# COMP9313: Big Data Management

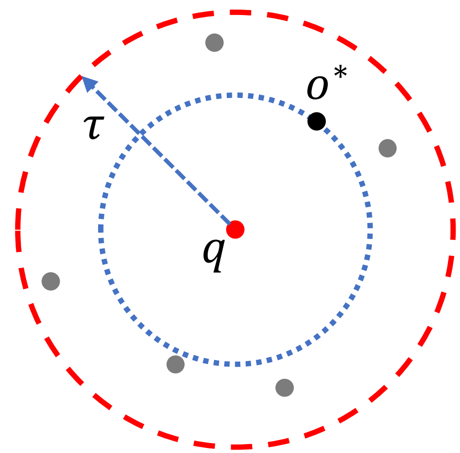
## High Dimensional Similarity Search

高维相似度搜索

相似搜索

# Similarity Search

- Problem Definition:
  - Given a query $q$ and dataset $D$, find o $\in D$, where $o$ is similar to $q$
- Two types of similarity search
  - Range search: 距离搜索
    - $dist(o, q) \leq \tau$




最近邻居搜索

  - Nearest neighbor search
    - $dist(o^*, q) \leq dist(o, q), \forall o \in D$
    - Top-k version

距离和相似度函数变化

- Distance/similarity function varies

欧几里德，德卡勤、内部生产

  - Euclidean, Jaccard, inner product, …
- Classic problem, with mutual solutions

相互解决

高维相似性搜索

# High Dimensional Similarity Search

- Applications and relationship to Big Data
  - Almost every object can be and has been 表示 represented by a high dimensional vector
    - Words, documents 高维向量
    - Image, audio, video
    - … 相似性搜索是信息检索的基本过程
  - Similarity search is a fundamental process in information retrieval 面部识别系统
    - E.g., Google search engine, face recognition system, …

高维

- High Dimension makes a huge difference!
  - Traditional solutions are no longer feasible 可行的
  - This lecture is about why and how
    - We focus on high dimensional vectors in Euclidean space

欧式空间中的高维向量

# **Similarity Search** in **Low Dimensional Space**

低维空间

# Similarity Search in One Dimensional Space

- Just numbers, use binary search, binary search tree, B+ Tree…
- The essential idea behind: objects can be sorted 对象可以排序

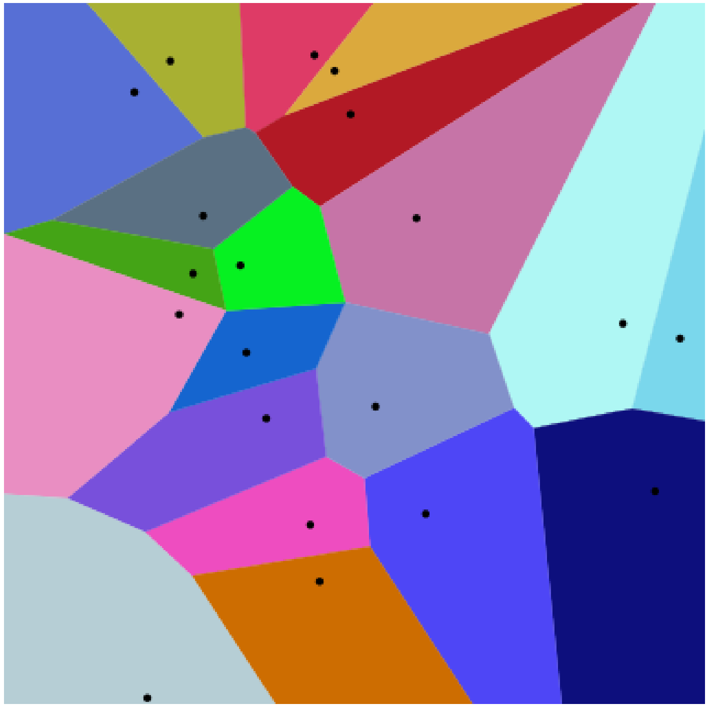
# Similarity Search in Two Dimensional Space

二进制搜索

- Why binary search no longer works?
  - No order!

维诺图

- Voronoi diagram



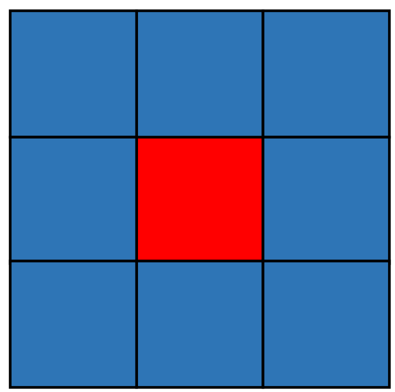
Euclidean distance

Manhattan distance

# Similarity Search in Two Dimensional Space

分割 算法

- Partition based algorithms
  - Partition data into “cells”
  - Nearest neighbors are in the same cell with query or adjacent cells



- How many “cells” to probe on 3-dimensional space?

度量空间

# Similarity Search in Metric Space

- Triangle inequality 三角不等式
  - $dist(x, q) \leq dist(x, y) + dist(y, q)$
- Orchard's Algorithm
  - for each $x \in D$, create a list of points in increasing order of distance to $x$
  - given query $q$, randomly pick a point $x$ as the initial candidate (i.e., pivot $p$), compute $dist(p, q)$
  - walk along the list of $p$, and compute the distances to $q$. If found $y$ closer to $q$ than $p$, then use $y$ as the new pivot (e.g., $p \leftarrow y$).
  - repeat the procedure, and stop when
    - $dist(p, y) > 2 \cdot dist(p, q)$

度量空间

# Similarity Search in Metric Space

- Orchard's Algorithm, stop when $dist(p, y) > 2 \cdot dist(p, q)$

$$
\begin{aligned}
& 2 \cdot dist(p, q) < dist(p, y) \text{ and} \\
& dist(p, y) \le dist(p, q) + dist(y, q) \\
\Rightarrow\ & 2 \cdot dist(p, q) < dist(p, q) + dist(y, q) \\
\Leftrightarrow\ & dist(p, q) < dist(y, q)
\end{aligned}
$$

- Since the list of $p$ is in increasing order of distance to $p$, $dist(p, y) > 2 \cdot dist(p, q)$ hold for all the rest $y$'s.

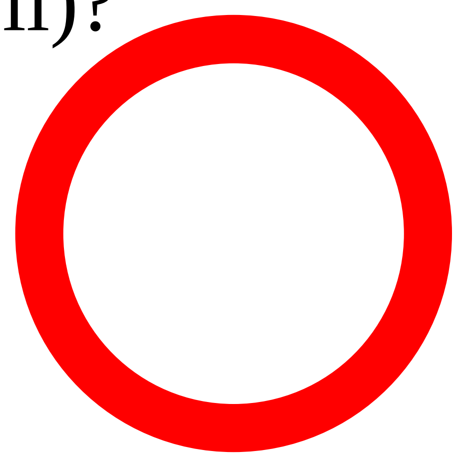
# None of the Above Works in High Dimensional Space!

维数灾难

# Curse of Dimensionality

各种现象

- Refers to various phenomena that arise in high dimensional spaces that do not occur in low dimensional settings.
- Triangle inequality 三角不等式
  - The pruning power reduces heavily 修剪能力大大降低
- What is the volume of a high dimensional "ring" (i.e., hyperspherical shell)? 高维的体积是多少 超球壳
  - $\frac{V_{ring}(w=1,d=2)}{V_{ball}(r=10,d=2)} = 29\%$
  - $\frac{V_{ring}(w=1,d=100)}{V_{ball}(r=10,d=100)} = 99.997\%$



高维空间中的近似近邻搜索

# Approximate Nearest Neighbor Search in High Dimensional Space

没有子线性解决方案可以找到最近邻居查询的确切结果

- There is no sub-linear solution to find the exact result of a nearest neighbor query

  条件

  - So we relax the condition

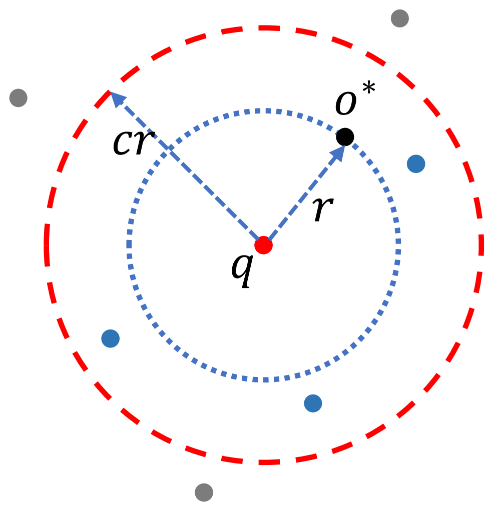
  近似最近邻居搜索

- approximate nearest neighbor search (ANNS)
  - allow returned points to be not the NN of query
  - Success: returns the true NN
  - use success rate (e.g., percentage of succeed queries) to evaluate the method
  - Hard to bound the success rate

近似

# c-approximate NN Search

- Success: returns $o$ such that
  - $dist(o, q) \leq c \cdot dist(o^*, q)$
- Then we can bound the success probability
  - Usually noted as $1 - \delta$
- Solution: Locality Sensitive Hashing (LSH) 位置敏感的哈希



位置敏感的哈希

# Locality Sensitive Hashing

- Hash function
  - Index: Map data/objects to values (e.g., hash key)
    - Same data ⇒ same hash key (with 100% probability)
    - Different data ⇒ different hash keys (with high probability)

检索：轻松检索相同的对象

  - Retrieval: Easy to retrieve identical objects (as they have the same hash key)
    - Applications: hash map, hash join
  - Low cost
    - Space: $O(n)$
    - Time: $O(1)$
  - Why it cannot be used in nearest neighbor search?
    - Even a minor difference leads to totally different hash keys

局部敏感哈希

# Locality Sensitive Hashing

使散列函数容错

- Index: make the hash functions error tolerant
  - Similar data ⇒ same hash key (with high probability)
  - Dissimilar data ⇒ different hash keys (with high probability)
- Retrieval: 检索
  - Compute the hash key for the query
  - Obtain all the data has the same key with query (i.e., candidates)
  - Find the nearest one to the query
  - Cost:
    - Space: $O(n)$
    - Time: $O(1) + O(|cand|)$ 局部敏感哈希
- It is not the real Locality Sensitive Hashing!
  - We still have several unsolved issues…

# LSH Functions

- Formal definition:
  - Given point $o_1, o_2$, distance $r_1, r_2$, probability $p_1, p_2$
  - An LSH function $h(\cdot)$ should satisfy
    - $\Pr[h(o_1) = h(o_2)] \geq p_1$, if $dist(o_1, o_2) \leq r_1$
    - $\Pr[h(o_1) = h(o_2)] \leq p_2$, if $dist(o_1, o_2) > r_2$
- What is $h(\cdot)$ for a given distance/similarity function?
  - Jaccard similarity
  - Angular distance
  - Euclidean distance

雅卡德相似度?
角距离

?欧几里德距离

雅卡德相似度?

# MinHash - LSH Function for Jaccard Similarity

- Each data object is a set
  - $Jaccard(S_1, S_2) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$
- Randomly generate a global order for all the elements in $C = \bigcup_1^n S_i$
- Let $h(S)$ be the minimal member of $S$ with respect to the global order

反向字母顺序排列

  - For example, $S = \{b, c, e, h, i\}$, we use inversed alphabet order, then re-ordered $S = \{i, h, e, c, b\}$, hence $h(S) = i$.

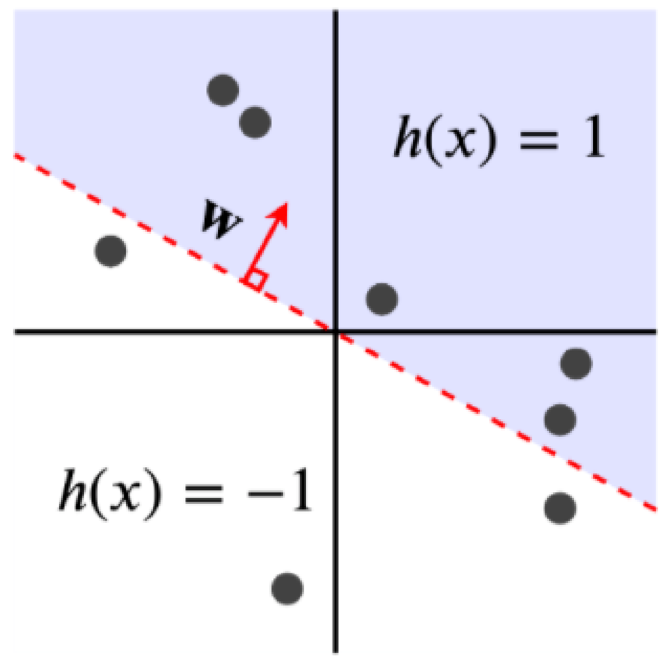
# MinHash

- Now we compute $\Pr[h(S_1) = h(S_2)]$
- Every element $e \in S_1 \cup S_2$ has equal chance to be the first element among $S_1 \cup S_2$ after re-ordering
- $e \in S_1 \cap S_2$ if and only if $h(S_1) = h(S_2)$
- $e \notin S_1 \cap S_2$ if and only if $h(S_1) \neq h(S_2)$
- $\Pr[h(S_1) = h(S_2)] = \frac{|\{e_i | h_i(S_1) = h_i(S_2)\}|}{|\{e_i\}|} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} = Jaccard(S_1, S_2)$

局部敏感哈希 角距离

# SimHash – LSH Function for Angular Distance

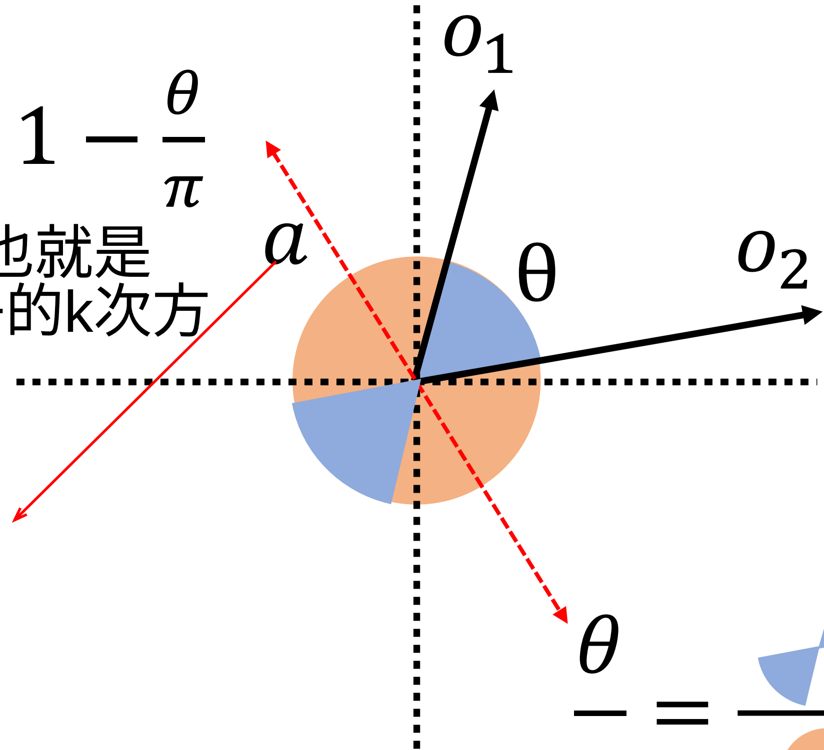
- Each data object is a d dimensional vector
  - $\theta(x, y)$ is the angle between $x$ and $y$
- Randomly generate a normal vector $a$, where $a_i \sim N(0,1)$
- Let $h(x; a) = \text{sgn}(a^T x)$
  - $\text{sgn(o)} = \begin{cases} 1; if\ o \geq 0 \\ -1; if\ o < 0 \end{cases}$
  - $x$ lies on which side of $a$

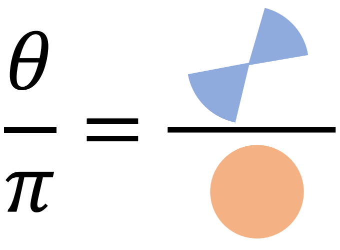


# SimHash

- Now we compute $\Pr[h(o_1) = h(o_2)]$
- $h(o_1) \neq h(o_2)$ iff $o_1$ and $o_2$ are on different sides of $a$
- $\Pr[h(o_1) = h(o_2)] = 1 - \frac{\theta}{\pi}$

一位hashcode相等的概率是多大，也就是黄色的区域，如果是k位，上面式子的k次方
a是随机向量



局部敏感哈希 欧氏距离；欧几里德距离

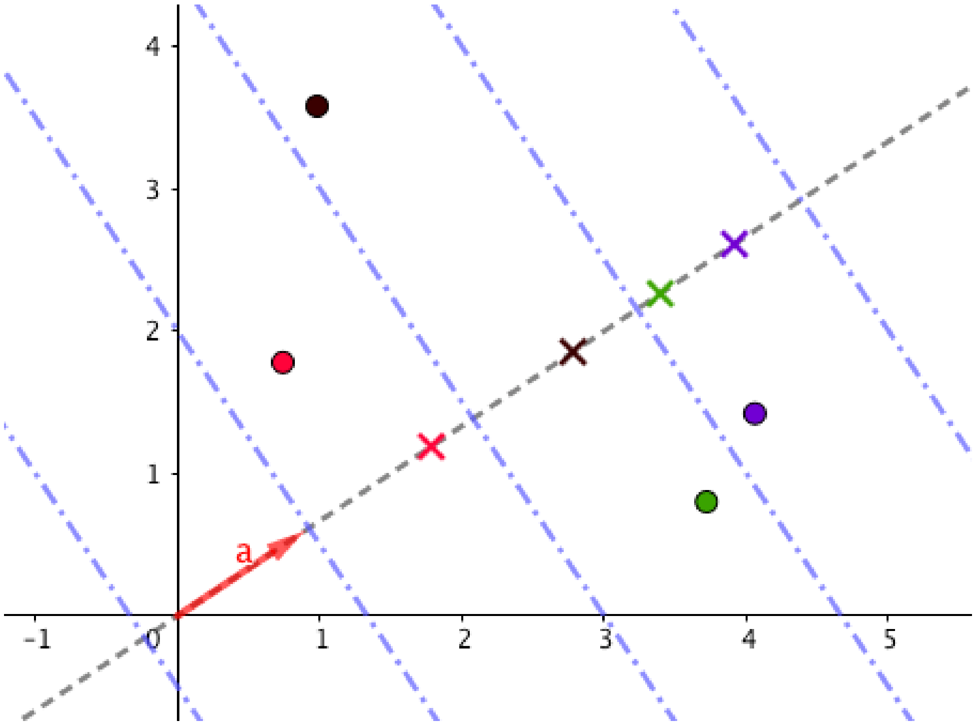
# p-stable LSH - LSH function for Euclidean distance

- Each data object is a d dimensional vector
  - $dist(x, y) = \sqrt{\sum_1^d (x_i - y_i)^2}$
- Randomly generate a normal vector $a$, where $a_i \sim N(0,1)$
  - Normal distribution is 2-stable, i.e., if $a_i \sim N(0,1)$, then $\sum_1^d a_i \cdot x_i \sim N(0, \|x\|_2^2)$
- Let $h(x; a, b) = \left\lfloor \frac{a^T x + b}{w} \right\rfloor$, where $b \sim U(0,1)$ and 规定的参数；额定参数 $w$ is user specified parameter
  - $\Pr[h(o_1; a, b) = h(o_2; a, b)] = \int_0^w \frac{1}{\|o_1, o_2\|} f_p\left(\frac{t}{\|o_1, o_2\|}\right)\left(1 - \frac{t}{w}\right) dt$
  - $f_p(\cdot)$ is the pdf of the absolute value of normal variable

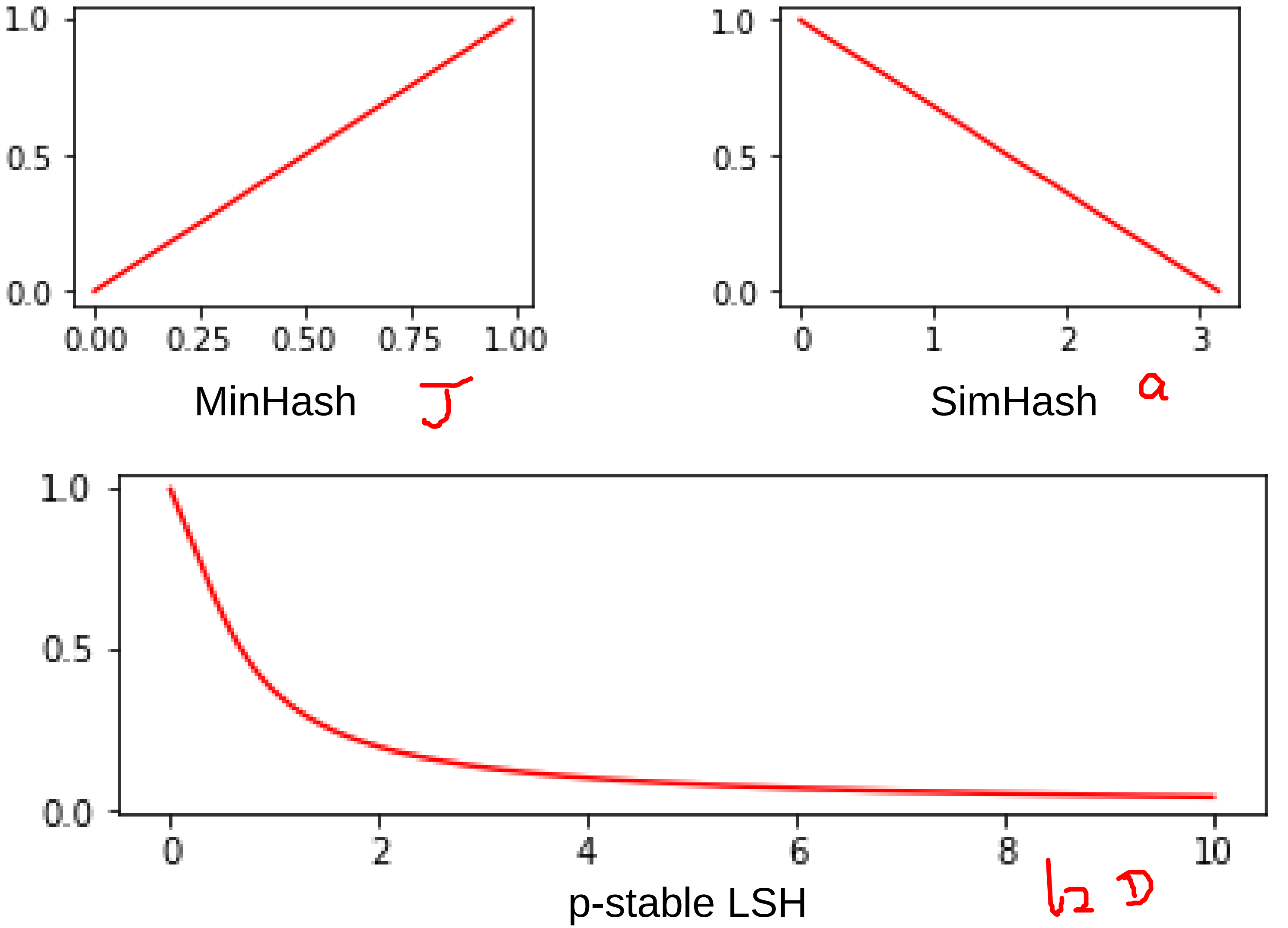
# p-stable LSH

直觉力

- Intuition of p-stable LSH
  - Similar points have higher chance to be hashed together

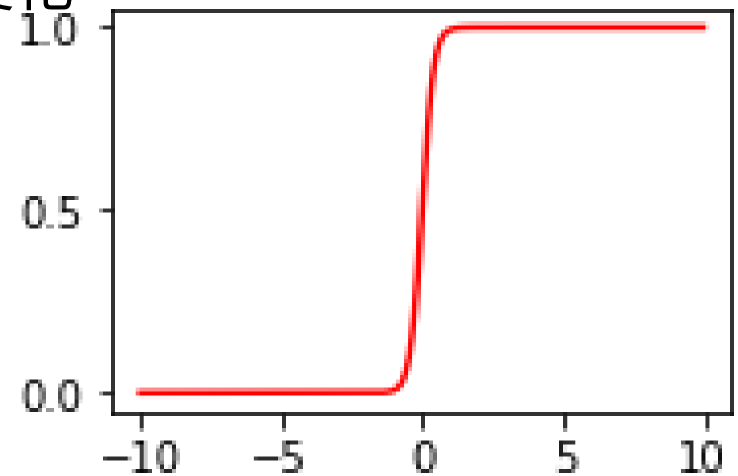


# $\Pr[h(x) = h(y)]$ for different Hash Functions



# Problem of Single Hash Function

- Hard to distinguish if two pairs have distances close to each other
  - $\Pr[h(o_1) = h(o_2)] \geq p_1$, if $dist(o_1, o_2) \leq r_1$
  - $\Pr[h(o_1) = h(o_2)] \leq p_2$, if $dist(o_1, o_2) > r_2$
- We also want to control where the drastic change happens… 剧烈的变化
  - Close to $dist(o^*, q)$
  - Given range



组成

# AND-OR Composition

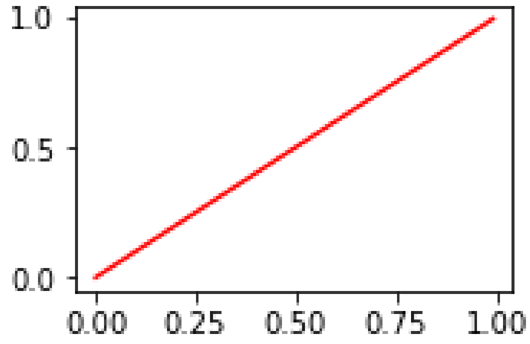
- Recall for a single hash function, we have
  - $\Pr[h(o_1) = h(o_2)] = p(dist(o_1, o_2))$, denoted as $p_{o_1,o_2}$

场景，情景

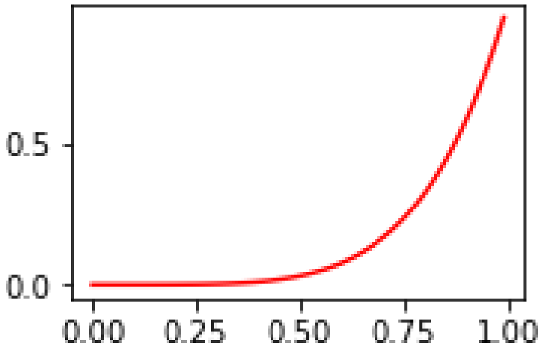
- Now we consider two scenarios:
  - Combine $k$ hashes together, using AND operation
    - One must match all the hashes
    - $\Pr[H_{AND}(o_1) = H_{AND}(o_2)] = {p_{o_1,o_2}}^k$
  - Combine $l$ hashes together, using OR operation
    - One need to match at least one of the hashes
    - $\Pr[H_{OR}(o_1) = H_{OR}(o_2)] = 1 - (1 - p_{o_1,o_2})^l$
      - Not match only when all the hashes don't match

# AND-OR Composition

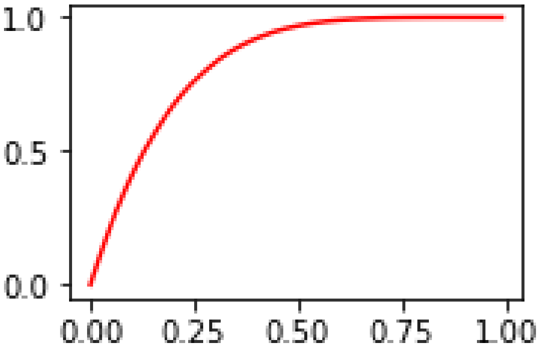
- Example with minHash, $k = 5, l = 5$




$\Pr[h(o_1) = h(o_2)]$




$\Pr[H_{AND}(o_1) = H_{AND}(o_2)]$




$\Pr[H_{OR}(o_1) = H_{OR}(o_2)]$

组成

# AND-OR Composition in LSH

- Let $h_{i,j}$ be LSH functions, where $i \in \{1,2,\dots,l\}, j \in \{1,2,\dots,k\}$
- Let $H_i(o) = [h_{i,1}(o), h_{i,2}(o), \dots, h_{i,k}(o)]$
  - super-hash
  - $H_i(o_1) = H_i(o_2) \Leftrightarrow \forall j \in \{1,2,\dots,k\}, h_{i,j}(o_1) = h_{i,j}(o_2)$
- Consider query $q$ and any data point $o$, $o$ is a nearest neighbor candidate of $q$ if
  - $\exists i \in \{1,2,\dots,l\},\ H_i(o) = H_i(q)$
- The probability of $o$ is a nearest neighbor candidate of $q$ is
  - $1 - (1 - p_{q,o}^k)^l$

# The Effectiveness of LSH

- $1-(1-p_{q,o}^{k})^{l}$ changes with $p_{q,o}$, $(k=20, l=5)$

| $p_{q,o}$ | $1-(1-p_{q,o}^{k})^{l}$ |
|---|---|
| 0.2 | 0.002 |
| 0.4 | 0.050 |
| 0.6 | 0.333 |
| 0.7 | 0.601 |
| 0.8 | 0.863 |
| 0.9 | 0.988 |

- E.g., we are expected to retrieve 98.8% of the data with Jaccard > 0.9

# False Positives and False Negatives

- False Positive:
  - returned data with $\text{dist}(\text{o}, \text{q}) > r_2$
- False Negative
  - not returned data with $\text{dist}(\text{o}, \text{q}) < r_1$
- They can be controlled by carefully chosen $k$ and $l$
  - It’s a trade-off between space/time and accuracy

局部敏感哈希

# The Framework of NNS using LSH

- Pre-processing
  - Generate LSH functions
    - minHash: random permutations 随机排列
    - simHash: random normal vectors
    - p-stable: random normal vectors and random uniform values
- Index
  - Compute $H_i(o)$ for each data object $o$, $i \in \{1, \dots, l\}$
  - Index $o$ using $H_i(o)$ as key in the $i$-th hash table
- Query
  - Compute $H_i(q)$ for query $q$, $i \in \{1, \dots, l\}$
  - Generate candidate set $\{o | \exists i \in \{1, \dots, l\}, H_i(q) = H_i(o)\}$
  - Compute the actual distance for all the candidates and return the nearest one to the query

计算所有候选者的实际距离，并向查询返回最接近的候选者

# The Drawback of LSH

- Concatenate k hashes is too “strong”
  - $h_{i,j}(o_1) \neq h_{i,j}(o_2) \Rightarrow H_i(q) \neq H_i(o)$ for any $j$
- Not adaptive to the distribution of the distances
  - What if not enough candidates?
  - Need to tune w (or build indexes different w’s) to handle different cases

# Multi-Probe LSH

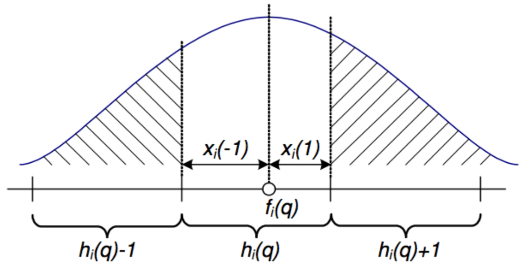



**Figure 3: Probability of $q$'s nearest neighbors falling into the neighboring slots.**

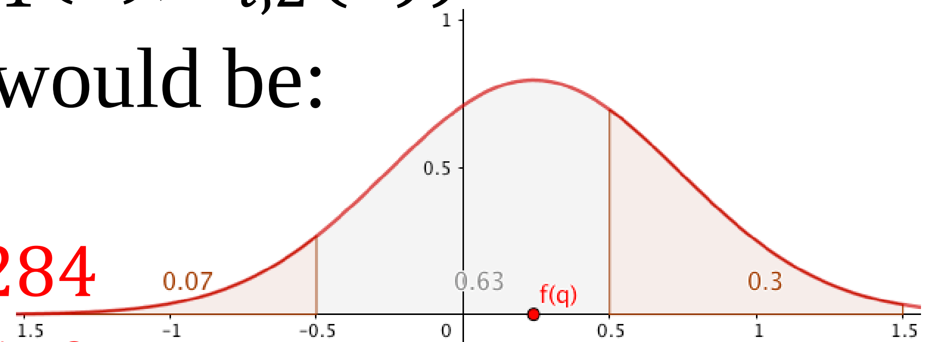
- Observation:
  - If $q$'s nearest neighbor does not falls into $q$'s hash bucket, then most likely it will fall into the adjacent bucket to $q$'s
  - Why? $\sum_1^d a_i \cdot x_i - \sum_1^d a_i \cdot q_i \sim N(0, \|x, q\|_2^2)$
- Idea:
  - Not only look at the hash bucket where $q$ falls into, but also those adjacent to it
- Problem:
  - How many such bucket? $2^k$
  - And they are not equally important!

他们并不重要

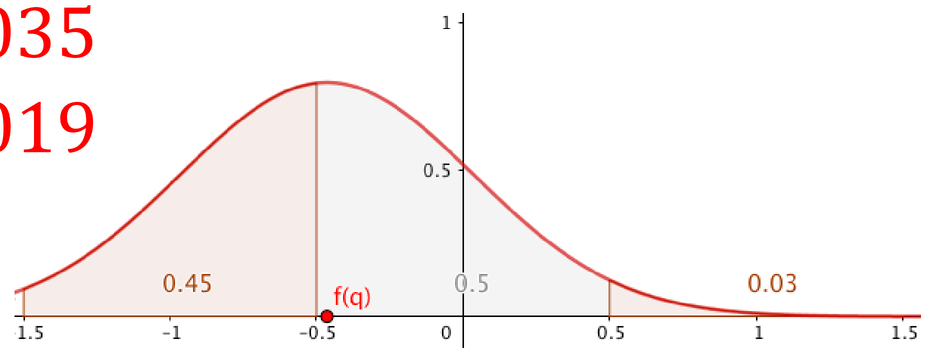
# Multi-Probe LSH

局部敏感哈希

- Consider the case when $k = 2$:
- Note that $H_i(o) = (h_{i,1}(o), h_{i,2}(o))$
- The ideal probe order would be:
  - $h_{i,1}(q), h_{i,2}(q)$: 0.315
  - $h_{i,1}(q), h_{i,2}(q) - 1$: 0.284
  - $h_{i,1}(q) + 1, h_{i,2}(q)$: 0.150
  - $h_{i,1}(q) - 1, h_{i,2}(q)$: 0.035
  - $h_{i,1}(q), h_{i,2}(q) + 1$: 0.019




$h_{i,1}$




$h_{i,2}$

We don't have to compute the integration, but use the offset between f($q$) and the boundaries.

多探针

# Multi Probe LSH

- Pros:
  - Requires less $l$
    - Because we use hash tables more efficiently
  - More robust against the unlucky points

更强的对抗不幸的点

- Cons:

失去有关结果的理论保证

  - Lose the theoretical guarantee about the results
  - Not parallel-friendly

碰撞计数局部敏感哈希

# Collision Counting LSH (C2LSH)

- C2LSH (SIGMOG'12 paper)
  - Which one is closer to $q$?

| $q$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $o_1$ | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 2 | 1 | 1 | 1 |
| $o_2$ | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |

我们将省略理论部分，从而导致与本文稍有不同的版本

- We will omit the theoretical parts hence leads to a slightly different version to the paper.
  - But the essential ideas are the same
- Project 1 is to implement C2LSH using PySpark!

计算碰撞

# Counting the Collisions

- Collision: match on a single hash function
  使用碰撞次数确定候选者
  - Use number of collisions to determine the candidates
  - Match one of the super hash with $q \rightarrow$ collides at least $\alpha m$ hash values with $q$
- Recall in LSH, The probability of $o$ with $\mathrm{dist}(o, q) \leq r_1$ is a nearest neighbor candidate of $q$ is $1 - (1 - p_1^k)^l$
- Now we compute the case with collision counting…

碰撞概率

# The Collision Probability

- $\forall o$ with $\text{dist}(o, q) \leq r_1$, we have
  - $\Pr[\#collision(o) \geq \alpha m] = \sum_{i=\alpha m}^{m} \binom{m}{i} p^i (1-p)^{m-i}$
  - $p = \Pr\left[h_j(o) = h_j(q)\right] \geq p_1$
- We define $m$ Bernoulli random variables $X_i \sim B(m, 1-p)$ with $1 \leq i \leq m$.
  - Let $X_i$ equal 1 if $o$ does not collide with $q$
    - i.e., $\Pr[X_i = 1] = 1 - p$
  - Let $X_i$ equal 0 if $o$ collides with $q$
    - i.e., $\Pr[X_i = 0] = p$
  - Hence $\mathrm{E}[X_i] = 1 - p$
    - Thus $E(\bar{X}) = 1 - p$ , where $\bar{X} = \frac{\sum_{i=1}^{m} X_i}{m}$.
- Let $t = p - \alpha > 0$, we have:
  - $\Pr[\bar{X} - E(\bar{X}) \geq t] = \Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} - (1-p) \geq t\right] = \Pr[\sum X_i \geq (1-\alpha)m]$

碰撞概率
# The Collision Probability

霍夫丁不等式
- From Hoeffding's Inequality, we have
  - $\Pr[\bar{X} - E(\bar{X}) \geq t] = \Pr[\sum X_i \geq (1-\alpha)m] \leq \exp\left(-\frac{2(p-\alpha)^2 m^2}{\sum_{i=1}^{m}(1-0)^2}\right) = \exp\left(-2(p-\alpha)^2 m\right) \leq \exp\left(-2(p_1-\alpha)^2 m\right)$
- Since the event "#collision($o$) $\geq \alpha m$" is equivalent to the event "$o$ misses the collision with $q$ less than $(1-\alpha)m$ times", 忽略了碰撞
  - $\Pr[\#collision(o) \geq \alpha m] = \Pr[\sum X_i < (1-\alpha)m] \geq 1 - \exp\left(-2(p_1-\alpha)^2 m\right)$
- Now you can compute the case for $o$ with $\mathrm{dist}(o,q) \geq r_2$ in a similar way…
- Then we can accordingly set $\alpha$ to control false positives and false negatives

虚拟哈希
# Virtual Rehashing

- When we are not getting enough candidates…
  - E.g., # of candidates < top-k
- Observation:
  - A close point $o$ usually falls into adjacent hash buckets of $q$'s if it does not collide with $q$
  - Why?
    - $\sum_1^d a_i \cdot x_i - \sum_1^d a_i \cdot q_i \sim N(0, \|x, q\|_2^2)$
- Idea: 将相邻的哈希桶包括在内
  - Include the adjacent hash buckets into consideration
    - So you don't need to re-hash them again…

虚拟哈希

# Virtual Rehashing

- At first consider $h(o) = h(q)$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $o_1$ | 1 | 0 | 2 | -1 | 1 | 2 | 4 | -3 | 0 | -1 |

- Consider $h(o) = h(q) \pm 1$ if not enough candidates

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $o_1$ | 1 | 0 | 2 | -1 | 1 | 2 | 4 | -3 | 0 | -1 |

- Then $h(o) = h(q) \pm 2$ and so on…

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $o_1$ | 1 | 0 | 2 | -1 | 1 | 2 | 4 | -3 | 0 | -1 |

# The Framework of NNS using C2LSH

- Pre-processing
  - Generate LSH functions
    - Random normal vectors and random uniform values
- Index
  - Compute and store $h_i(o)$ for each data object $o$, $i \in \{1, \ldots, m\}$
- Query
  - Compute $h_i(q)$ for query $q$, $i \in \{1, \ldots, m\}$
  - Take those $o$ that shares at least $\alpha m$ hashes with $q$ as candidates
  - Relax the collision condition (e.g., virtual rehashing) and repeat the above step, until we got enough candidates

候选生成的伪代码

# Pseudo code of Candidate Generation in C2LSH

```
candGen(data_hashes, query_hashes, αm, βn):
    offset ← 0
    cand ← ∅
    while true:
        for each (id, hashes) in data_hashes:
            if count(hashes, query_hashes, offset) ≥ αm:
                cand ← cand ∪ {id}
        if |cand| < βn:
            offset ← offset + 1
        else:
            break
    return cand
```

# Pseudo code of Candidate Generation in C2LSH

count(hashes_1, hashes_2, offset):

    counter ← 0

    for each $hash_1$, $hash_2$ in hashes_1, hashes_2:

        if $|hash_1 - hash_2| \leq$ offset:

            counter ← counter + 1

    return counter

# Project 1

- Spec has been released, deadline: 18 Jul, 2020
  - Late Penalty: 10% on day 1 and 30% on each subsequent day.
- Implement a light version of C2LSH (i.e., the one we introduced in the lecture)
- Start working ASAP

正确性和效率

- Evaluation: Correctness and Efficiency
- Must use PySpark, some python modules and PySpark functions are banned.
  - E.g., numpy, pandas, collect(), take(), …
  - Use transformations!

转换；变换

# Project 1

- There will be a bonus part (max 20 points) to encourage efficient implementations.
  - Details in the spec
- Make sure you have valid output
- Make your own test cases, a real dataset would be more desirable
  - Toy example in the spec is a real "toy" (e.g., for babies…)
- Won't accept excuses like "it works on my own computer"
- Don't violate the Student Conduct!!!