DATOS MASIVOS I

UNIDAD III MEDIDAS DE SIMILITUD Y DISTANCIA

FUNCIONES HASH SENSIBLES A LA LOCALIDAD

8 de Marzo de 2023

Hashing Sensible a la Localidad (LSH)

- Método para realizar búsqueda del vecino más cercano aproximado en espacios de alta dimensionalidad.

Hashing Sensible a la Localidad (LSH)

- La idea es proyectar el espacio original a otro de mucho menores dimensiones que preserve las distancias/similitudes entre los objetos de forma aproximada con alta probabilidad.

Hashing Sensible a la Localidad (LSH)

- La LSH generalizada se basa en algún tipo de "distancia" entre puntos.
  - Los puntos similares están "cerca".

Hashing Sensible a la Localidad (LSH)

- Norma L_1: suma de las diferencias en cada dimensión.
  - Distancia Manhattan = distancia si tuviéramos que recorrer sólo las coordenadas.

- Norma L_2: $d(x,y)$ = raíz cuadrada de la suma de los cuadrados de las diferencias entre *x* e *y* en cada dimensión.
  - La noción más común de "distancia".

- Distancia Hamming = número de posiciones en las que difieren los vectores de bits.

- Distancia de Jaccard para conjuntos = 1 menos la similitud de Jaccard.

- Distancia coseno = ángulo entre vectores desde el origen hasta los puntos en cuestión.

Hashing Sensible a la Localidad (LSH)

1. Una "función hash" es cualquier función que toma dos elementos y dice si son o no "iguales" (en realidad, son candidatos para la comprobación de similitud).
    - h(x) = h(y) significa "que *x* e *y* son iguales".

2. Una familia de funciones hash es cualquier conjunto de funciones como en (1).

Hashing Sensible a la Localidad (LSH)

- Para ello se define una familia de funciones H sensibles a la localidad para una distancia $dist(x^{(i)}, x^{(j)})$.

Hashing Sensible a la Localidad (LSH)

Para facilitar la terminología, llamaremos par candidato a dos elementos de datos que tienen el mismo hash.

Una función LSH toma como entrada un par *x* e *y* da como resultado. Sí *x* e *y* son un par candidato y no lo son en caso contrario. Es decir, h(x) = h(y) implica que h declara que *x* e *y* son un par candidato.

Familia de Funciones Sensibles a la Localidad

- Una familia de funciones $\mathcal{H} = \{h : \mathbb{R}^d \rightarrow \mathbb{Z}\}$ se llama *sensible a la localidad* para una distancia *dist* si para cualquier par $\{\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\} \in \mathbb{R}^d$, existen números reales r_1, r_2, p_1, p_2 tal que:

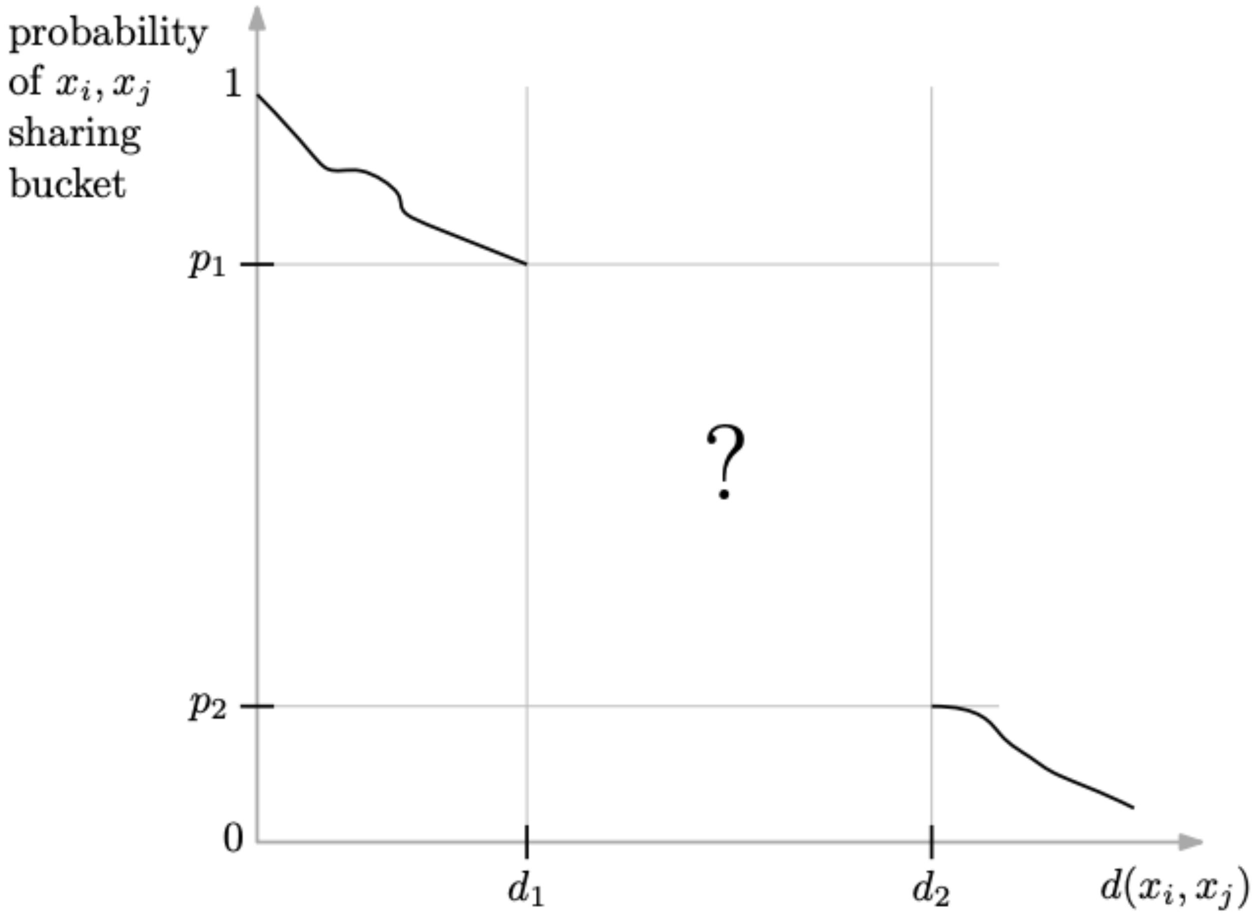
$$dist(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \leq r_1 \Rightarrow P[h(\mathbf{x}^{(i)}) = h(\mathbf{x}^{(j)})] \geq p_1$$
$$dist(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \geq r_2 \Rightarrow P[h(\mathbf{x}^{(i)}) = h(\mathbf{x}^{(j)})] \leq p_2$$

donde $r_1 < r_2$.

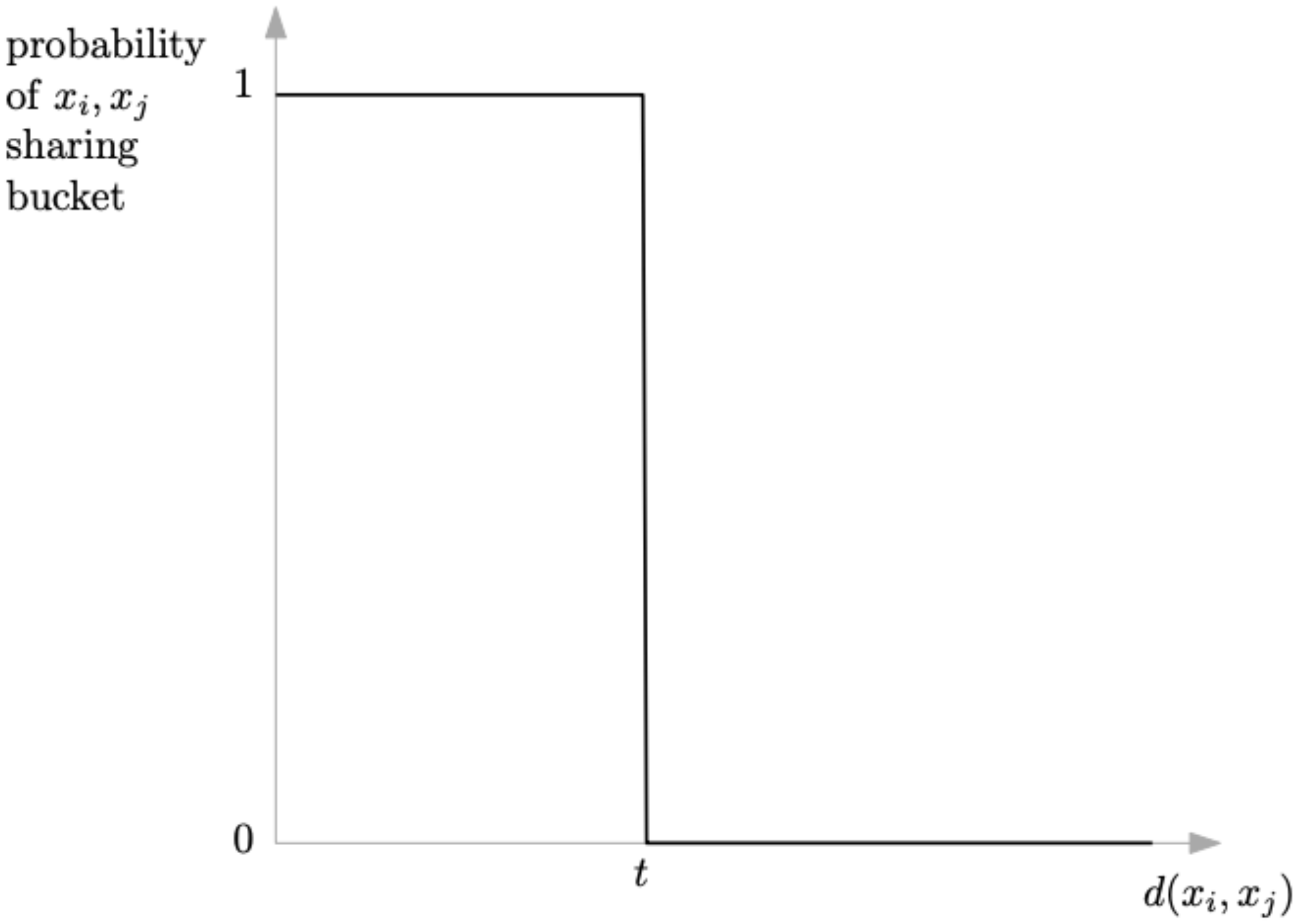
- En general, es deseable que $p_1 \gg p_2$.

Familia de Funciones Sensibles a la Localidad

- Se garantiza que, si $d(x, y) \leq d_1$ entonces:
- Los elementos irán al mismo bucket con probabilidad al menos p_1.
- Y si $d(x, y) \geq d_2$ entonces los ítems irán al mismo cubo con probabilidad como máximo p_2.



Caso ideal en términos de distancia.



Familia de Funciones Sensibles a la Localidad

La primera afirmación limita los falsos negativos, que son pares altamente similares pero que no *hashean* al mismo bucket.

La segunda limita los falsos positivos, que no son pares altamente similares pero que sí hashean al mismo bucket.

Familia de Funciones Sensibles a la Localidad

El problema de la primera es que no habla de falsos positivos.

El problema de la segunda es que no habla de falsos negativos.

Familia de Funciones Sensibles a la Localidad

Teniendo en cuenta la tarea de detección de casi duplicados, nos gustaría que p_1 fuera muy alto (cercano a 1) para reducir los falsos negativos.

Del mismo modo, nos gustaría que p_2 fuera muy bajo (cercano a 0) para reducir los falsos positivos.

Además, como no tenemos ninguna garantía sobre los pares cuya distancia está entre d_1 y d_2, también nos gustaría que tanto d_1 como d_2 estuvieran cerca de t (umbral de casi duplicados) para reducir el rango de distancias sin garantías.

Tuplas de Funciones LSH para FP y FN

- Para ampliar el margen entre p_1 y p_2, se generan l tuplas $g_1, \ldots g_l$ de r funciones *hash*[1]:

$$\begin{aligned} g_1 &= (h_{11}, \ldots, h_{1r}) \\ \vdots \quad & \qquad \vdots \\ g_l &= (h_{l1}, \ldots, h_{lr}) \end{aligned}$$

- Se pueden ver como una familia de funciones con $d_1, d_2, (p_1)^r, (p_2)^r$.
- Para buscar se construyen l tablas (una por tupla) y se almacena cada punto en la cubeta correspondiente.[2]

[1] Sacadas de forma independiente y uniforme de $\mathcal{H}$

[2] Esto se logra mediante una función *hash* universal que toma la tupla y la mapea a un índice de la tabla.

Tuplas de Funciones LSH para FP y FN

Tratamiento de los falsos positivos.

Utilizamos varias funciones hash independientes de H y consideramos los pares que son declarados candidatos por todas ellas. Esto puede verse como una operación AND sobre la salida de las funciones hash.

Hacemos el hash en los mismos buckets mediante el AND de (la salida de) estas funciones hash.

Tuplas de Funciones LSH para FP y FN

Tratamiento de los falsos positivos

Si dos elementos de datos tienen una distancia *grande*, es cada vez menos probable que se clasifiquen en el mismo bucket a medida que aumenta el número de funciones hash (debido a la construcción AND).

Por lo tanto, es menos probable que vectores distintos se conviertan en pares candidatos y habrá menos falsos positivos.

Tuplas de Funciones LSH para FP y FN

Tratamiento de los falsos negativos.

Para reducir los falsos negativos, damos a estos pares más oportunidades de convertirse en pares candidatos. Esto se consigue mediante una construcción OR.

Tuplas de Funciones LSH para FP y FN

Tratamiento de los falsos negativos.

Seutilizan varias funciones hash independientes de H y se consideran los pares declarados candidatos por cualquiera de ellas. Esto puede verse como una operación OR sobre la salida de las funciones hash.

Ahora es más probable que los vectores similares se conviertan en pares candidatos.

LSH para Distancia de Hamming

- Para vectores binarios $\mathbf{x}^{(i)} \in \{0, 1\}^d$ (o cadenas de bits de longitud d) y la distancia de Hamming, una familia LSH se construye obteniendo el valor de una posición j

$$h_j(\mathbf{x}^{(i)}) = x_j^{(i)}$$

- Más generalmente, esta familia de funciones se puede aplicar a vectores M-arios $\mathbf{x}^{(i)} \in \{0, 1, \ldots, M\}^d$.

LSH para Distancia de Hamming

	1	2	3	4	5	6	7	8	9
$\mathbf{x}$	1	0	1	1	0	1	1	0	0

$h'_1 = \{h_2, h_5, h_7\}$

$h'_2 = \{h_1, h_4, h_8\}$ $\quad h'_1(\mathbf{x}) \neq h'_1(\mathbf{y}) \quad h'_2(\mathbf{x}) \neq h'_2(\mathbf{y}) \quad h'_3(\mathbf{x}) = h'_3(\mathbf{y})$

$h'_3 = \{h_6, h_9\}$

AND

	1	2	3	4	5	6	7	8	9
$\mathbf{y}$	0	1	1	0	0	1	0	1	0

	1	2	3	4	5	6	7	8	9
$\mathbf{x}$	1	0	1	1	0	1	1	0	0

$h''_1 = \{h_2, h_5, h_7\}$

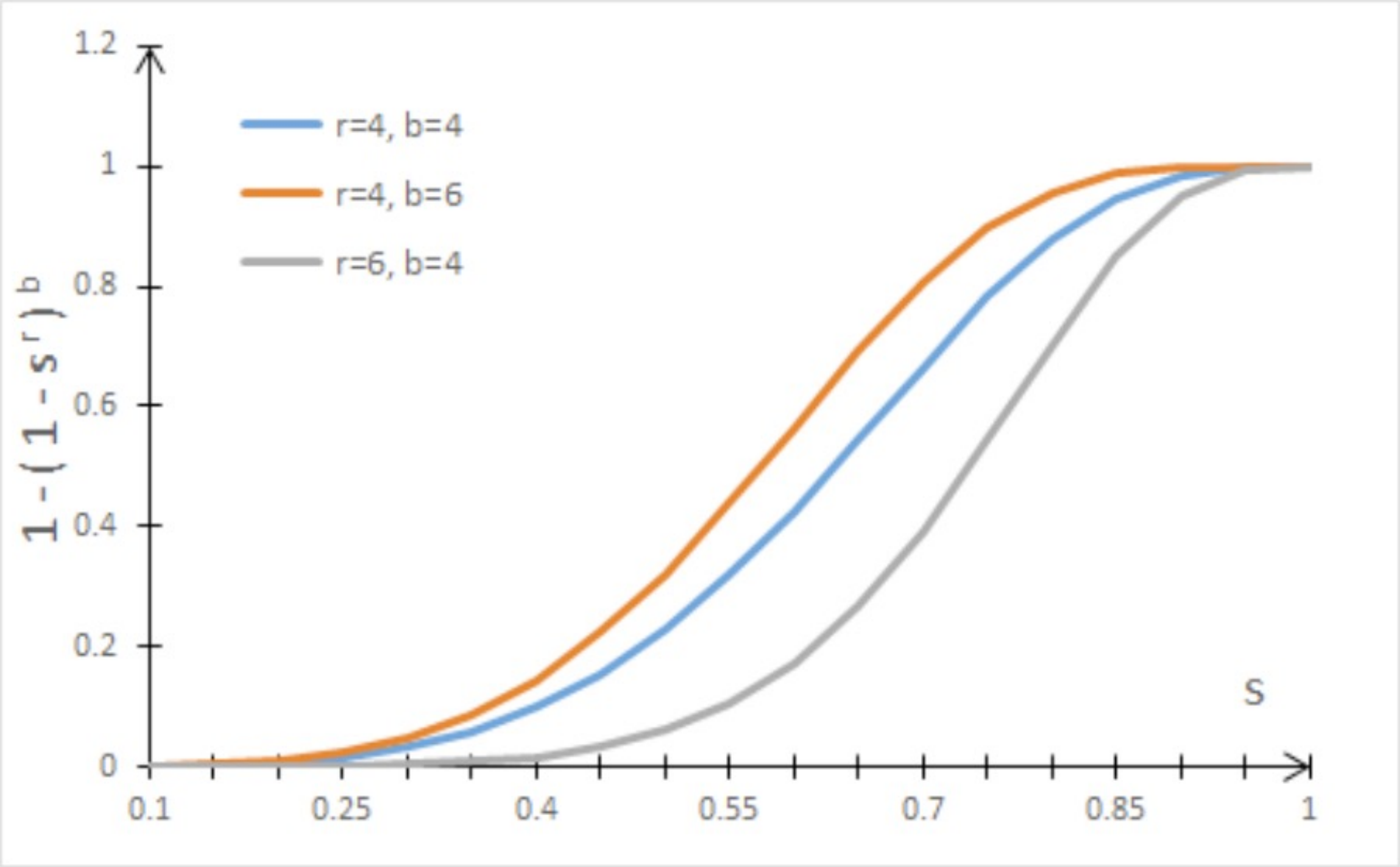
OR

$h''_2 = \{h_1, h_4, h_8\}$ $\quad h''_1(\mathbf{x}) = h''_1(\mathbf{y}) \quad h''_2(\mathbf{x}) \neq h''_2(\mathbf{y}) \quad h''_3(\mathbf{x}) = h''_3(\mathbf{y})$

$h''_3 = \{h_6, h_9\}$

	1	2	3	4	5	6	7	8	9
$\mathbf{y}$	0	1	1	0	0	1	0	1	0

LSH para Distancia de Hamming



Extensión a Distancia ℓ_1

- Sean $\{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ puntos en un espacio de d dimensiones y C el valor máximo de cualquier coordenada, cada punto se transforma a un vector de Cd bits:

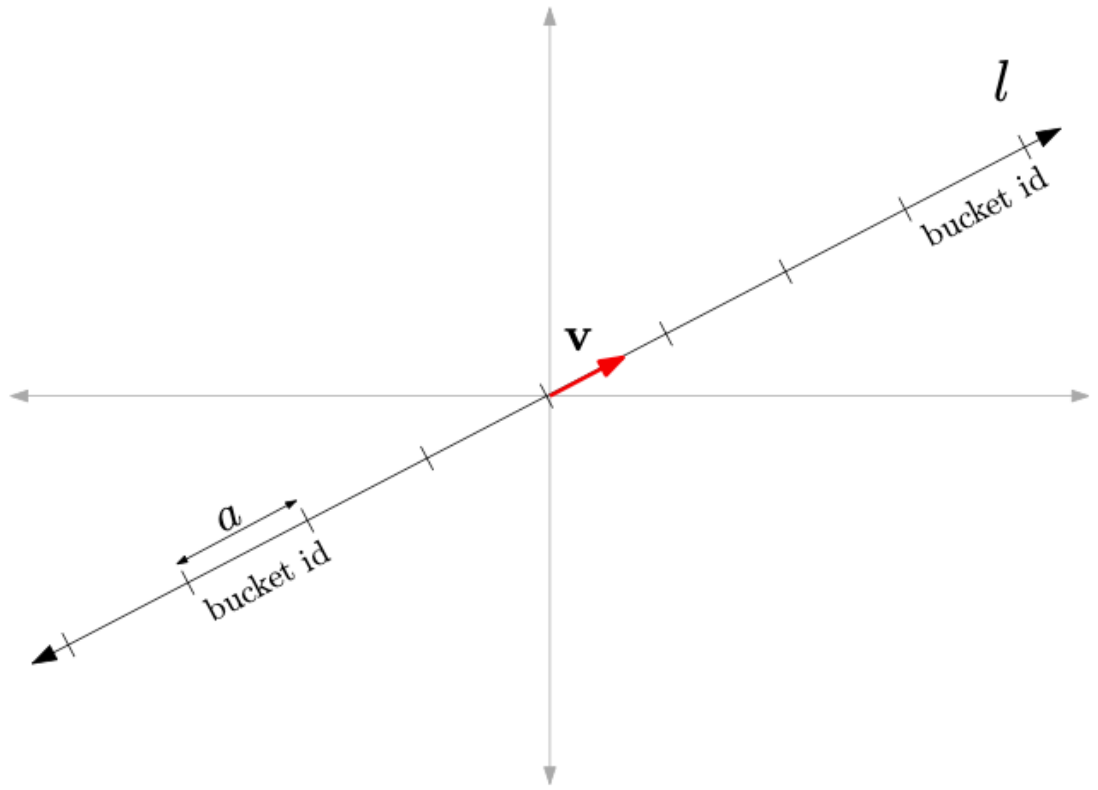
$$f(\mathbf{x}^{(i)}) = [t(x_1); t(x_2); \cdots; t(x_d)]$$

  donde $t(x_k)$ es una cadena de bits con x_k unos seguidos de $C - x_k$ ceros.

- La distancia de Hamming sobre $f(\mathbf{x}^{(i)})$ y $f(\mathbf{x}^{(j)})$ es igual a la distancia ℓ_1 sobre $\mathbf{x}^{(i)}$ y $\mathbf{x}^{(j)}$

LSH para Distancia Euclideana

La idea general de LSH para la distancia euclidiana es que, si dos puntos están "cerca" y los proyectamos sobre otro vector, entonces deberían permanecer "cerca" el uno del otro.



LSH para Distancia Euclideana

La función $h_v = h_l$ (correspondiente a la línea l o al vector unitario v) asigna un vector x a un bucket (segmento de l) donde se encuentra la proyección de x sobre l.

$$h_v(x) = \left\lfloor \frac{\langle \mathbf{x}, \mathbf{v} \rangle}{a} \right\rfloor$$

Esencialmente, h_v proyecta x sobre v y luego discretiza la proyección en un múltiplo de a.

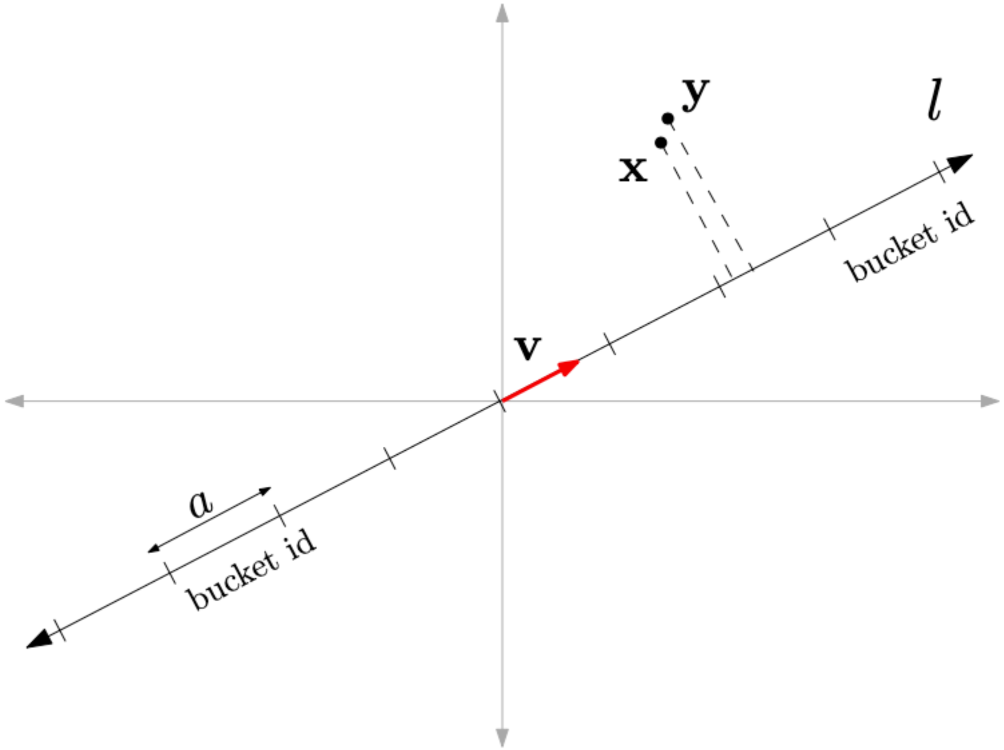
LSH para Distancia Euclideana

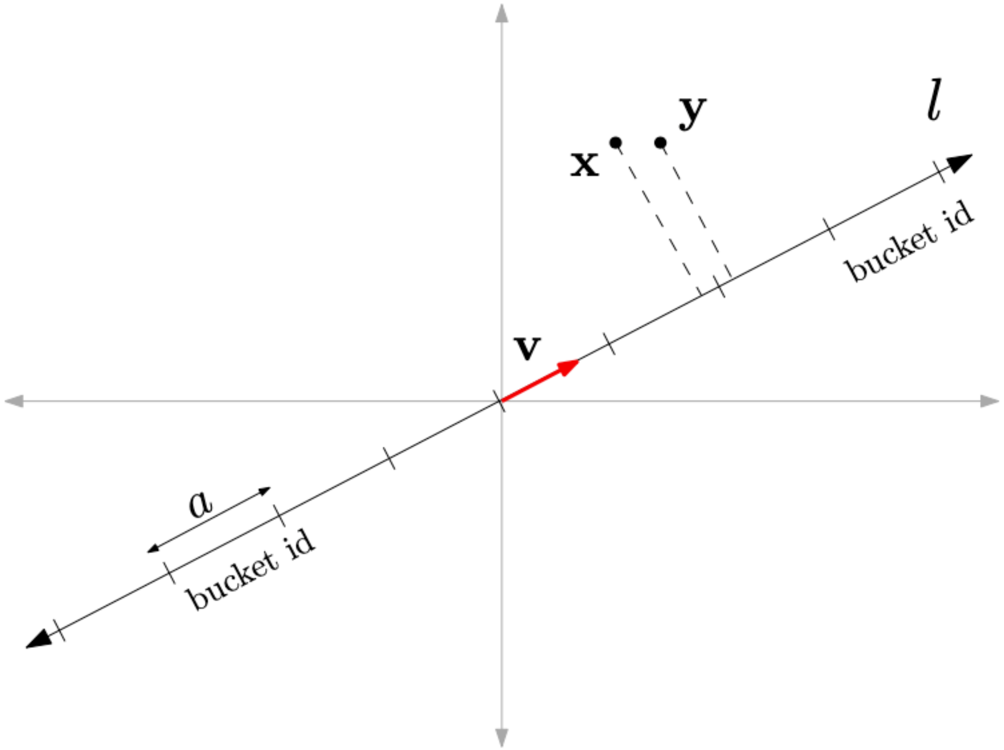
Sea *d* la distancia entre dos puntos *x* e *y*.
Dependiendo de que tan grande o pequeño sea *d* en comparación con *a*, podemos calcular las probabilidades de que *x* e *y* caigan o no en el mismo bucket.

$$Pr[h_v(x) = h_v(y)] \rightarrow d(x, y)$$

Si *d(x, y)* es pequeño en comparación con *a*, entonces es probable que *x* e *y* caigan en el mismo bucket, aunque no es necesario.

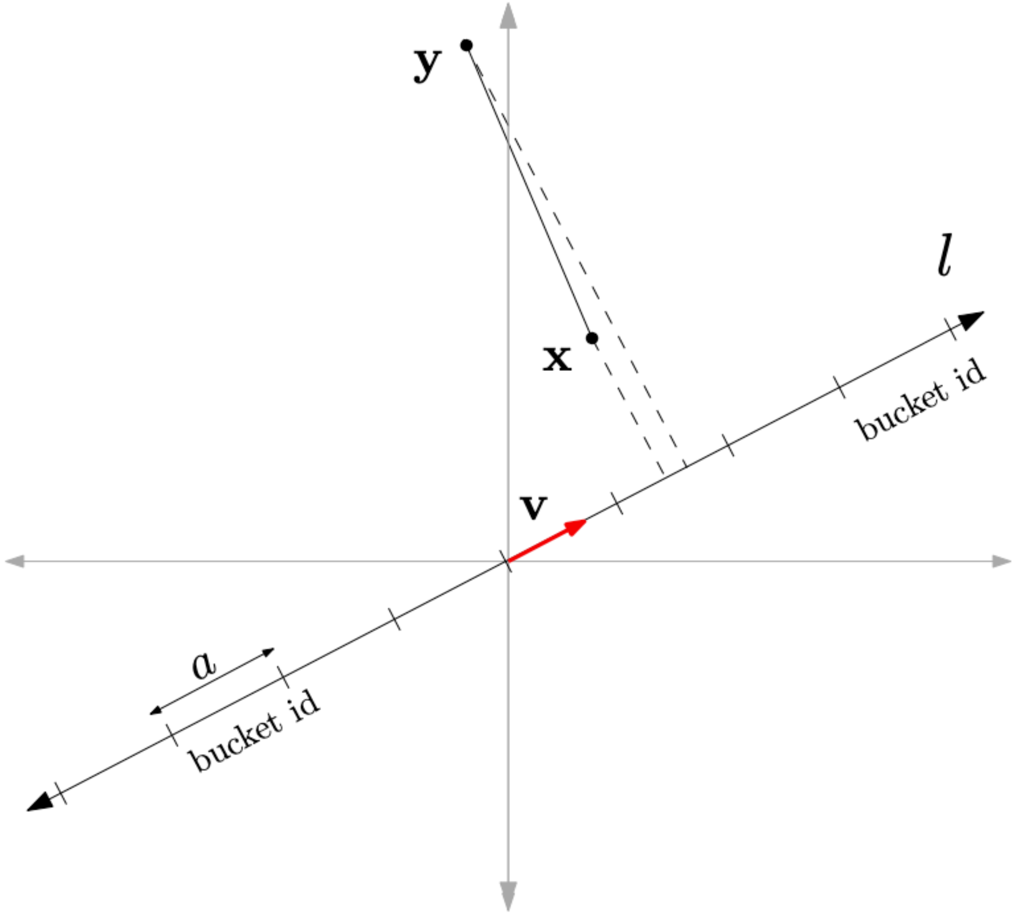
LSH para Distancia Euclideana

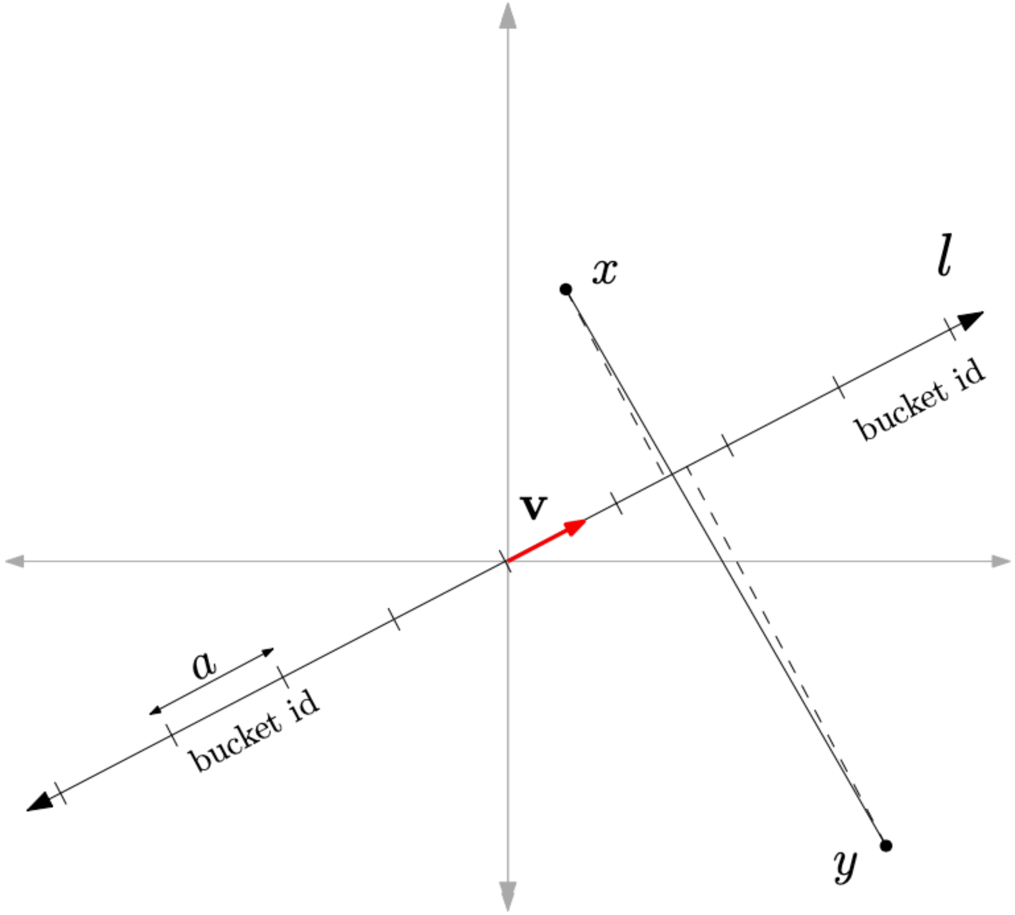




LSH para Distancia Euclideana

En otras palabras, si *d* es grande, el ángulo tiene que ser cercano a 90 grados para que vayan al mismo bucket.





LSH para Distancia Angular

- Dado un par de puntos $\{\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\} \in \mathbb{R}^d$, el ángulo entre ellos se obtiene de la siguiente manera:

$$\theta(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \arccos\left(\frac{\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}}{\|\mathbf{x}^{(i)}\| \cdot \|\mathbf{x}^{(j)}\|}\right)$$

- Una familia LSH para la distancia angular $1 - \theta(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ es:

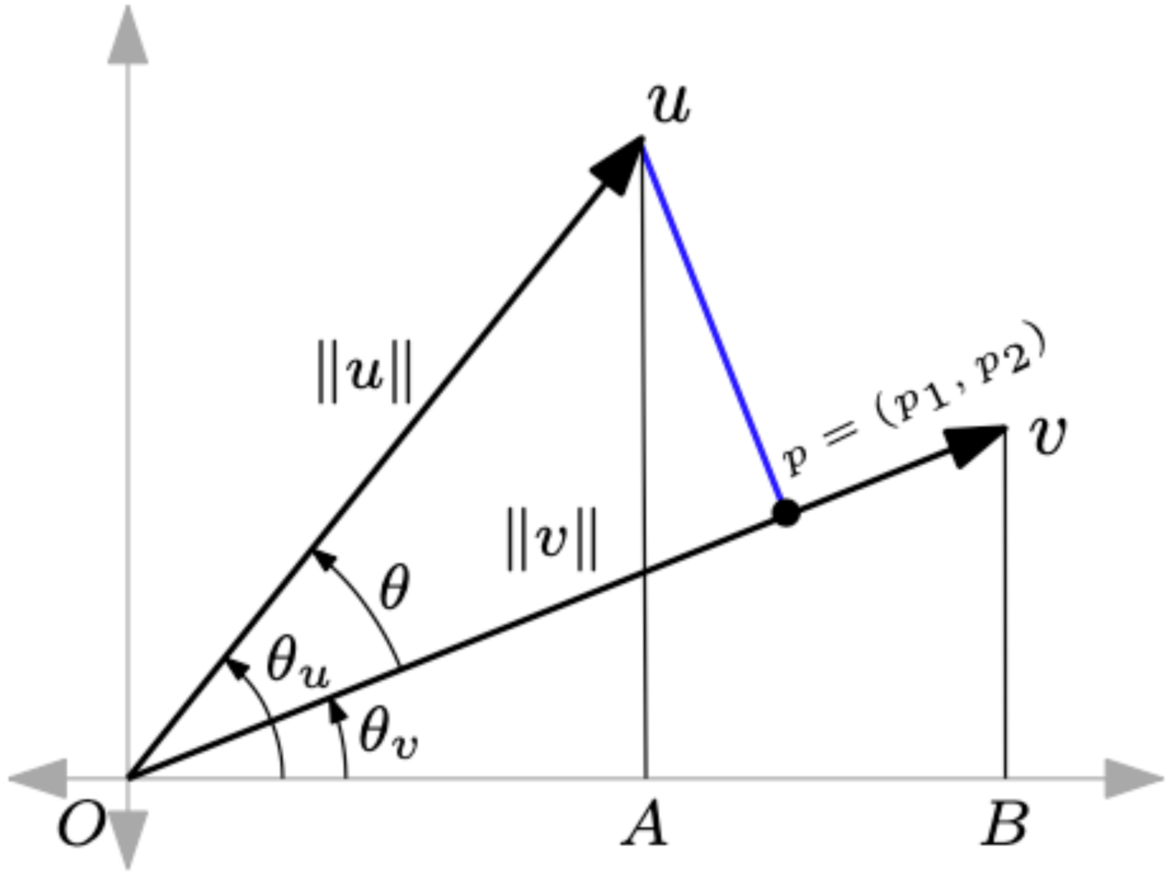
$$h_{\mathbf{v}}(\mathbf{x}^{(i)}) = signo(\mathbf{v} \cdot \mathbf{x}^{(i)})$$

  donde $\mathbf{v} \in \mathbb{R}^d$ es un vector aleatorio de tamaño unitario.

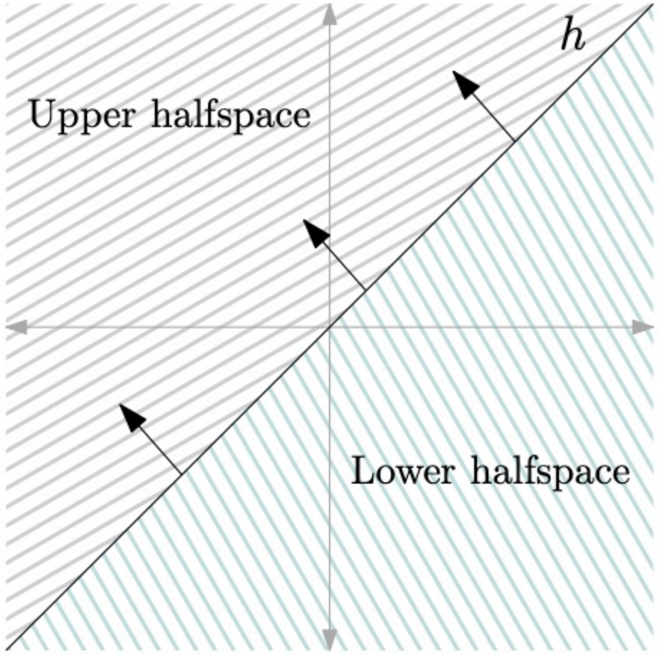
- Esta función se puede ver como dividir el espacio en 2 por un hiperplano elegido aleatoriamente

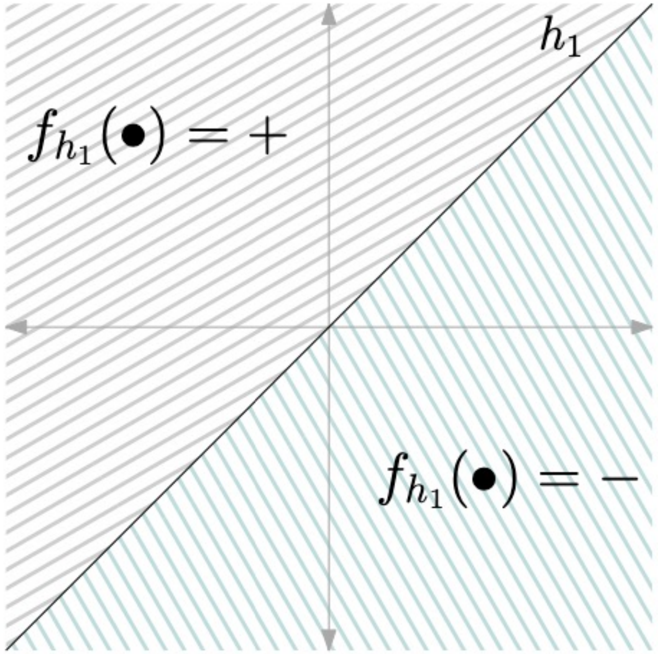
$$Pr[h_{\mathbf{v}}(\mathbf{x}^{(i)}) = h_{\mathbf{v}}(\mathbf{x}^{(j)})] = 1 - \frac{\theta(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})}{\pi}$$

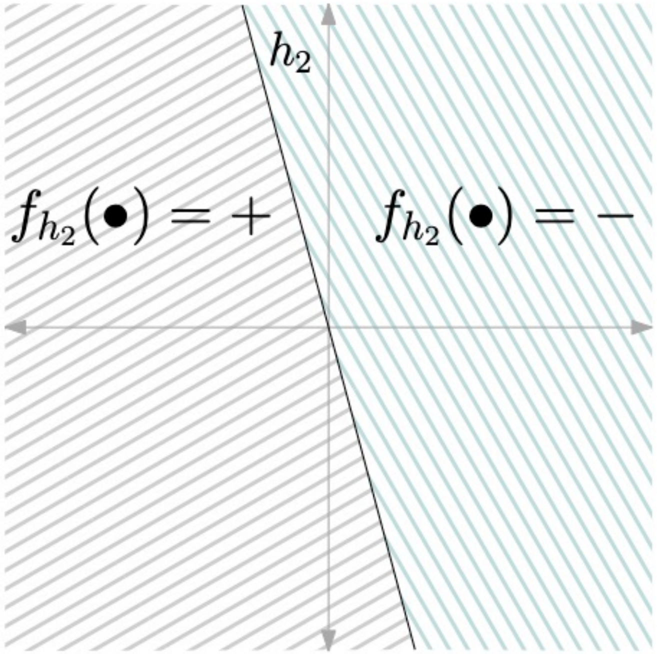
LSH para Distancia Angular



LSH para Distancia Angular







LSH para Distancia Angular

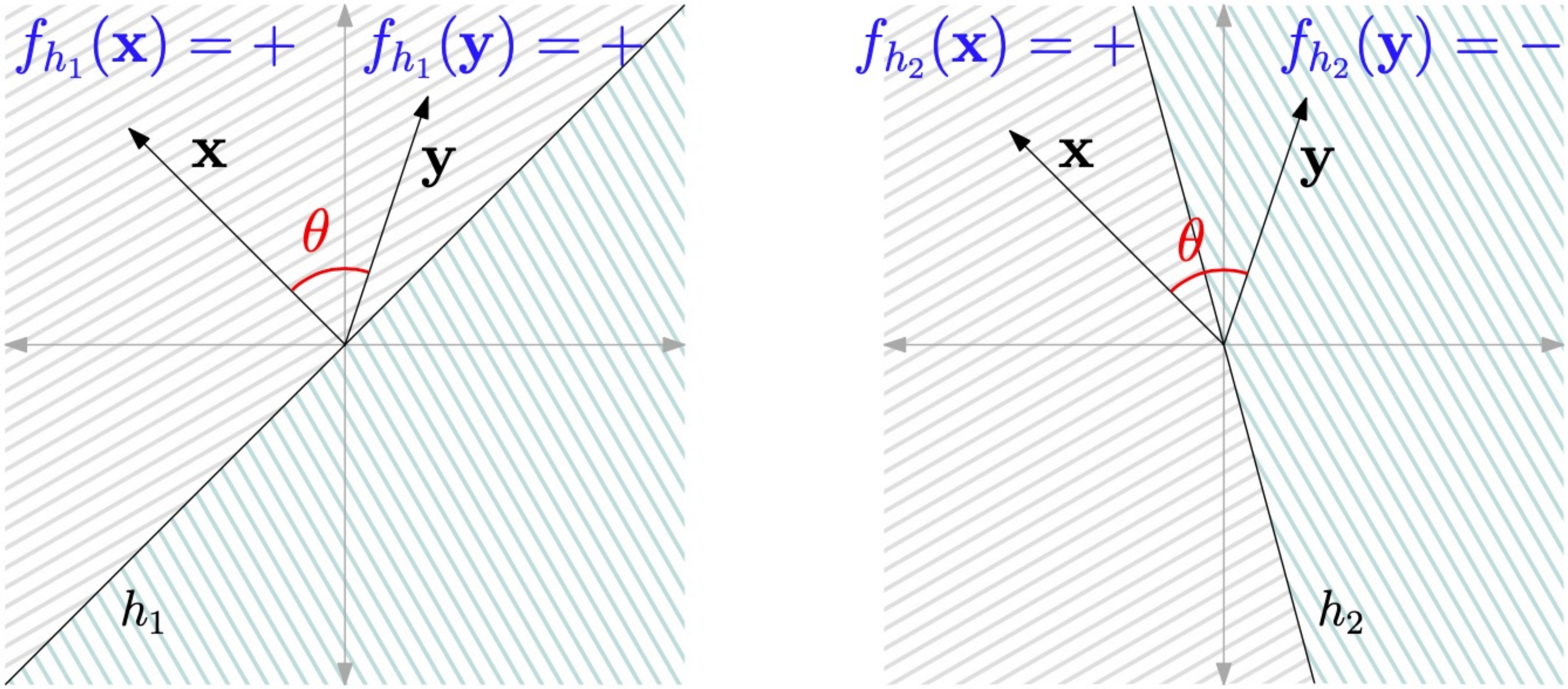
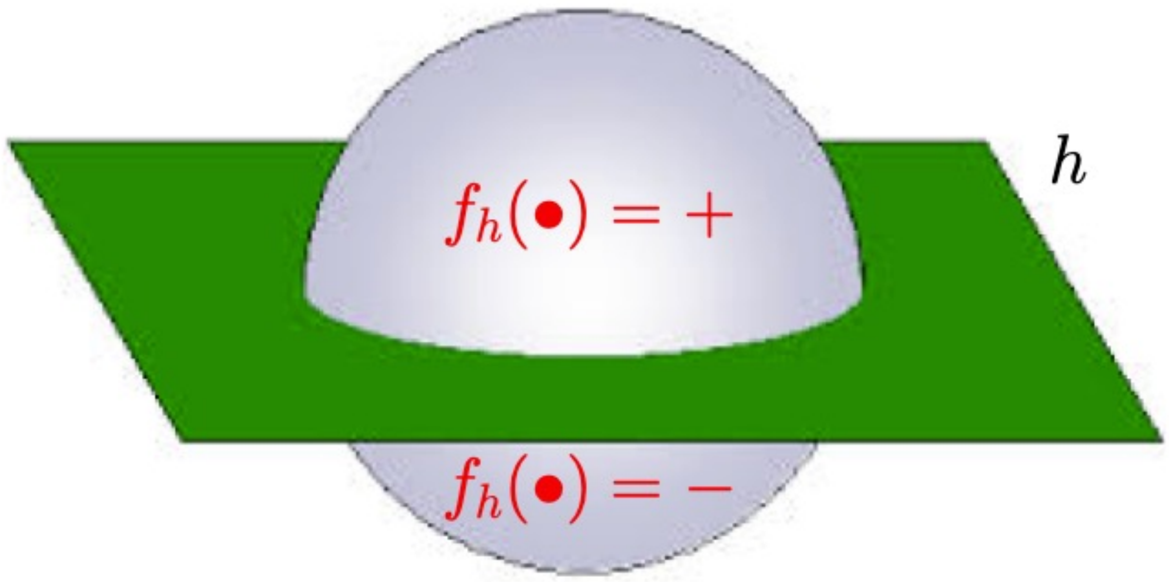


Figure 16 Vector $\mathbf{x}$ and $\mathbf{y}$ is a candidate pair under f_{h_1} (left) but not under f_{h_2} (right)

LSH para Distancia Angular



LSH para Distancia Angular

- La distancia coseno es el ángulo entre dos vectores en el intervalo [0 - 180] y se calcula computando primero el coseno del ángulo entre los dos vectores y calculando después el arc-coseno para obtener el ángulo.

- Aquí ignoramos la magnitud del vector y sólo consideramos su dirección, es decir, un vector es igual a un vector unitario en esa dirección.

LSH para Distancia Angular

- Cada hiperplano divide el espacio en dos mitades (que denominamos superior o positivo e inferior o negativo).

- La función f_h respecto a un hiperplano h asigna cada vector del semiespacio superior al bucket + y cada vector del semiespacio inferior al bucket –

- Para un hiperplano h, decimos que dos vectores u y v comparten un bucket (se convierten en pares candidatos) si la función hash correspondiente f_h los asigna al mismo cubo, es decir, f_h(u) = f_h(v), de lo contrario no se convierten en pares candidatos.

LSH para detección de plagio

Supongamos que tenemos 1M de documentos, cada uno de longitud 2,000.

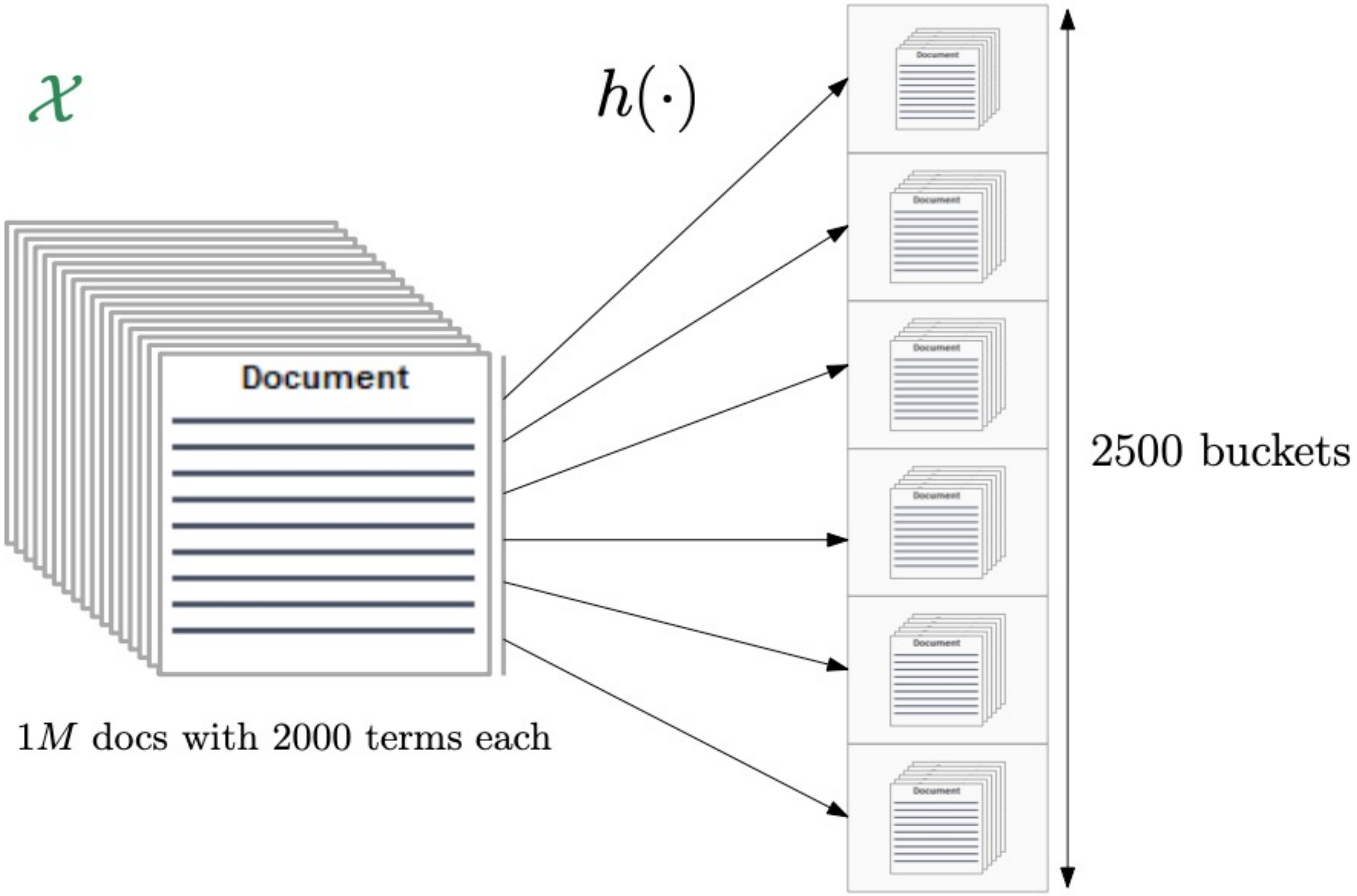
Nuestro objetivo es encontrar duplicados, es decir, los pares de documentos cuya similitud es superior al 90% = 0,9 (la distancia es inferior al 10% = 0,1).

LSH para detección de plagio

El método ingenuo (fuerza bruta) consiste en calcular todas las distancias entre pares y mostrar las que son inferiores a 0,1.

- Hay un total de medios $\binom{1M}{2} = 10^{12}$pares y cálculo de distancias entre un par llevará al menos 2,000 pasos (por ejemplo, distancia euclideana o coseno).
- Por tanto, el número total de operaciones es de $2x10^{15}$.

LSH para detección de plagio



LSH para detección de plagio

- Supongamos ahora que utilizamos una función aleatoria h de una familia H de (0.15, 0.4, 0.8, 0.2) funciones LSH.

- Según las propiedades de estas funciones, si elegimos al azar una función h de H, para dos documentos cualesquiera *x* e *y*.

LSH para detección de plagio

- Supongamos ahora que utilizamos una función aleatoria h de una familia H de (.15, .4, .8, .2) funciones LSH.

- Según las propiedades de estas funciones, si elegimos al azar una función h de H, para dos documentos cualesquiera *x* e *y*.

  - **If** $d(x, y) \leq .15$, **then** $Pr[h(x) = h(y)] \geq .8$
  - **If** $d(x, y) \geq .40$, **then** $Pr[h(x) = h(y)] \leq .2$

LSH para detección de plagio

- Cada bucket tiene 1M/2500 = 400 documentos.
- La distancia total entre pares son $2{,}500\binom{400}{2} = 200M$
- Cada cálculo de distancia requiere 2,000 pasos.
- Por lo que el número total de operaciones es 200 × 2,000 ≈ $4x10^{11}$ operaciones.
- Esto es unas 2,500 veces más rápido que el método ingenuo.

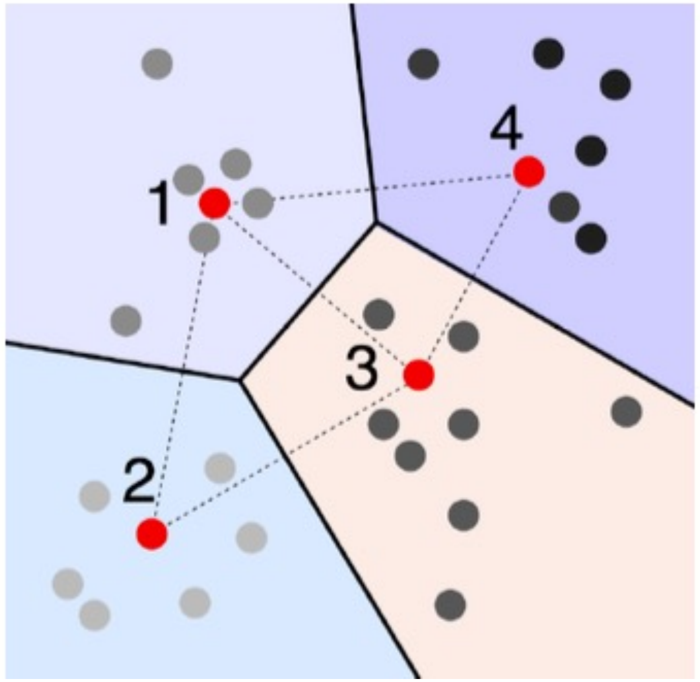
LSH dependientes de los datos

- Todos los esquemas LSH que hemos visto son sensibles a una medida de distancia específica.

- No tienen en cuenta los datos (no miran los datos).

LSH dependientes de los datos

- Todos los esquemas LSH que hemos visto son sensibles a una medida de distancia específica.
- Sin embargo, no tienen en cuenta los datos (no miran los datos).
- Un esquema LSH que depende de los datos es Clustering LSH.
- Agrupa conjuntos de datos en *k* clusters (utilizando algún método y medida de proximidad) y cada cluster sirve como un hash bucket.
- El ID de bucket de cada punto es su ID de clúster.



Otras distancias *no hashables*

Sorensen-Dice

$$sim_{sd}(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

Overlap similarity

$$sim_{ov}(X,Y) = \frac{|X \cap Y|}{\min\{|X|, |Y|\}}$$

Anderberg

$$sim_{an}(X,Y) = \frac{|X \cap Y|}{|X \cap Y| + 2|X \oplus Y|}$$

Rogers-Tanimoto

$$sim_{rt}(X,Y) = \frac{|X \cap Y| + |X \cup Y|}{|X \cap Y| + \overline{X \cup Y} + 2|X \oplus Y|}$$