Something Technical

[2003 CIKM] Efficient Query Evaluation using a Two-Level Retrieval Process

本文支持怎样的Query?



top-k keyword search,用的ranking function是additive score method,也即是query 和document的相关性是每一个query term和document相关性的累加,同时,本文还可以用OR semantic的方式来支持AND semantic,当然比用signature等index来判断一个document是不是包含所有的keyword,效率还是低,只是通用性好一些。

DAAT相比TAAT的优势是什么?



- DAAT需要的内存比较小,它不像TAAT那样要维护很多中间结果
- DAAT的并发度要更好,比如我可以对document id做partition,然后各个 partition之间是相互独立的

算法的思路是怎样的?

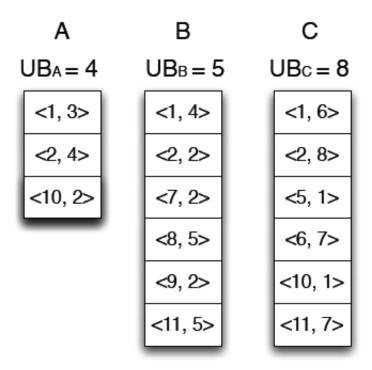


算法的框架还是服从DAAT,所有的inverted lists都是按照document id排序的,然后每一个query term对应的list会维护一个max score,用来算upper bound。DAAT的 pruning是通过检查哪些范围里头的document id不可能找到一个更好的结果从而不需要扫描整个list,换句话说,它在扫inverted list的时候有点像skip list不停地往前跳(相比之下,TAAT是顺序扫的,没有跳跃)。这种跳跃就是本文的 contribution,它可以有效地减小访问的document的数目,而实现这种跳跃的核心要素是对document的访问严格按照id的顺序来的。

举例说明WAND的原理



假设我们有这么3个inverted list按照doc id排序,对每一个inverted list,我们维护它的max relevance score,比如A里头doc 2的relevance最高,是4.



对每一个inverted list,我们维护一个指针,假设A扫到<10,2>, B扫到<7,2>, C扫到<5,1>,这时候doc 1和2的relevance score都已经可以完整地算出来了,并且放在heap里面。假设我们要找的是top-2,那么剩下的doc的relevance至少应该是13,才能进入top2。这时候我们看各个指针指向的doc的upper bound,我们的目的是找到pivot doc (doc id最小并且upper bound > 13)。

这里有个很重要的地方需要注意,就是给定一个doc,在算upper bound的时候,不是用doc的relevance加上剩下list的upper bound,而是 $\sum_{1 \leq t \leq p} UB_t$,举个例子,我们从doc 5开始,doc 5的relevance是1,虽然doc 5不出现在其它的inverted list,但它的upper bound是8,而不是1。显然doc 5不是pivot doc,下一个candidate 是list B的doc 7,它的upper bound是list C的upper bound加上list B的upper bound,所以是13,但还是没有比13大,也不是candidiate,最后是list A的doc 10,它的upper bound是8+5+4=17,是一个candidate.

也许你会奇怪,为什么不用更小一些的upper bound呢?因为我们通常理解的upper bound是说自己的doc relevance加上潜在的term的upper bound。那样的话结果可能会是错的,比如list B里面的doc 8的score从5换成15,这样doc 8肯定是个candidate,如果不是用几个list的upper bound相加,doc 7依然不是pivot doc,这样doc 8就被跳过去了,造成结果是错的。

跟max_score算法相比,本文做了哪些提高?



max_score的算法毕竟是95年提出的,还比较粗糙,需要顺序访问document,没有跳跃的部分。本文的WAND核心就是尽可能减小next()的数目,也就是尽可能fully evaluate更少的document

参考



■ [1995 Information Processing and Management] Query Evaluation Strategies and Optimizations (max_score的实现算法)