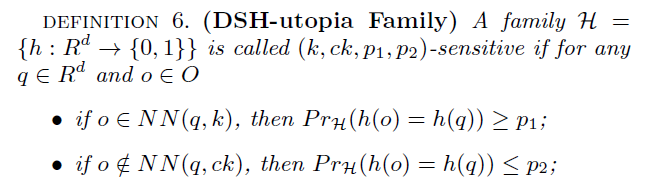
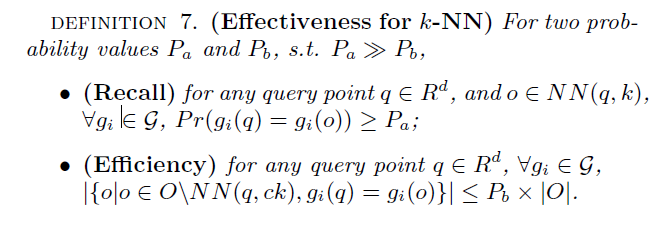
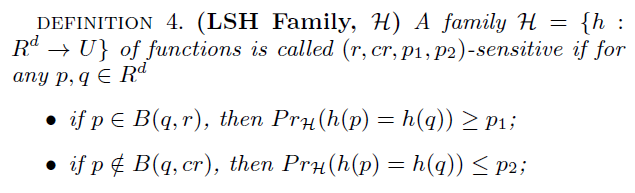
DSH:

1. **通过LSH衍生出这种专门针对KNN的DSH**



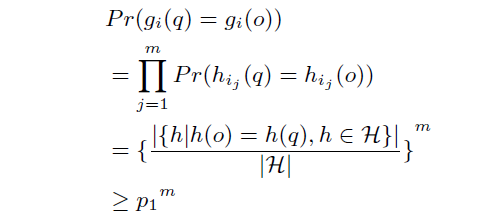
1. **通过串联扩大概率**：



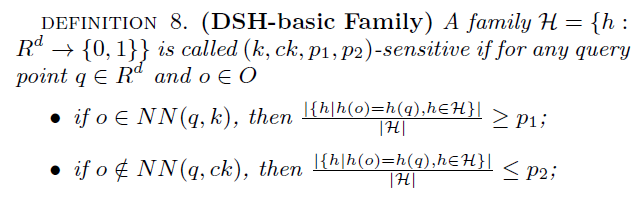
可能我们理解不同就是这里，这里我理解的gi(q) = gi(o)中的“=”，就是完全等于，其中i就是hash table的个数，我为什么这么说呢，因为也是这篇论文里，他对LSH的描述就是右边这样的：

1. **通过上一步，就推出了DSH-basic Family**

如果将第二步中gi(q) = gi(o)的实现就是通过整个hash函数族能产生碰撞的概率来表示，



**那么第二步中的公式就演变成这样：**



1. **这样所有的理论成立的基础就变成了只要以下公式成立就行了**



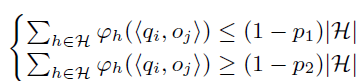
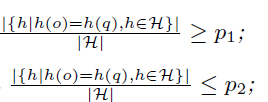


为了定量表述，所以论文定义了一个公式，



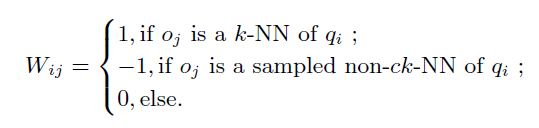
也就是说*h(o) = h(q)*相等，ϕ就为0，不相等就为1。

所以

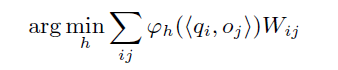


1. **如何保证上面的公式成立呢，如果要保证所有的数据都满足上面公式，计算量太大，那么论文就采用采样，只需要保证样本满足上面公式就行，**

就有了权重矩阵W，最原始的W就是对KNN和CKNN的表示，



**回到原来的问题上，如何保证样本都满足上面的公式呢，为了能够定量的计算，论文就提出下面这个公式（和类内距离和有点类似）**

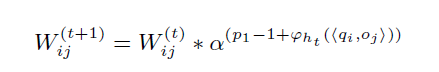
**，其中**

****

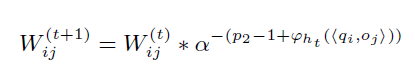
然后通过自使用增强算法，不断调整W，着重调整那些hash出错的地方。（这个自适应增强我看的也是一知半解）。

通过迭代计算，每次都求出一个hash函数，然后对hash函数进行检测，如果hash符合KNN则减小Wij权重，不符合则增大。下一次迭代的时候在着重调整这些高权重的地方，也就是对Hash函数进行不断改进。这个权重更新的的过程是保证概率p1,p2的呢，

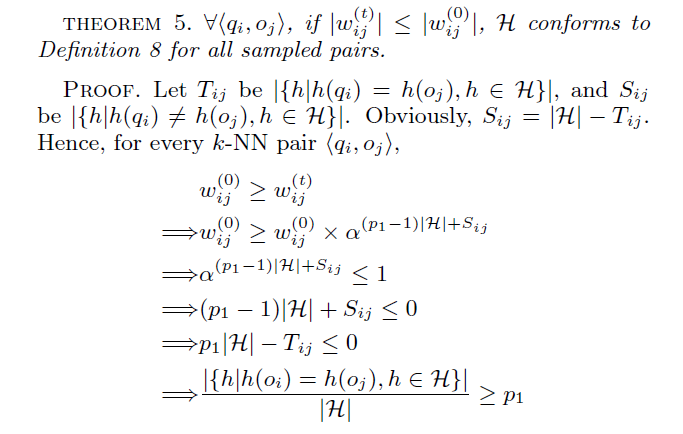
对于KNN中的点（也就是看hash是否正确， true positive）



对于non-KNN中的点（也就是false positive）



证明过程是这样的：



**标红圈的地方我感觉是**

****

1. **最后可以得到一个hash函数族，就可以生成hash table，然后就可以进行KNN搜索**

**根据上面的推导，我认为是hash签名完全相同就落入一个桶。**