 或  $DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{log_2(i+1)}$

假设有 10 个文档，相关度为 1-3 之间，10 个文档的得分依次如下: + 3, 2, 3, 1, 0, 1, 2, 3, 1, 0

理想的输出结果顺序为: 1, 3, 3, 2, 2, 1, 1, 0, 0, 0

由计算得 DCG 值为: 1, 3.6, 7.0, 8.0, 8.75, 10.02, 10.08, 10.08, 10.08

而与此同时，基本的 DCG 结果如下: + 3.5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61 (使用非线性特性)

由计算得 NDCG 结果为: 1, 0.83, 0.87, 0.76, 0.71, 0.69, 0.73, 0.8, 0.88, 0.88

可以看到在查出的结果中的 NDCG 值都略高于 DCG 值

多查询的排序: 1) **MAP**: 对每个文档的 (高雷) AP 求简单平均 (可以反映全部查询的综合效果，但在查询难度不平衡的条件下有误导) 2) **MRR**: 多个查询所得的倒数 (第一相关文档排序求评价 (eg. 两个查询, 第一个的第一个相关文档在位置 2, 第二个的第一个相关文档在位置 4, 则  $MRR = (1/2 + 1/4) \div 2 = 3/8$ ，即平均在 8/3 的位置上找到一个相关文档) 3) **为差**: 一个检索系统对不同查询的查准率越大，则 MAP 和 MRR 对相应查询的方差，即评价查准率误差越大

**多样性评价**: 基本形式: 给定一个查询 q，返回一个多样化的结果文档集合 R(q) (R(q)作为一个整体，应满足以下条件: + R(q)中所有结果文档都与查询 q 本身具有较强的相关性。 + 总体上要有较多的冗余，以覆盖 q 的不同方面。) 核心思想: 降低用户无法获得所需信息的可能性 (即不能按排序靠前的文档中只有一个结果满足用户需要的需求) 总需求: 除不同文档之间的主题

题差异性。两种衡量方式: 1) 隐式模型: 只计算文档之间的差异性，文档是什么内容，不会也无法进行详细查询 2) 显式模型: 更加具体地考虑文档针对的用户意图 + 会从文档中抽地主题，并显式地给用户提供多样化的 + 隐式模型: **MMR** + 显式模型: **FM-LDA**

最大边界相关性 (Maximal Marginal Relevance, MMR) + Facet model with LDA (FM-LDA)

最早的多样性评价模型，由 Caron 和 Zeldenz 在 2009 年提出，考虑文档的不同子主题 (Facet)

$MMR(p) = \argmax_{p \in P} \{ \alpha \log \frac{1}{|P|} + (1 - \alpha) \log \frac{1}{|P \cap A|} \}$

该公式表示文档集合 P 中，其中: + 前半部分表示文档与主题 A 的相关性 + 后半部分表示文档与主题 A 的多样性

其中， $p_i \in A$  表示主题 A 中 (隐含主题) 的文档， $Y_i$  = 表示主题 A 中被选中 + 的文档，如果部分表示主题 A 至少被一个文档选中 + 的文档

3. 用文档 A 与主题 A 的关联，或主题 A 与主题 A 的关联，或主题 A 与主题 A 的关联

当有约束  $\sum_{i=1}^n x_i \leq t$  时，返回的文档数量满足约束 / 篇

**相关性反馈**: 用户在查询后标记相关/不相关，然后迭代更新查询，以获得更好的结果 过程: 1. 首先，用户提出一个查询条件 (Query) 2. 对于返回的文档，用户标记相关与不相关部分 3. 系统根据用户反馈，获得用户信息需求更为准确的描述 (a) 基于相关性反馈，更新查询条件 b) 基于新查询条件，获取新的搜索结果并再次提交给用户进行评价) > 上述过程将根据情况进行一次或多次的迭代，从而不断逼近最佳查询结果

目的: 通过相关性反馈，获得用于表达用户查询意图的最佳查询条件 (即优化方式: 为已查词项 添加不相关权重，或增加 新词项，这一过程应对用户敏感) 相关性反馈类型: 1) **显式反馈**: 用户显式地参与交互过程 (如通过用户评论)，但是只有正样本，并且用户不点击，不代表完全不相关，所以有了拓展的显式反馈: 收集负面评价的反馈信息 (如差评，还有用户标记) 2) **隐式反馈 (更常见)**: 系统跟踪用户行为来推断返回文档的相关性 (判定不一定准确，但会告诉用户的行为方式与行为，如鼠标单击查询，用户鼠标移开等) 优点: + 不需要用户显式参与，减轻用户负担，提升用户体验 + 用户行为为某种操作可以反映出其兴趣，因此具有可行性 缺点: + 对行为分析有着较高的要求，准确难度以保证 + 某些情况下需要添加额外条件 (如差评) 3) **负反馈**: 在没有用户参与的前提下，直接移除负面信息的要求，并进而进行 (准确性差，可能查错漏) 4) **查询扩展**: 用户针对资料的合适程度给出反馈，这些反馈将被用来构建更为完整的查询条件。 + 合适功能: 用户选择和确认的查询扩展能够更好表达其查询意图。 实现: 利用同义词典，可以实现查询条件的扩展。一般而言，查询扩展用于提升查询的召回率 (找得多)，但是，可能会影响准确率，尤其对扩展留存歧义的情况下，编辑和修正同义词典需要很大的代价。

类型: 1. 利用人工编辑的同义词典 2. 全部分词与同义词典的自动生成 3. 基于搜索引擎进行优化

自动构建相关词项的两种思想: 2. 分析两个文档集中的相同部分，来自动生成同义词/近义同义词典: 1. 如果两个词项经常和相似的词共同出现，则它们俩可能都是 (相近) 2. 如果两个词项经常与同一词共同出现，那么它们俩可能也是相似的

基于搜索日志的查询扩展: 搜索日志目前都是搜索引擎查询/扩展的重要方式 (落在扩展范围) 1. 看了一个网页又点另一个 (后断的) 2. 查了两个词 (互为) 后都点同一个链接

相关词扩展的潜在在问题: 词项关联的质量是一个问题 1. 有歧义的查询词可能导致统计上相关，而意思上不相干的词 2. 同时，由于扩展的查询词与原查询词高度相关，扩展后的查询词也未必能够 获得更多的相关文档)

**排序**

排序的缺点: Web 网页的信息组织与内容质量参差不齐，用户难以缺乏知识和经验，用户意图多种多样差异巨大

好的排序: 网页内容质量与内容 + 网页内容质量

信息检索模型: 文档与查询内容 + 相关性框架，实质是对文档基于相关性进行排序，在理解用户的基础之上产生近似用户决策的结果从而在顶部页面展示相关的信息。[D,Q,R][D,q] 布尔模型 D 文档表-词项的组合; Q 查询表-布尔表达式; F 完全 (二值)匹配; R 满足布尔表达式相关性为 1 (否则 0)

**相关系数**: Jaccard 系数:  $JACCARD(A, B) = \frac{|A \cap B|}{|A \cup B|}$

对数词频  $w_{fd, a} = \begin{cases} 1 + \log_{10} t_{fd, a} & t_{fd, a} > 0 \\ 0 & otherwise \end{cases}$

词项频率 Tf(t,d): 词项 t 在文档 d 中出现的次数

**排序**

排序的缺点: Web 网页的信息组织与内容质量参差不齐，用户难以缺乏知识和经验，用户意图多种多样差异巨大

好的排序: 网页内容质量与内容 + 网页内容质量

信息检索模型: 文档与查询内容 + 相关性框架，实质是对文档基于相关性进行排序，在理解用户的基础之上产生近似用户决策的结果从而在顶部页面展示相关的信息。[D,Q,R][D,q] 布尔模型 D 文档表-词项的组合; Q 查询表-布尔表达式; F 完全 (二值)匹配; R 满足布尔表达式相关性为 1 (否则 0)

**相关系数**: Jaccard 系数:  $JACCARD(A, B) = \frac{|A \cap B|}{|A \cup B|}$

对数词频  $w_{fd, a} = \begin{cases} 1 + \log_{10} t_{fd, a} & t_{fd, a} > 0 \\ 0 & otherwise \end{cases}$

词项频率 Tf(t,d): 词项 t 在文档 d 中出现的次数

文档频率 Df(t): 出现词项 t 的文档数数 逆文档频率 IDF:  $idf_t = \log_{10} \frac{N}{df_t}$

$W_{t,d} = (1 + \log t_{f,t,d}) \cdot \log \frac{N}{df_t}$  此为 TF-IDF

**向量空间模型 VSM**: 文档和查询词作为一个词项向量构成的向量 + 计算向量之间的相似性 + 按相似度大小排序返回 Top-K 文档给用户。 (D: 文档表，每个文档可视为一个向量，其中每一维对应词项的 tf-idf 值; Q: 查询表，可视为一个向量，其中每一维对应词项的 tf-idf 值; F: 非完全匹配系数; R: 使用两个向量之间的相似性度量衡量文档与查询文档之间的相关性) 优点: 简洁直观，可支持多种不度量量和权重方式，实用效果不错; 缺点: 缺乏语义层面的理解和匹配，同时依赖 tf-idf 值也可能造成干扰，用户无法描述词项之间的关系，词项的独立性假设实际上不成立。

文档匹配与迭代优化: 欧氏距离 (不行，太远了，对于向量长度 (文档长度) 非常敏感)  $dist(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$

余弦相似度 (文档向量与查询向量的夹角大小来计算):  $cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^n q_i d_i}{\sqrt{\sum_{i=1}^n q_i^2} \sqrt{\sum_{i=1}^n d_i^2}}$   $q_i d_i = \alpha_i \beta_i + \beta \frac{1}{|D_{i,1}|} \sum_{j=1}^n d_{ij} - \gamma \frac{1}{|D_{i,1}|} \sum_{j=1}^n d_{ij}$

**Rocchio (罗切罗) 算法** (使用查询尽可能离与相关的文档更近，离与不相关的文档更远)

其中，Dr 为已知相关文档的向量集合，Dnr 为已知不相关文档的向量集合 + q0 为初始查询向量， $\alpha, \beta, \gamma$  为权重，根据手工调节经验设定。由此，新的查询向量将逐渐向相关文档向量移动，远离不相关文档向量。 相关反馈信息融入到向量空间模型

基于迭代的查询向量更新: + 正反馈 vs 负反馈 + 正反馈的权重往往大于负反馈，用户更关心符合需求的标准答案，而不是错误答案。相应的，可以通过设置  $\beta > \gamma$  来给予正反馈更大的权重 + 很多系统甚至只允许正反馈，即  $\gamma = 0$  + 收集真正的负反馈往往比较困难

**Word2vec**: 1. 根据上下文预测中心词 (CBOW) 2. 根据中心词预测上下文 (Skip-gram)

+ 根据上下文预测中心词，小模型其模型 (Skip-gram 模型基本模型类似) + CBOW 模型上是在给定的上下文中预测中心词的问题:  $y = softmax(Wx)$  + 如何训练这样的模型: 基于训练好的 CBOW 模型训练 CBOW 模型 + 使用 Word2vec (CBOW) 模型表示词频:  $x_i$  + 计算交叉熵损失，反向传播得到权重:  $L = -\sum_{i=1}^n \log p_i$  + 基于多个不同词组进行交叉熵交叉熵 + 最终得到每个词组的向量:  $W, x_i$

两个模型比较: 1) 从性能上说 + CBOW 模型只预测中心词，复杂度约为 O(V)，即词表规模 + 而 Skip-gram 模型基于中心词预测周边词，复杂度约为 O(VW)，即考虑窗口 2) 从效果上说 + Skip-gram 模型中，由于每个词都可以作为中心，都将得到针对性更强的词，因此，对于生词 (数据稀疏) 的词语而言，Skip-gram 模型效果更好

Word2vec 优缺点: 1) 优点: 有效表征了词项之间的上下文关系 + 无监督，通用性强，可适用于各种 NLP 任务 2) 缺点: 无法解决一次多义的问题，例如 play music and play football + Word2vec 是一种静态的方式，其词项表征一旦训练确定就不会再改变。因此，虽然通用性强，但是无法针对特定任务进行动态优化

**排序学习算法**: Pointwise 算法 Pairwise 算法 Listwise 算法

**Pointwise**: 基本假设: 训练样本中的任何一个查询-文档对，都可以映射到一个分值或一个有序的类别 (如类似中差) + 相应的，给定一个查询-文档对，Pointwise LTR 将试图预测其得分/类别。 + 常见的类别类别包括: 1) 回归，将查询-文档对映射到具体得分 2) 分类，将排序问题转化为一个向有序类别的二分/多分问题 3) 有序回归，在映射到具体得分的同时保持得分之间的有序关系 + 可以简单且广泛地套用于已有回归、分类算法 局限性: 1) 首先，Pointwise 类方法往往更注重文档的相关度得分，而并不注重文档间的相对性排序 (Pairwise 类方法的出现，为解决这一问题提供了新的手段) 2) 其次，不同查询所对应的文档，尤其相关程度不同，对反馈造成的损失也会各不相同，一定程度上影响效果

**Pairwise**: 基本假设: 同样是将排序问题转化为分类问题 (二分类或三类) + 每次比较一个查询与两个文档，衡量两个文档的排序 (Partial Order) + 分类器的目的在判断哪个文档应该排在前面 (对应的标签为 1, -1) 或 (0, 1, -0) (0 表示两个文档之间可以并列)

相比于 Pointwise 算法，Pairwise 类排序算法通过衡量样本大小之间的顺序关系，实现了从绝对相关性 (分值) 到相对排序的进阶缺陷: 1) 两两对比导致样本数大为提升，计算资源开支增加 2) Pairwise 类算法仍然受样本不平衡问题的影响 3) 最后，Pairwise 算法无法体现整体排序的合理性。综上，由此引出了最后一类算法: Listwise 类算法 Pairwise 局限性: 样本不平衡性的影响以及算法无法体现整体排序

**Listwise**: 基本思路: 直接面对整体排序结果进行优化 (直接将排序的完整序列作为学习的对象) 解决这一问题采用以下两种思路: 1) 直接采用某种 IR 指标对排序进行优化 2) 直接针对面向完整排序的损失函数 代表性思路: 采用某种 IR 指标对排序进行优化 通常情况下，由于全局优化的作用，Listwise 类排序算法可以采取相比于 Pairwise 类算法更优的效果。 + 的挑战: 例如两个网页并列的情况 (相当于 Pairwise 中文档标签为 0 的情况) 拓展: Listwise 与 Pairwise 的融合: FocusedRank

**PageRank**: 优质网页引用或推荐的网页一定也优质

$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$  PR(p) 为网页 p 的 PageRank 值，初值为 1/N; PR(p) 为指向网页 p 的某个网页 p\_j 的 PageRank 值; L(p) 为网页 p\_j 出链的链接数量; d 为阻尼系数，取值在 0-1 之间; N 为网页总数，M(p) 为网页 p 的页面集合。

邻接矩阵 > 转移矩阵; 转置 (行列互易) 而各数值除以出链数量

$A = \begin{bmatrix} 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix} \Rightarrow R = \begin{bmatrix} 0.5 & 1 & 10.5 & 0.5 & 10.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 10.5 & 0.5 & 0.5 & 10.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 10.5 \\ 10.5 & 1 & 0.5 & 1 & 0.5 \end{bmatrix}$

$M = \begin{bmatrix} 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 \end{bmatrix}$