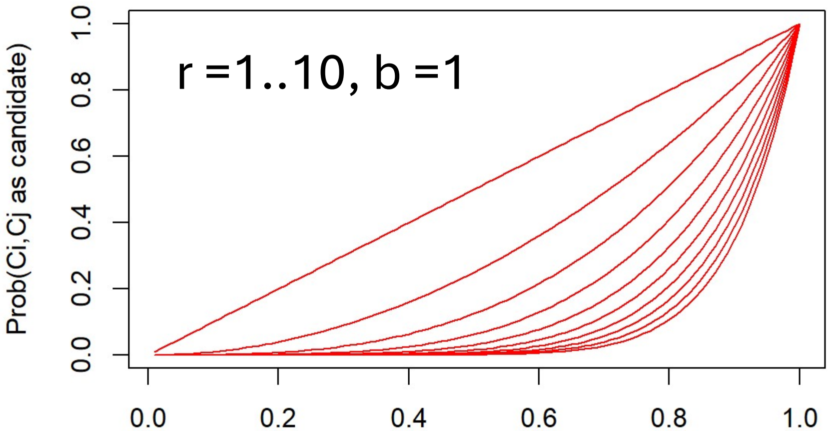
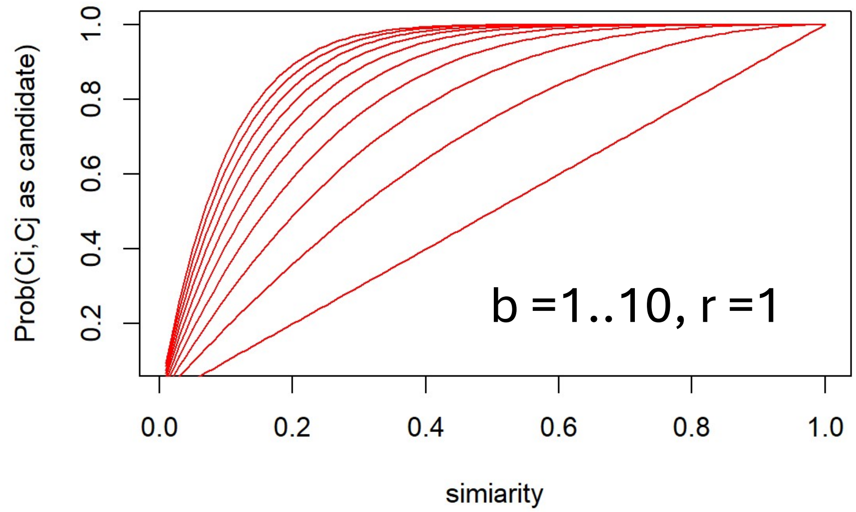
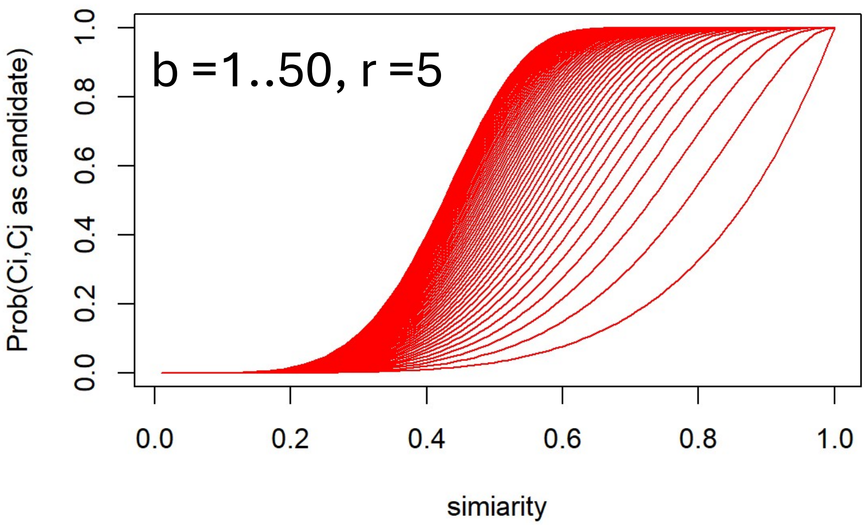
Next step: LSH Tuning

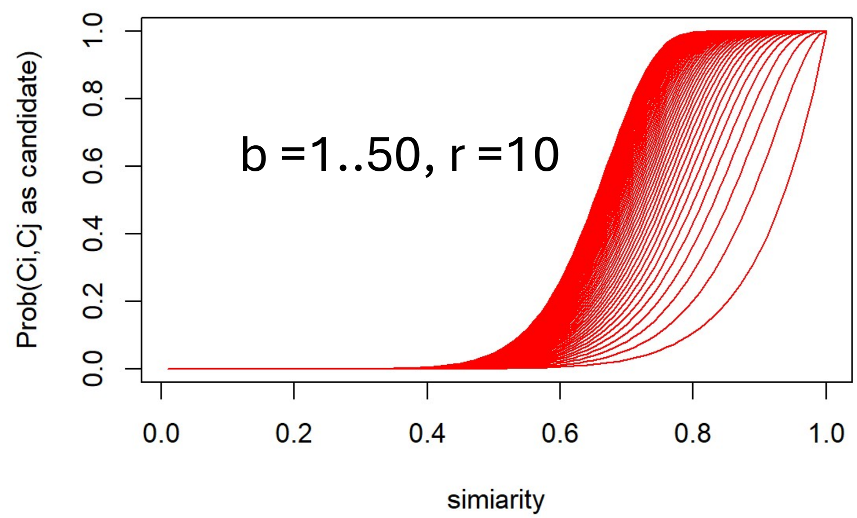
- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

The S function









LSH: families of functions

Locality-Sensitive functions

- The example we have previously seen is an example of **family of function** that can be combined to distinguish between pair at a low distance from pairs at a high distance
- Here we will describe other families of functions, beside the minhash one.
- There are 3 conditions that we need to ensure:
  - Close pair must be more likely candidates than distant ones
  - Statistically independent: estimate probability that two or more function will give a certain response
  - Efficient:
    - Time needed to identify candidate pair much less than computing distance between all pairs
    - Combinable to build functions that are better at avoiding false positive and negatives

Locality-Sensitive functions

- What is a Locality-Sensitive function?
  - **It is a function that takes in input two items and render a decision about whether these items should be a candidate pair.**
  - Shorthands:
    - $f(x) = f(y)$ x and y **are** candidate pair
    - $f(x) \neq f(y)$ x and y **are not** candidate pair
- A collection of such functions is called a **family of Locality-Sensitive functions**
- We say that a family **F** is $\boldsymbol{(d_1, d_2, p_1, p_2) - sensitive}$ if:
  - $\forall f \in F$:
    - If $d(x, y) \leq d_1$ the probability that $f(x) = f(y)$ is at least p_1
    - If $d(x, y) \geq d_2$ the probability that $f(x) = f(y)$ is at most p_2

LSH for min-hashing

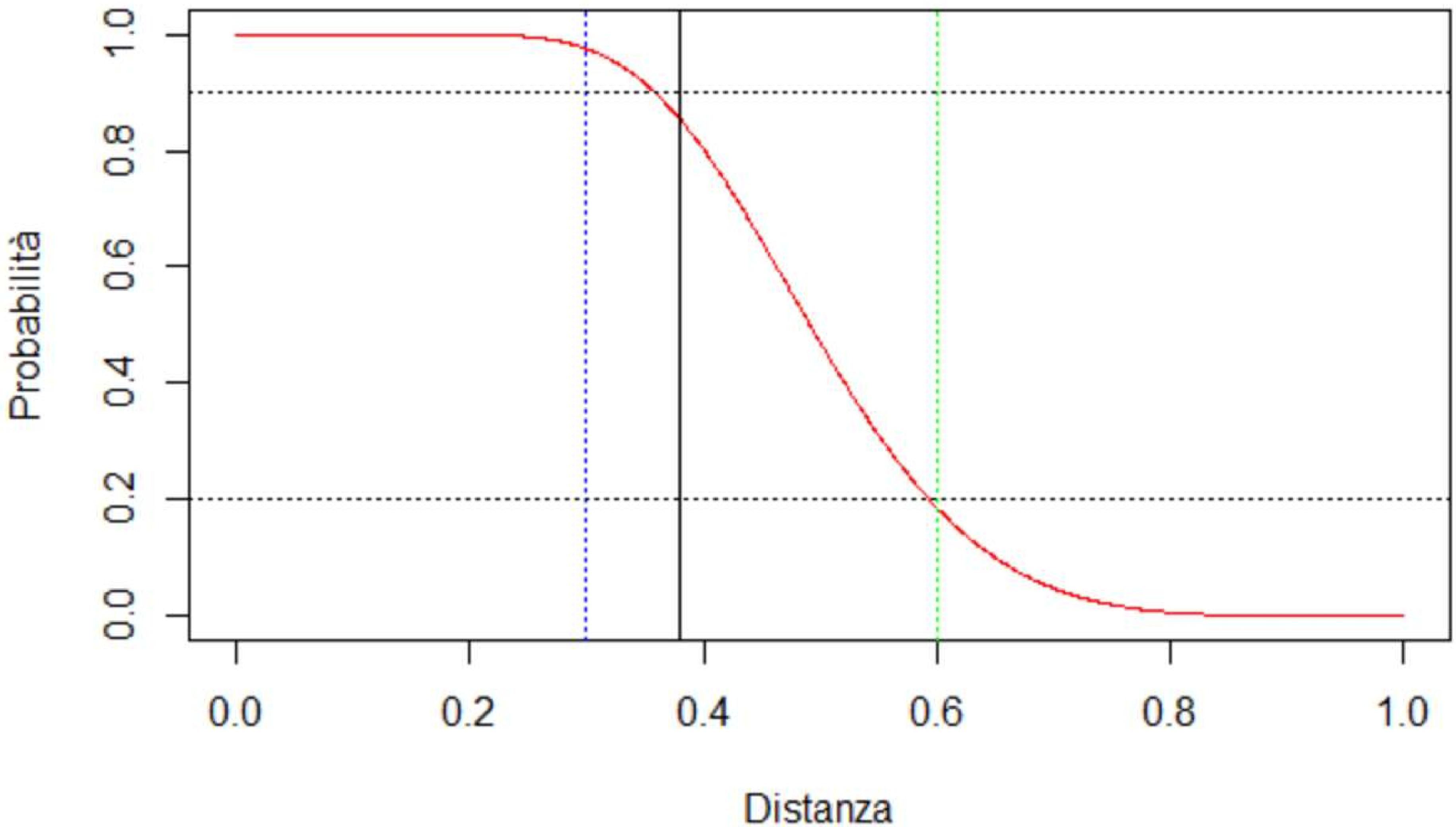
- We have
  - ***S*** = space of all sets
  - ***d*** = Jaccard distance,
  - ***H*** is family of Min-Hash functions for all permutations of rows
- For any hash function $\boldsymbol{h} \in \boldsymbol{H}$:
$$P[h(x) = h(y)] = 1 - d(x, y)$$
- Simply restates theorem about Min-Hashing in terms of distances rather than similarities

Locality-Sensitive functions

- The family of **minhash functions**, assuming that we are using a Jaccard Distance, is

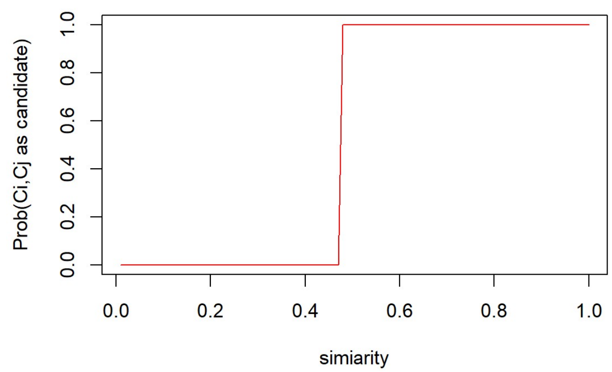
$$(\boldsymbol{d_1}, \boldsymbol{d_2}, \mathbf{1} - \boldsymbol{d_1}, \mathbf{1} - \boldsymbol{d_2}) - \boldsymbol{sensitive}$$

- For any $\boldsymbol{d_1}$ and $\boldsymbol{d_2}$ such that $0 < \boldsymbol{d_1} < \boldsymbol{d_2} \leq \mathbf{1}$
- For other distances, there is **no guarantee** that it might have a locality-sensitive family of hash functions.
- Min-hash H is a (1/3, 2/3, 2/3, 1/3)-sensitive family for S and d.



Probabilità
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Distanza

- Can we reproduce the "S-curve" effect we saw before for any LS family?



- The "bands" technique we learned for signature matrices carries over to this more general setting
- Can do LSH with any (d1, d2, p1, p2)-sensitive family!
- Two constructions:
  - AND construction like "rows in a band"
  - OR construction like "many bands"

Amplifying Locality-Sensitive functions

- Lets build a new family **F'** where each $\boldsymbol{f} \in \boldsymbol{F'}$ is built from **r** members of a family **F**, for some fixed **r**.
  - **f** is built from $\{\boldsymbol{f_1}, \ldots \boldsymbol{f_r}\}$ such that $\boldsymbol{f(x) = f(y)}$ if and only if $\boldsymbol{f_i(x) = f_i(y)}$ for all $\boldsymbol{i = 1, 2, \ldots, r}$
- If F is $(\boldsymbol{d_1, d_2, p_1, p_2}) - \boldsymbol{sensitive}$

F' will be $(\boldsymbol{d_1, d_2, (p_1)^r, (p_2)^r}) - \boldsymbol{sensitive}$

- **We are lowering all probabilities if we choose F and r judiciously**
- This process is called **AND-construction**

Hash function independecies

- Independence of hash functions (HFs) really means that the prob. of two HFs saying “yes” is the product of each saying “yes”But two particular hash functions could be highly correlated
- For example, in Min-Hash if their permutations agree in the first one million entries

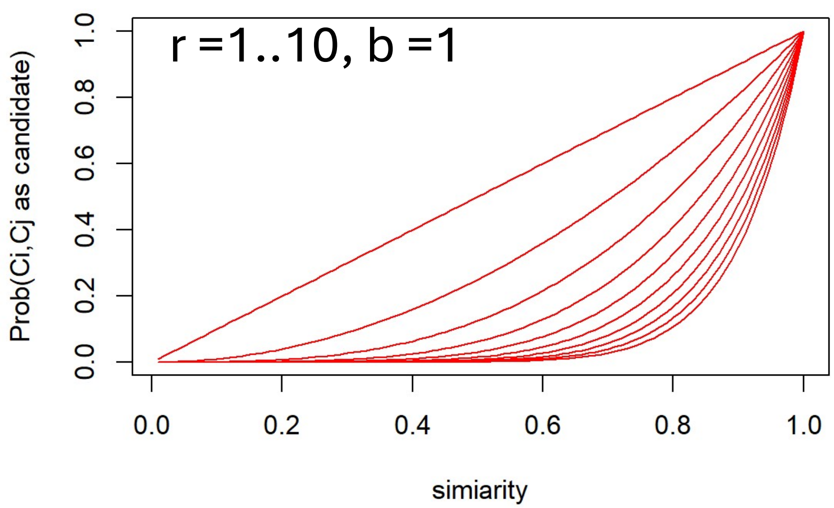
- However, the probabilities in definition of a LSH-family are over all possible members of H, H’ (i.e., average case and not the worst case)

Amplifying Locality-Sensitive functions

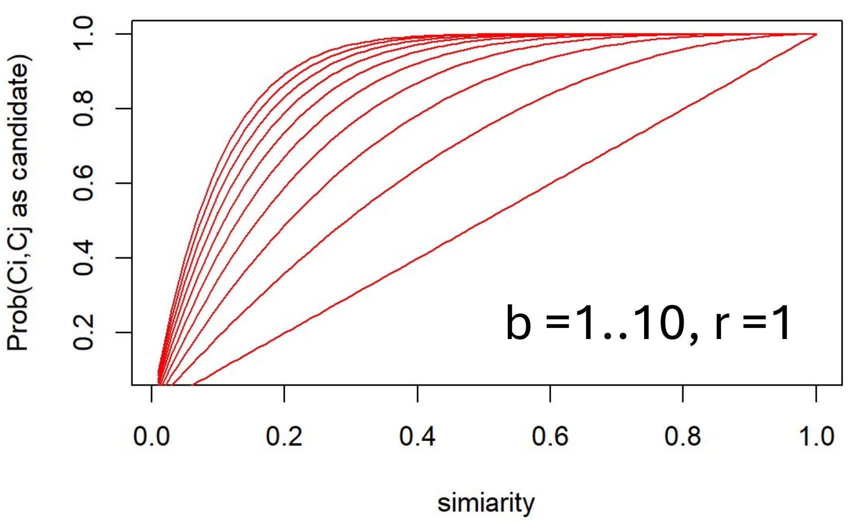
- Now lets build a family **F'** where each $\boldsymbol{f} \in \boldsymbol{F'}$ is built from **b** members of a family **F**, for some fixed b.
  - **f** is built from $\{\boldsymbol{f_1}, \ldots \boldsymbol{f_b}\}$ such that $\boldsymbol{f(x) = f(y)}$ if and only if $\boldsymbol{f_i(x) = f_i(y)}$ for one or more values $\boldsymbol{i}$
- If F is $(\boldsymbol{d_1, d_2, p_1, p_2}) - \boldsymbol{sensitive}$

  F' will be $\left(\boldsymbol{d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b}\right) - \boldsymbol{sensitive}$
- **We are increasing all probabilities if we choose F and b judiciously**
- This process is called **OR-construction**

- AND and OR constructions can be cascaded in order to make the low probability close to 0 and the high probability close to 1.



AND



OR

- By chosing b and r properly we can make the lower probability close to 0 and the higher close to 1.

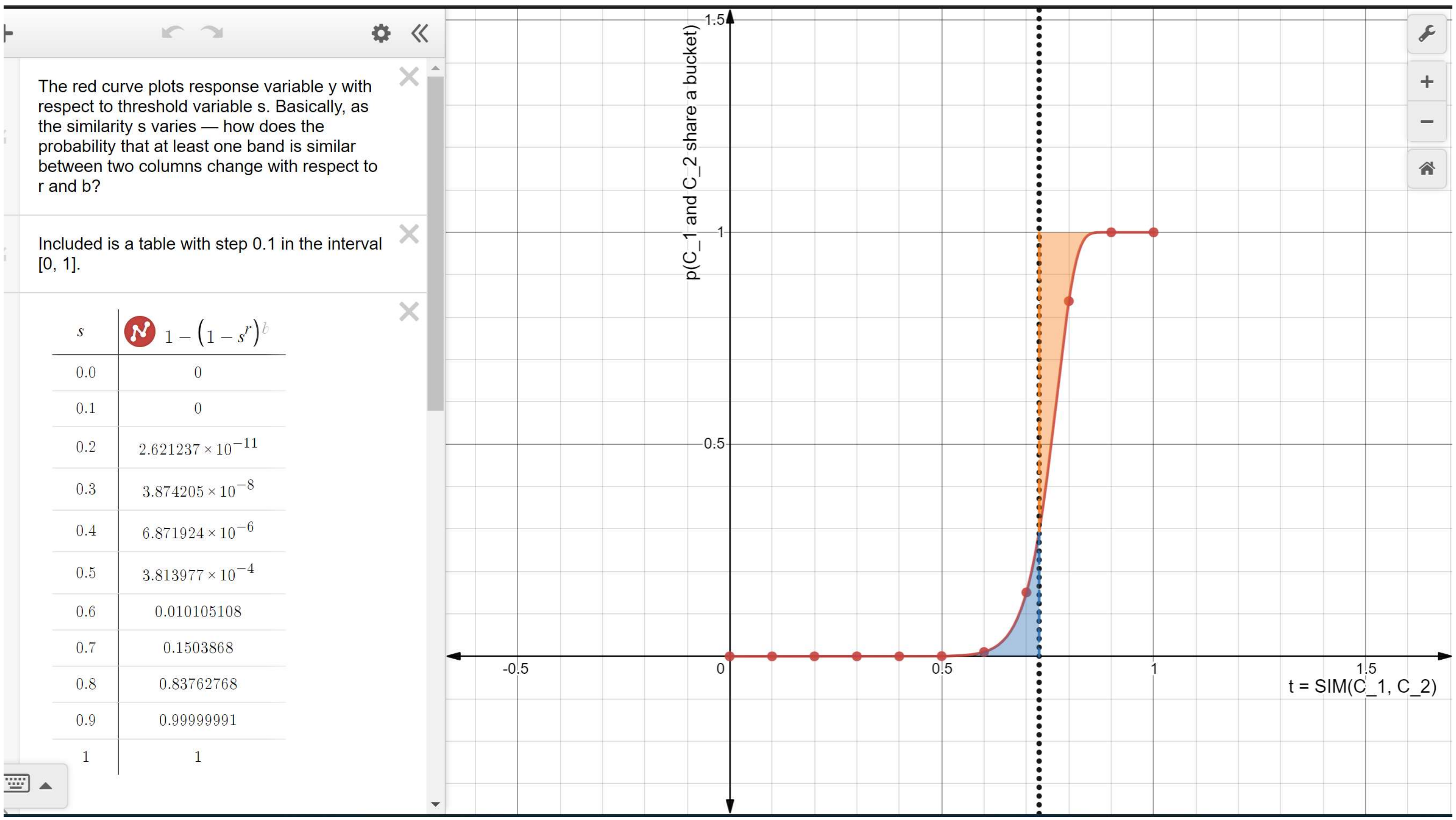
- As for the signature matrix, we can use the AND construction followed by the OR constructionOr vice-versa
- Or any sequence of AND’s and OR’s alternating

Example

- (0.2, 0.8, 0.8, 0.2)-sensitive
- B=4 r = 4.
  - **AND OR (0.2, 0.8, 0.878, 0.0064)-sensitive**
  - **OR AND (0.2, 0.8, 0.9936, 0.1215)-sensitive**
  - **Apply again OR and next AND**
    - **(0.2, 0.8, 0.9999996, 0.0008715)-sensitive**

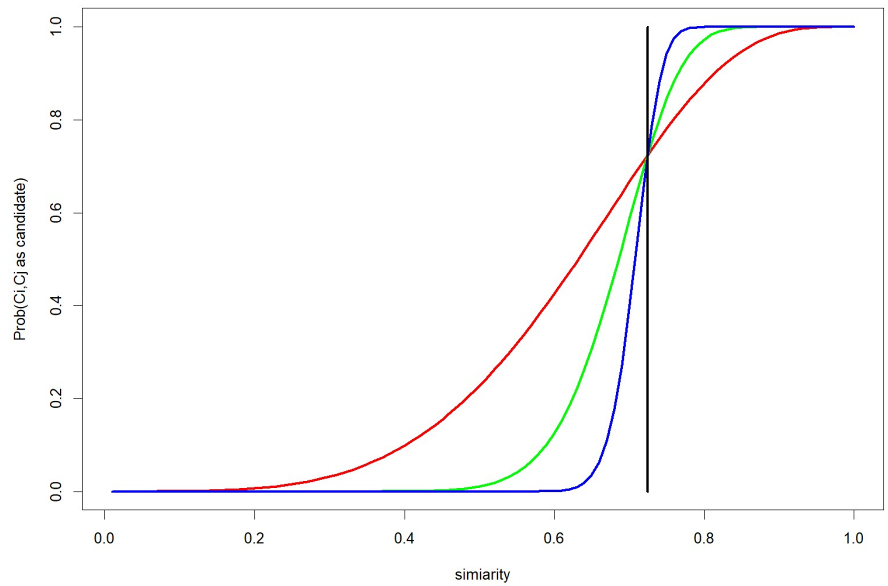
Demo

- https://www.desmos.com/calculator/lzzvfjiujn



- For each AND-OR S-curve 1-(1-sr)b, there is a *threshold* *t*, for which 1-(1-t^r)^b = t
- Above *t*, high probabilities are increased; below *t*, low probabilities are decreased
- You improve the sensitivity as long as the low probability is less than *t*, and the high probability is greater than *t*
- Iterate as you like (computation is not an issue)

- RED curve AND-OR
- GREEN curve AND-OR + AND-OR
- BLUE curve AND-OR + AND-OR + AND-OR
- Threshold 0.73



- Similar observation for the OR-AND type of S-curve: (1-(1-s)^b)^r

Distance Measures

Until now: Jaccard Distance

- We talked about **Jaccard Distance and Hamming**
  - measures how close are two sets
- There are other relevant functions to measure the closeness (distance)
- Lets recall some properties of distance metrics:
  - $d(x, y) \geq 0$ (no negative distances are allowed)
  - $d(x, y) = 0 \Leftrightarrow x = y$ (distances are positive, except a point from itself)
  - $d(x, y) = d(y, x)$ (distances are symmetric)
  - $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

Hamming Distance

- Used with spaces of any vector
- It is computed as the number of components in which two vectors differ.
- **Example:**
  - x = 10101
  - y = 11110
  - d(x,y) = 3

Hamming Distance

- Building an LSH family of function for **Hamming Distance** is quite simple
- Let x and y be two **d-dimensional vectors**
  - ...we denote with $\boldsymbol{h(x,y)}$ their hamming distance
- We define a function $\boldsymbol{f_i(x)}$ to be the *i-th* position of vector x
  - $\boldsymbol{f_i}(x) = \boldsymbol{f_i}(y)$ if and only if x and y agree in the *i-th* position.
  - the **probability of** $\boldsymbol{f(x) = f(y)}$ can be computed as $\boldsymbol{1 - h(x,y)/d}$
- The **family F** consisting of the functions previously defined is

$$\boldsymbol{(d_1, d_2, 1 - d_1/d\,, 1 - d_2/d) - sensitive}$$

- with $\boldsymbol{d_1 < d_2}$

Cosine Distance

- It is a measure defined in **Euclidean spaces** and **discrete Euclidean spaces**.
- It is the cosine of the **angle** that the vectors associated to those points make.
- d(A, B) = θ= arccos(A · B / ‖A‖·‖B‖)
- Has range $[\mathbf{0}, \boldsymbol{\pi}]$
- The angle is always in the **range 0 to 180 degrees**.

$$d([x_1, \dots, x_n], [y_1, \dots, y_n]) = 1 - \frac{A \cdot B}{\|A\| \|B\|}$$

$$A \cdot B = \|A\| \|B\| \cos \theta$$

Cosine Distance

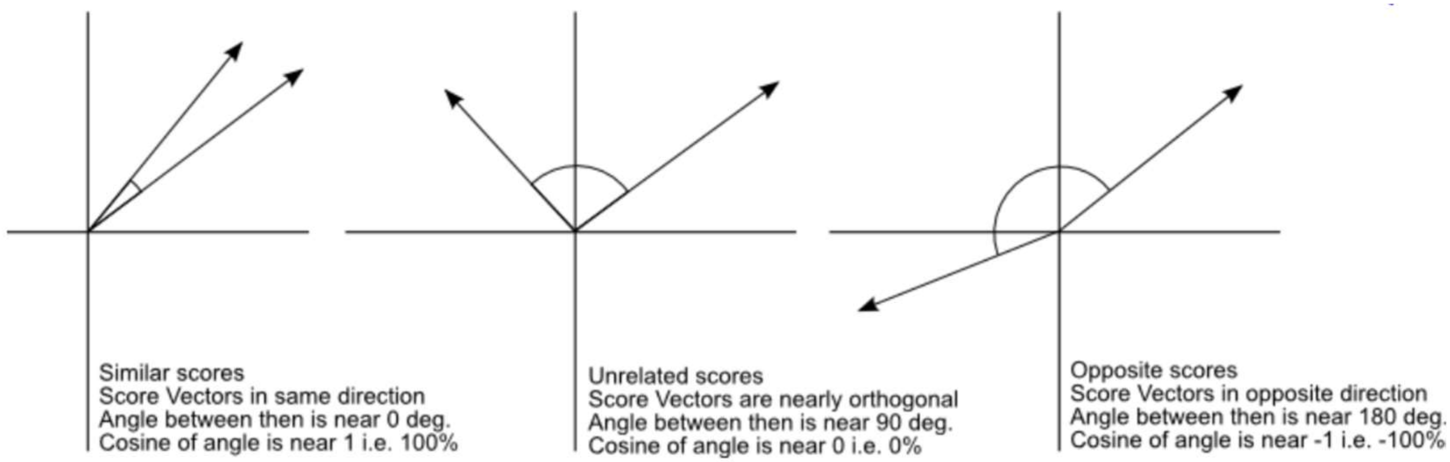
- What is Cosine Distance between x and y?
  - the cosine of the angle between them
- For cosine distance, there is a technique
  - called Random Hyperplanes
  - Technique similar to Min-Hashing
- Lets pick a random hyperplane and its normal vector
  - ...then compute the dot products $v \cdot x$ and $v \cdot y$
    - two cases: $sign(v \cdot x) = sign(v \cdot y)$ or $sign(v \cdot x) \neq sign(v \cdot y)$

Cosine Distance

- We define a function $\boldsymbol{f(x)}$ by choosing a random vector $\boldsymbol{v_f}$
  - $\boldsymbol{f(x) = f(y)}$ if and only $sign(v_f \cdot x) = sign(v_f \cdot y)$.
- The **family F** consisting of the functions previously defined is
  $\boldsymbol{(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180) - sensitive}$
- with $\boldsymbol{d_1 < d_2}$

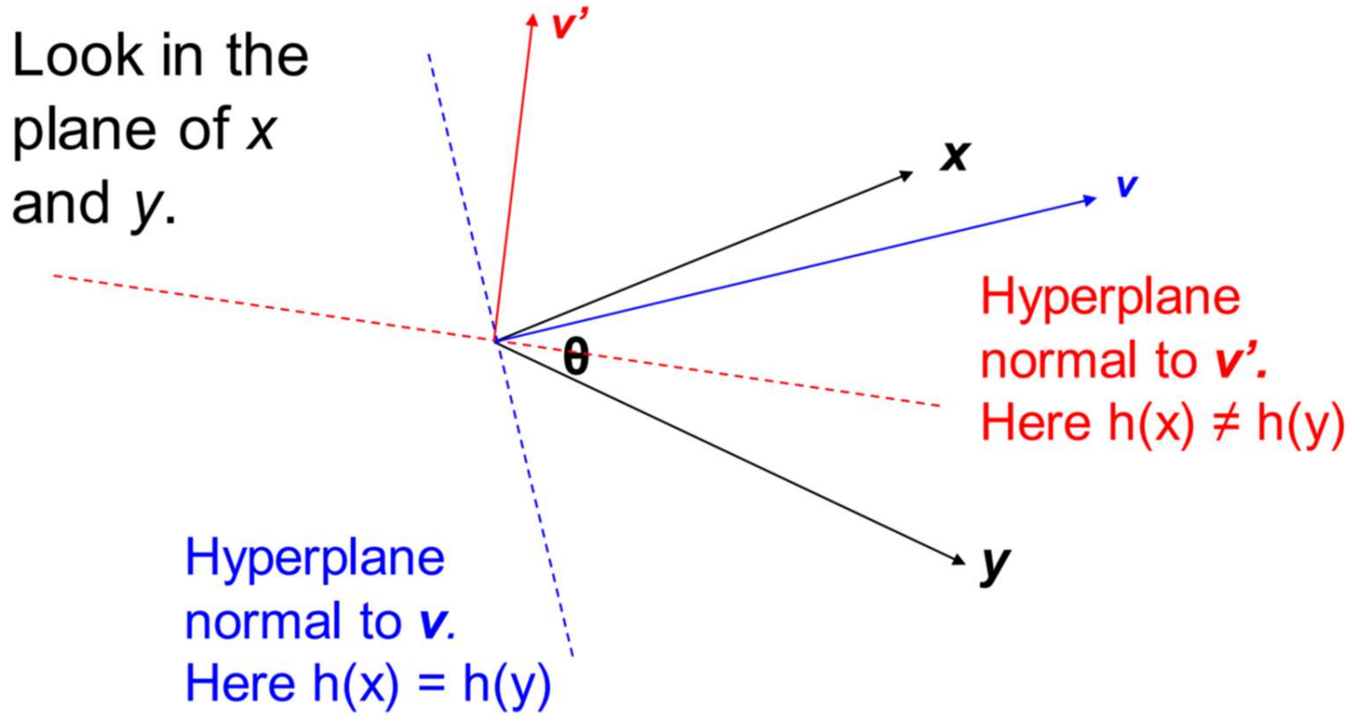
- Each vector ***v*** determines a hash function h_v with two buckets
- $h_v(x) = +1 \; if \; v \cdot x \geq 0$
- $h_v(x) = -1 \; if \; v \cdot x < 0$
- LS-family ***H*** = set of all functions derived from any vector
- **Claim:**
  - For points **x** and **y**, **Pr[h(x) = h(y)] = 1 – d(x,y) / π**

Random hyperplanes



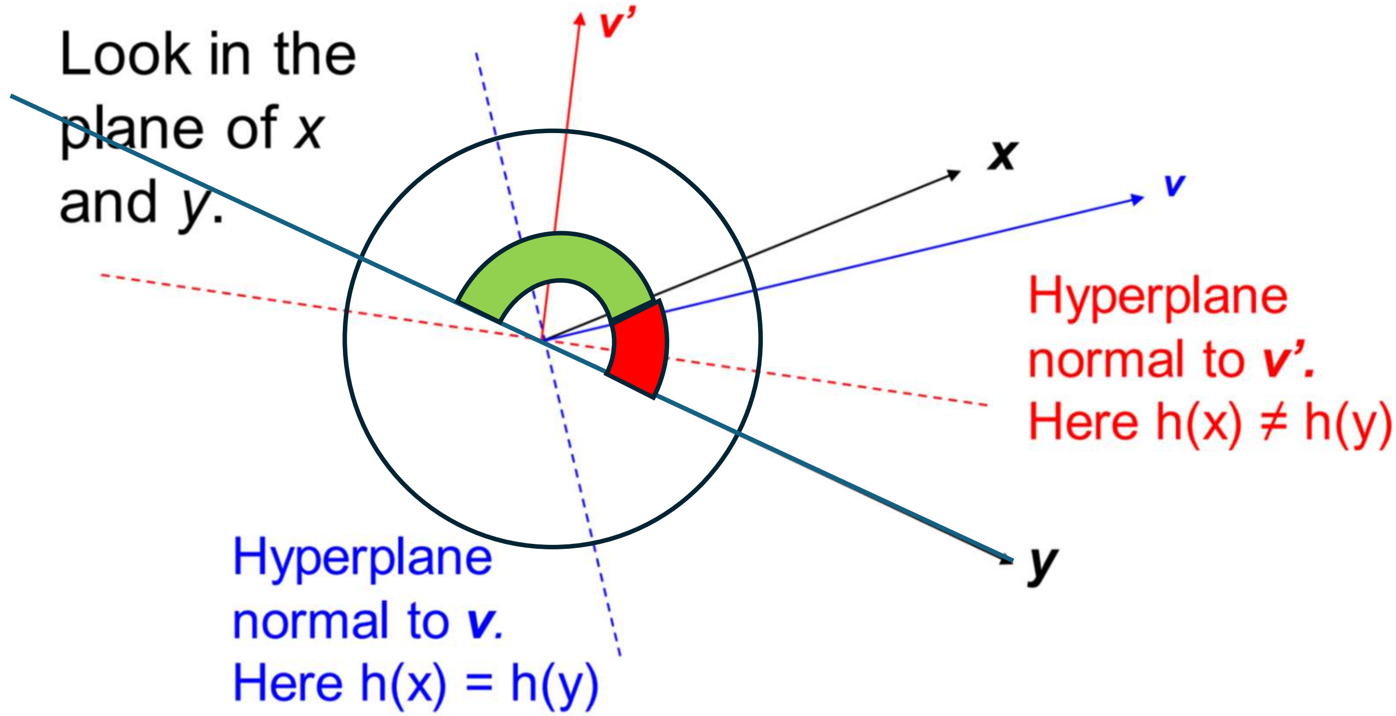
Consider points ***x*** and ***y***

- Let's analyze hyperplanes ***v*** and ***v'***



- $h_v(x) = +1$ if $v \cdot x \geq 0$
- $h_v(x) = -1$ if $v \cdot x \leq 0$

- Prob. Red case $=\theta/\pi$
- Prob green case 1- θ/π



Hot to buld the signature

- Pick some number of random vectors, and hash your data for each vector
- The result is a signature (*sketch*) of **+1**'s and **–1**'s for each data point
- Can be used for LSH like we used the Min-Hash signatures for Jaccard distance

- Amplify using **AND/OR** constructions

- We can demonstrate that by restricting our choice to vectors whose component are **+1** and **-1**, we don't loose randomness.

Distance in Euclidean spaces

- n-dimensional Euclidean space:
  - points are vectors of **n** real numbers.
- For any constant **r**, an **L_r-norm** is a measure defined as:
- $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \left(\sum_{i=1}^{n} |x_i - y_i|^r\right)^{1/r}$
- Some famous norms:
  - L_1-norm – **Manhattan Distance**
    - $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \sum_{i=1}^{n} |x_i - y_i|$
  - L_2-norm – **Euclidean Distance**
    - $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

Euclidean Distance

- Building an LSH function for Euclidean distance is much more difficult
- 2-dimensional example: lets pick a random horizontal line
  - project each point into such a line then divide it into segments of length $\boldsymbol{a}$
    - each segment is a bucket where our **function** $\boldsymbol{f(x)}$ hashes each point.
- We define **a family F** by choosing **random lines** through the space and a bucket size **“a”** that partitions each line.
  - We will hash points by projecting them onto the line

$$\boldsymbol{(d_1, d_2, p_1, p_2) - sensitive}$$

- We are not able to determine an expression for $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ but we can be certain that $\boldsymbol{p_1 > p_2}$ for each $\boldsymbol{d_1 < d_2}$

Random Projection for Euclidean Distance

- Let x be a random vector with coordinates selected randomly from a Gaussian Distribution $N(0,1)$
- $Let\ v\ a\ query\ point, h(v) = x \cdot v$ is the scalar product of the two vectors.
- The scalar projection is then quantized into a set of hash bins with the intention that close items in the original space will fall in the same bin. The hash function is:

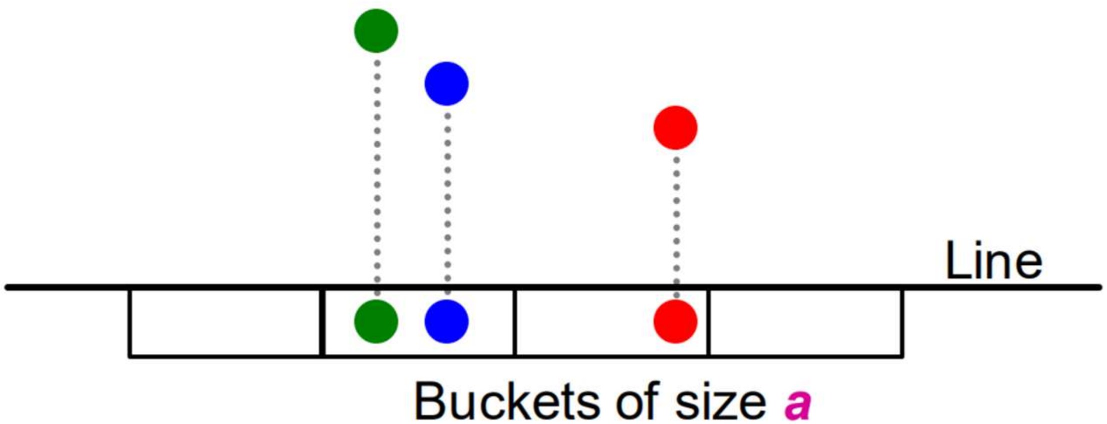
$$h^{x,b}(v) = \left\lfloor \frac{x \cdot v + b}{w} \right\rfloor$$

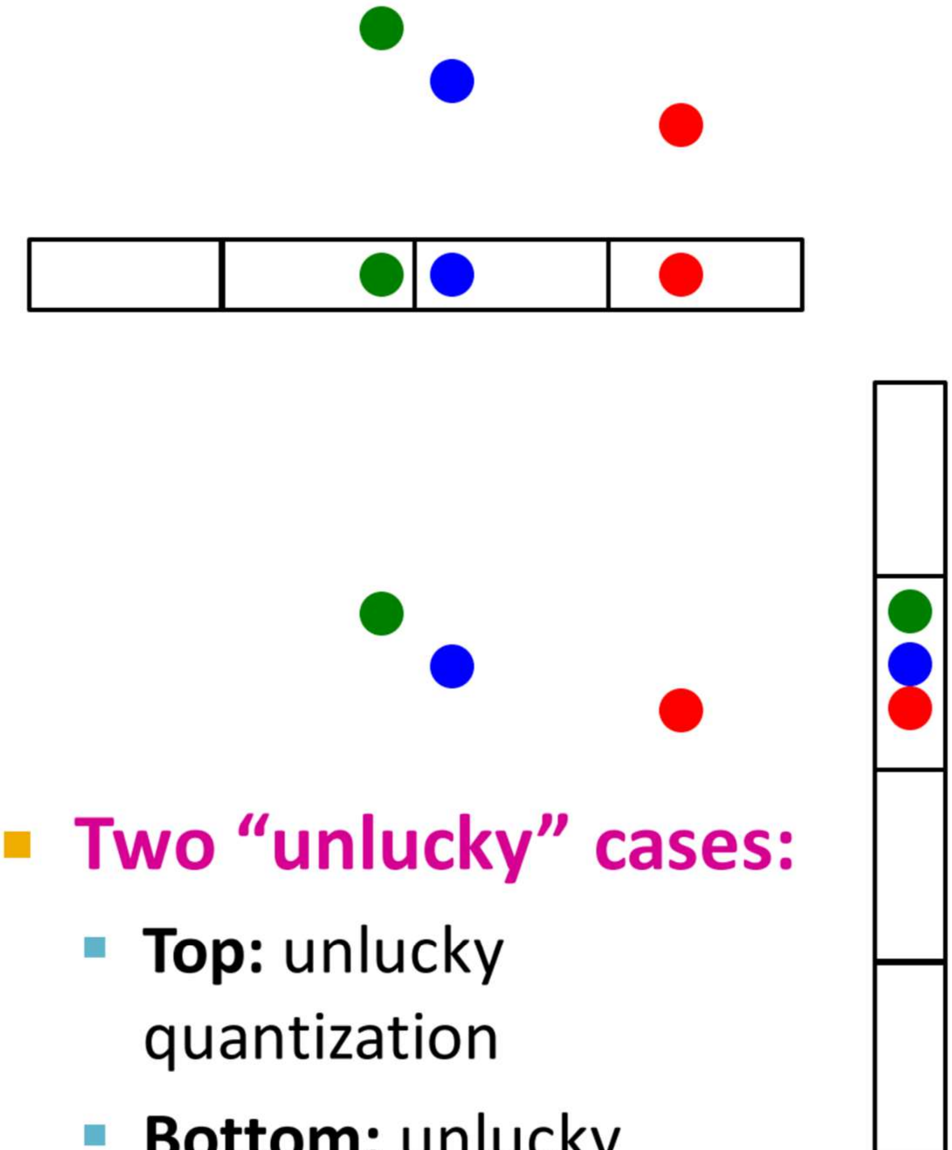
- *w* is the width of each quantization bin, and *b* is a random variable uniformly distributed between 0 and *w* that makes the quantization error easier to analyze, with no loss in performance.
- for any points *p* and *q* in R^d that are close to each other, there is a high probability $\boldsymbol{p_1}$ that they fall into the
- same bucket

$$P_h[h(p) = h(q)] \geq p_1\ for \|p - q\| \leq \boldsymbol{d_1}$$

- for any points *p* and *q* in R^d that are far apart, there is a low probability $\boldsymbol{p_2} < \boldsymbol{p_1}$ that they fall into the same bucket

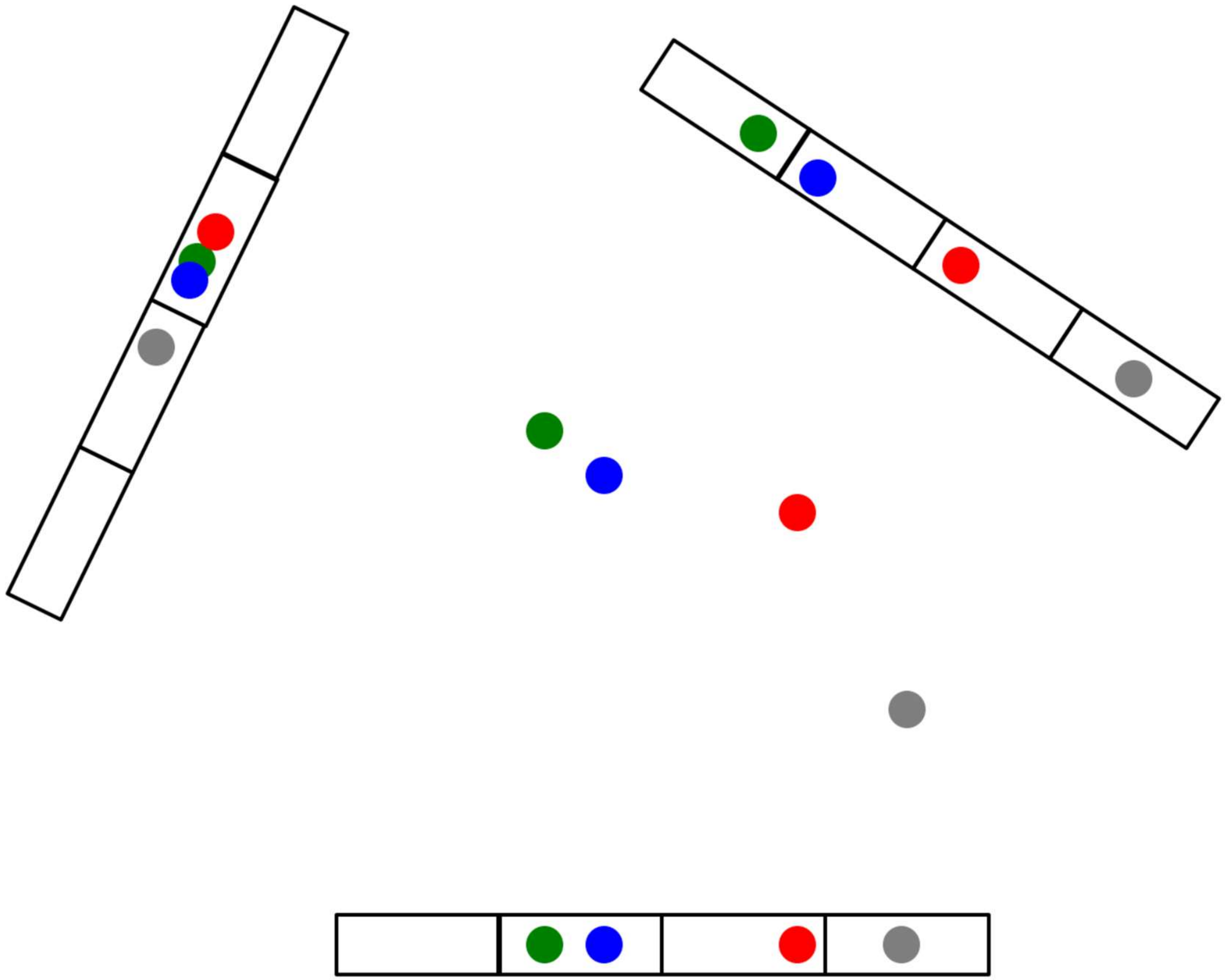
$$P_h[h(p) = h(q)] \leq p_2\ for \|p - q\| \geq c * d_1 = \boldsymbol{d_2}$$



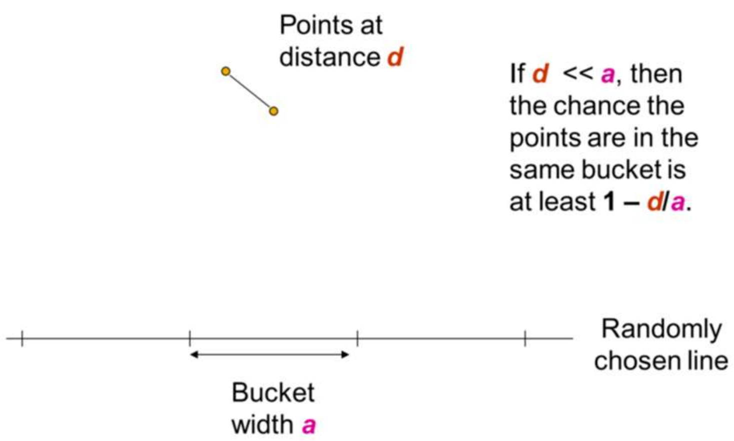


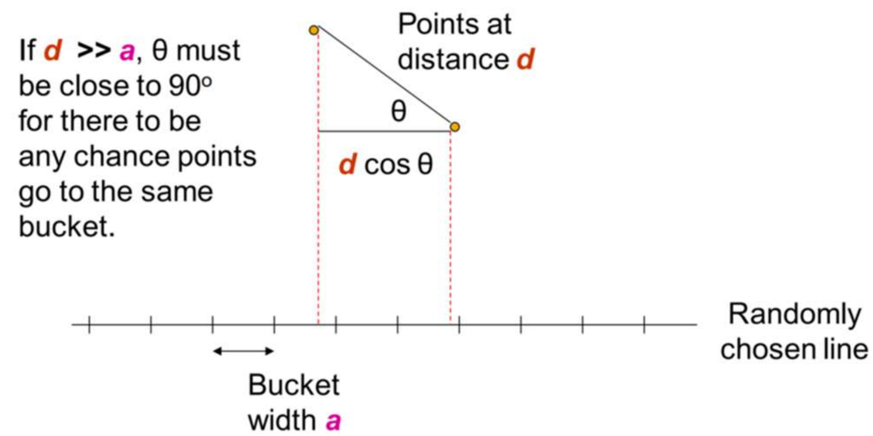
- **"Lucky" case:**
  - Points that are close hash in the same bucket
  - Distant points end up in different buckets

- **Two "unlucky" cases:**
  - **Top:** unlucky quantization
  - **Bottom:** unlucky projection



Two cases





- If points are distance $d < a/2$, prob. they are in same bucket $\geq 1 - d/a = ½$
- If points are distance $d > 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
- $\cos \theta \leq ½$
- $60 < \theta < 90$, i.e., at most 1/3 probability

- Yields a (a/2, 2a, 1/2, 1/3)-sensitive family of hash functions for any a

Edit distance

- Used with **strings**
- The distance between two strings **x** and **y** is the smallest number of insertions and deletions of single characters that will convert x to y.
- **Example:**
  - x = abcde
  - y = acfdeg
  - d(x,y) = 3
    - delete «b»
    - insert «f» after «c»
    - insert «g» after «e»

Other applications of LSH

- **Fingerprint matching:** comparing fingerprints
- **Newspaper articles matching:** distinguishing newspaper article from all the extraneous material
- **Entity resolution:** mapping data records that refer to the same entity (i.e. person)

Summary: 3 Steps

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity $\geq$ **s**

Notebook

- Spark Implementation MinHashLSH

https://colab.research.google.com/drive/1FTVN0dvm-eCGSEIF0d7x3i-PH7BM8yJb?usp=sharing