UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

LSH – Imagenes
Locality Sensitive Hashing

Isabella Ariza Cuberos
Luisa María Zapata Saldarriaga

2024

Una Facultad abierta y transformadora

Contenido

1. El problema de búsqueda vecina
2. Familias de funciones sensibles a la localidad
3. Aplicación LSH en imagenes
4. Procesamiento de los datos
5. Implementación
6. Evaluación

Problemas del NNS en alta dimensionalidad

Nearest neighbor search

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

El **Nearest Neighbor Search (NNS)**, o búsqueda de vecinos más cercanos, es un problema clásico en el campo de la recuperación de información y el aprendizaje automático. Consiste en buscar los puntos (o elementos) más cercanos a un punto de consulta dentro de un espacio de datos.

Dado un conjunto de datos D de n puntos en un espacio métrico $\mathbb{R}^d$, y un punto de consulta q, el objetivo de **Nearest Neighbor Search (NNS)** es encontrar el punto en D que sea más cercano a q, según una métrica de distancia predefinida (por ejemplo, distancia Euclidiana, Manhattan, o coseno).

Formalmente:

- Sea $D = \{x_1, x_2, ..., x_n\}$ el conjunto de puntos (o vectores de características) en $\mathbb{R}^d$.
- Para un punto de consulta q, queremos encontrar el punto $x_i \in D$ tal que:

$$x_i = \arg\min_{x \in D} \text{distancia}(q, x)$$

donde la **distancia** puede ser cualquier métrica de similitud o distancia (Euclidiana, coseno, etc.).

Nearest neighbor search

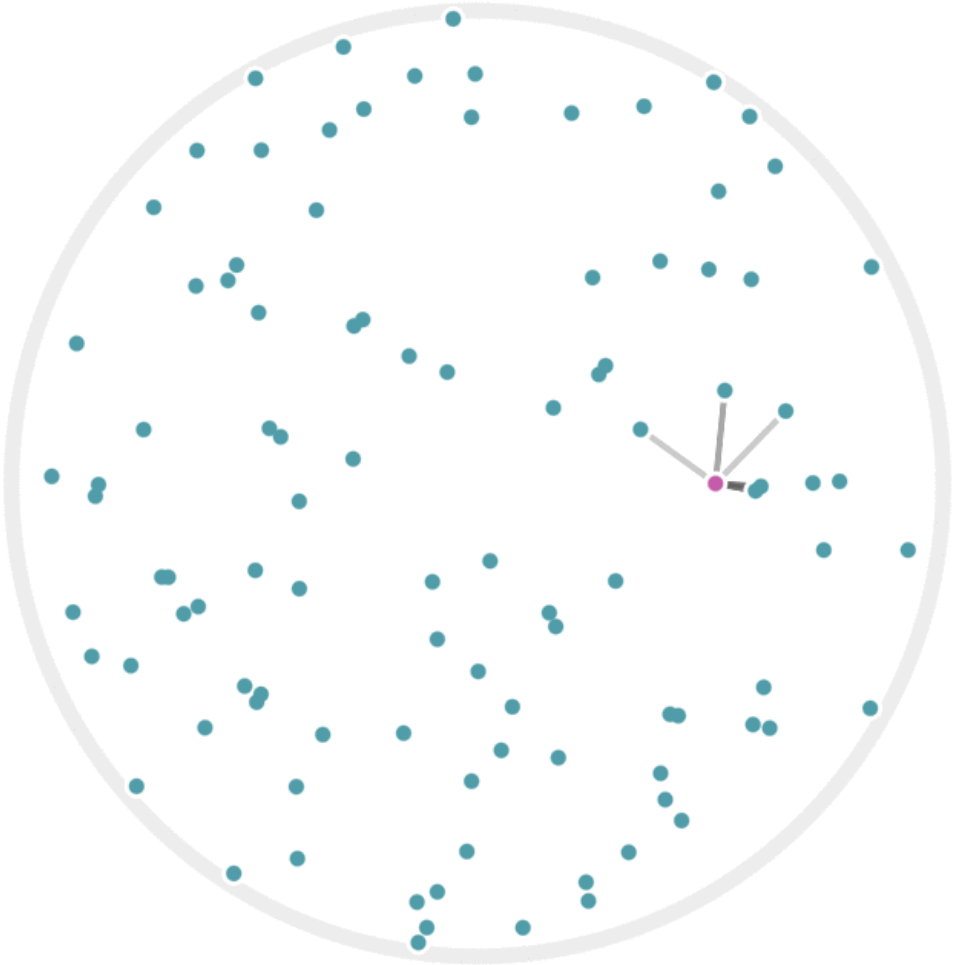
UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

- **Maldición de la dimensionalidad** : A medida que aumenta la dimensionalidad del espacio, la distancia entre los puntos tiende a volverse casi uniforme, lo que hace que las distancias euclidianas sean menos útiles para distinguir vecinos cercanos de los distantes.

- En un espacio de alta dimensión, la búsqueda de vecinos más cercanos utilizando métodos exactos, como una búsqueda de fuerza bruta (que compara el punto de consulta con cada punto en el conjunto de datos), se vuelve extremadamente lenta. El costo es proporcional al tamaño del conjunto de datos y la dimensionalidad, es decir, O(n * d), donde n es el número de puntos y d la dimensión.

Fundamentos algoritmo LSH

¿Qué es **LSH?**



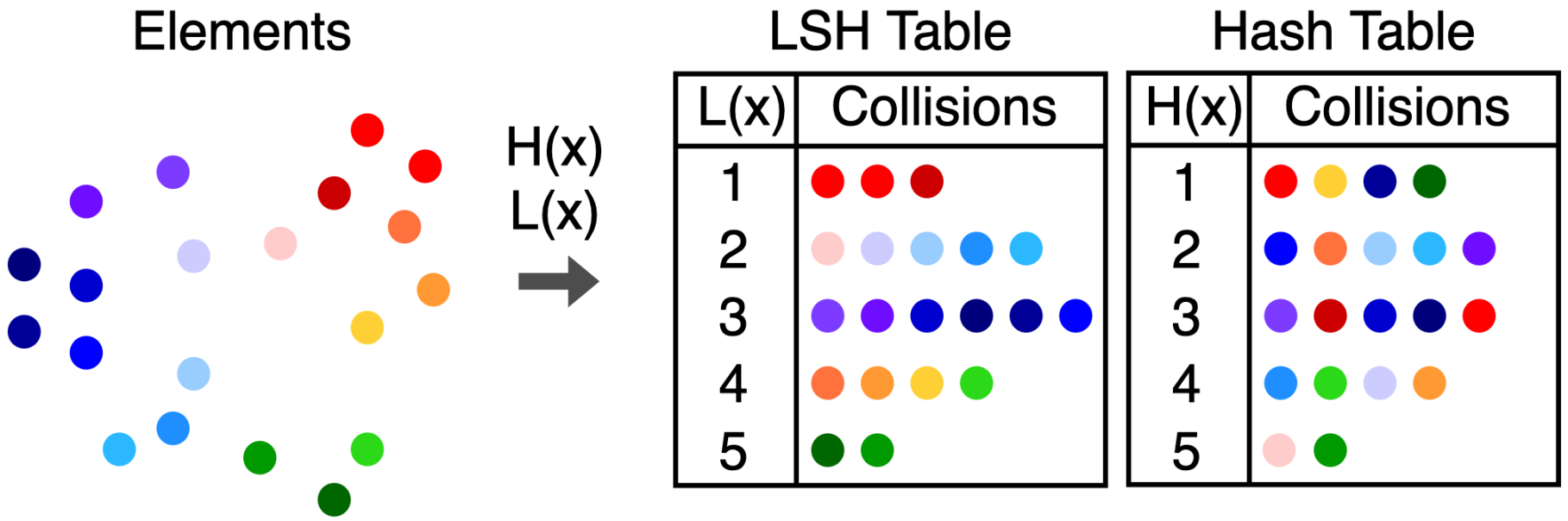
El hashing sensible a la localidad (LSH) es un conjunto de técnicas que aceleran drásticamente la búsqueda de vecinos o la detección de duplicaciones cercanas en los datos.

Estas técnicas se pueden utilizar, por ejemplo:

- filtrar duplicados de páginas web
- para realizar búsquedas en tiempo casi constante de puntos cercanos de un conjunto de datos geoespaciales.

https://randorithms.com/2019/09/19/Visual-LSH.html

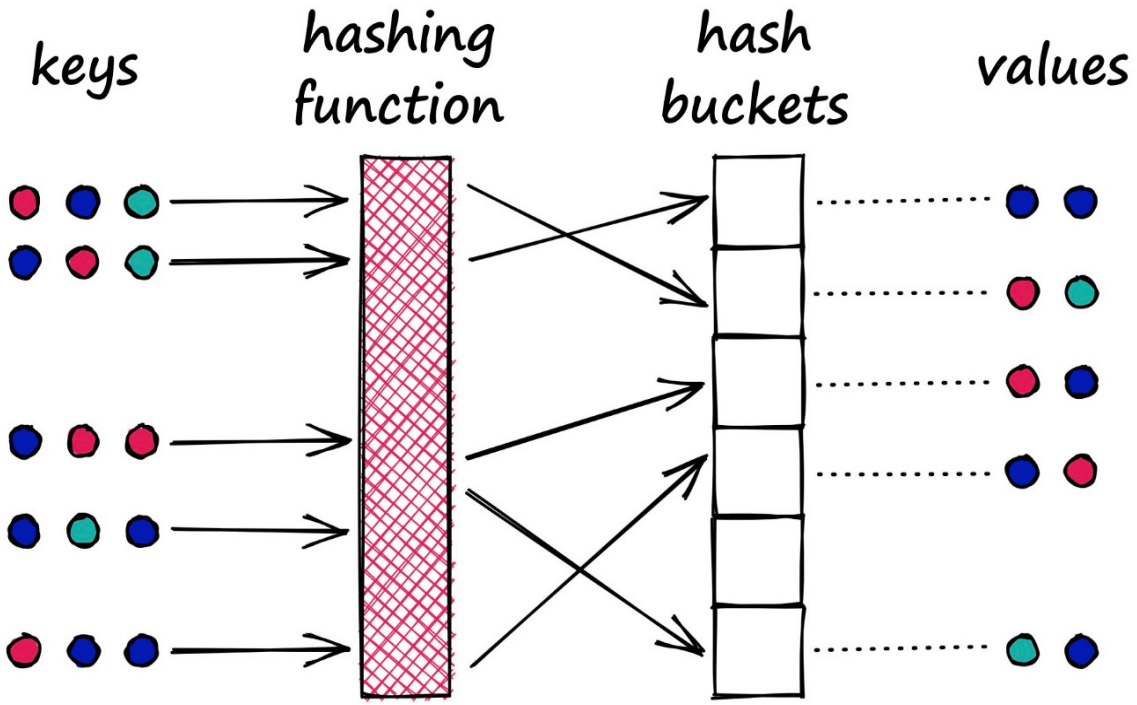
¿Qué es **LSH?**

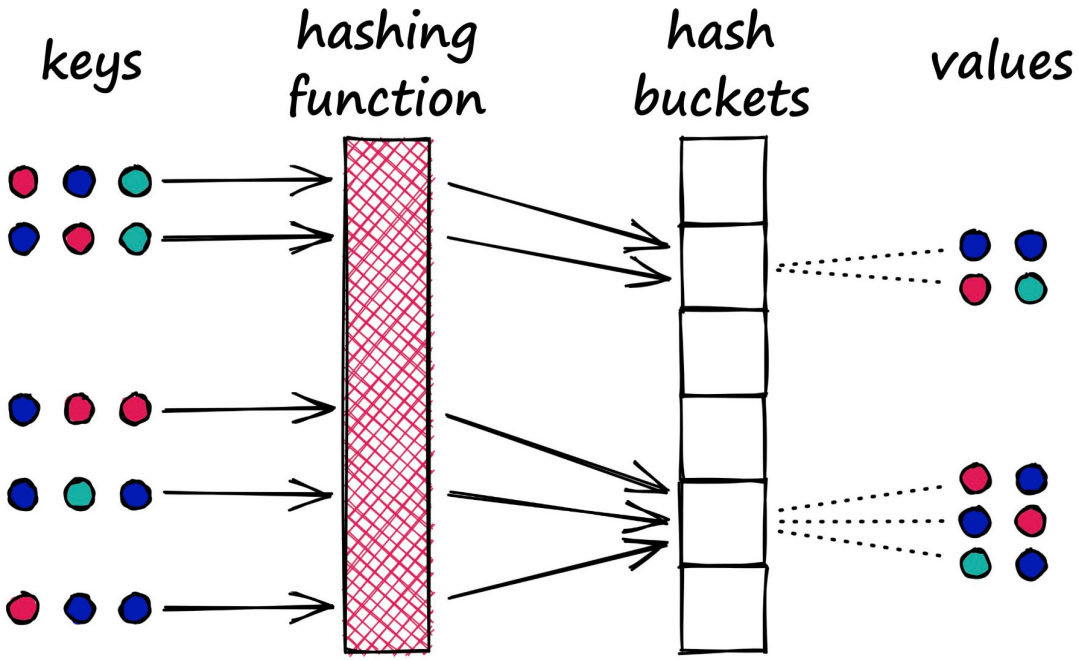


las funciones LSH intentan agrupar elementos similares en contenedores hash.

https://randorithms.com/2019/09/19/Visual-LSH.html

¿Qué es **LSH?**





https://randorithms.com/2019/09/19/Visual-LSH.html

¿Qué es **LSH?**

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Ventajas

- **Reducción del tiempo de búsqueda**: LSH reduce el tiempo necesario para encontrar vecinos aproximados en espacios de alta dimensionalidad
- LSH es adecuado para grandes bases de datos porque se puede escalar fácilmente en términos de la cantidad de datos y la dimensionalidad.
- LSH se puede ajustar para diversas métricas de distancia (como la distancia euclidiana, coseno, Manhattan), siempre que existan funciones hash sensibles a la localidad adecuadas para dichas métricas

Desventajas

- **Resultado aproximado**: Aunque LSH es eficiente, solo garantiza un resultado aproximado y no necesariamente el vecino más cercano exacto.
- **Alta memoria para grandes datasets**: A medida que aumenta el número de tablas hash y funciones hash, la cantidad de memoria necesaria para almacenar estas estructuras puede volverse muy grande.
- **Colisiones falsas**: Aunque LSH se basa en el principio de que puntos cercanos tienen mayor probabilidad de colisionar en el mismo bucket, puede haber colisiones falsas, donde puntos que no son cercanos terminan en el mismo cubo hash.

https://randorithms.com/2019/09/19/Visual-LSH.html

Probabilidades de Colisión de LSH

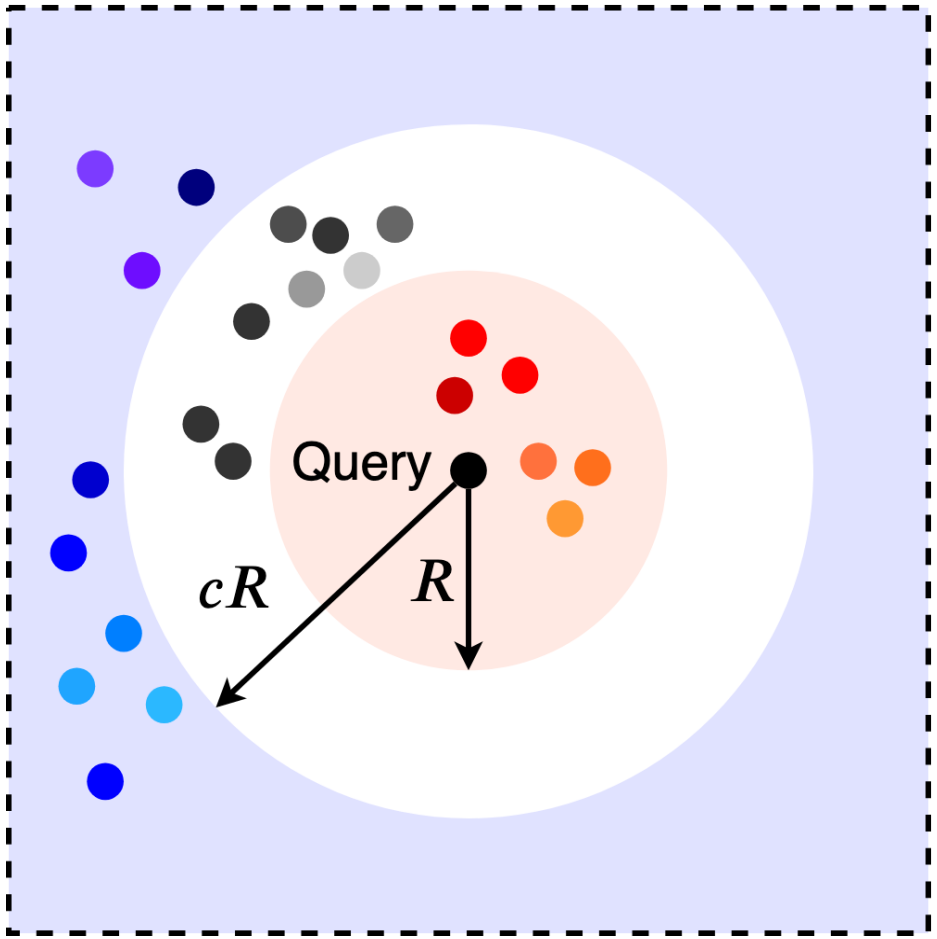
Una colisión hash ocurre cuando dos objetos X y Y tienen el mismo valor hash.

En LSH la probabilidad de que los objetos colisionen (denominada probabilidad de colisión) depende de cuán similares sean los objetos.

Definition 1: Locality Sensitive Hash Family

We say that a hash family $\mathcal{H}$ is (R, cR, p_1, p_2)-sensitive with respect to a distance function $d(x, y)$ if for any $h \in \mathcal{H}$ we have that

- If $d(x, y) \leq R$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \geq p_1$
- If $d(x, y) \geq cR$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \leq p_2$



https://randorithms.com/2019/09/19/Visual-LSH.html

Familias LSH según la métrica de distancia

Muestreo de bits LSH

El muestreo de bits es una de las funciones LSH más simples y económicas. Está asociada con la **distancia de Hamming.**

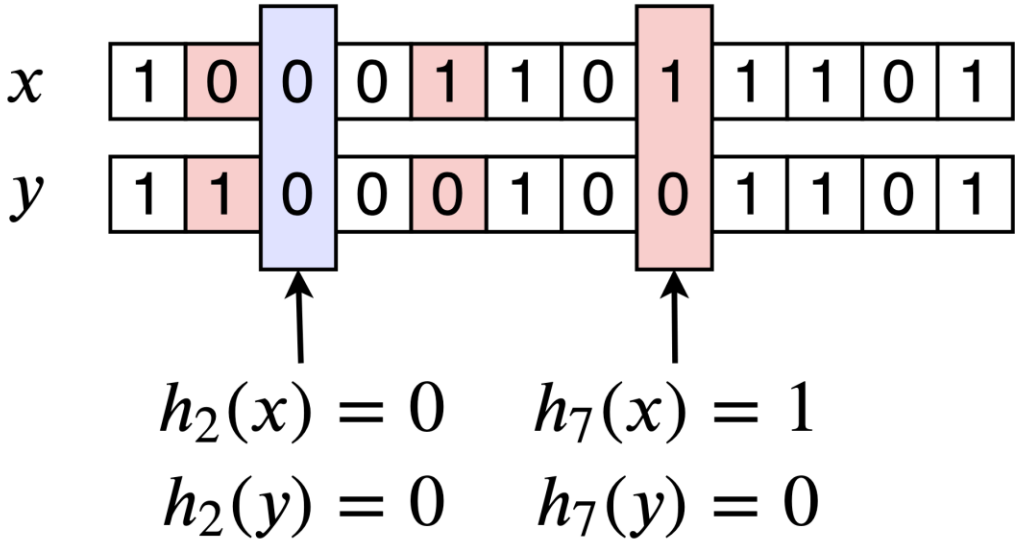
La distancia de Hamming es el número de bits en que difieren entre dos vectores

$$h_i(\mathbf{x}) = x_i \in \{0, 1\}$$

We choose a random index i between 0 and $n-1$ and have the LSH function output the bit x_i. To find the collision probability, observe that the number of indices for which $x_i = y_i$ is $n - d(\mathbf{x}, \mathbf{y})$. The collision probability is simply the chance that we picked one the indices where $x_i = y_i$.

$$p(\mathbf{x}, \mathbf{y}) = 1 - \frac{d(\mathbf{x}, \mathbf{y})}{n}$$

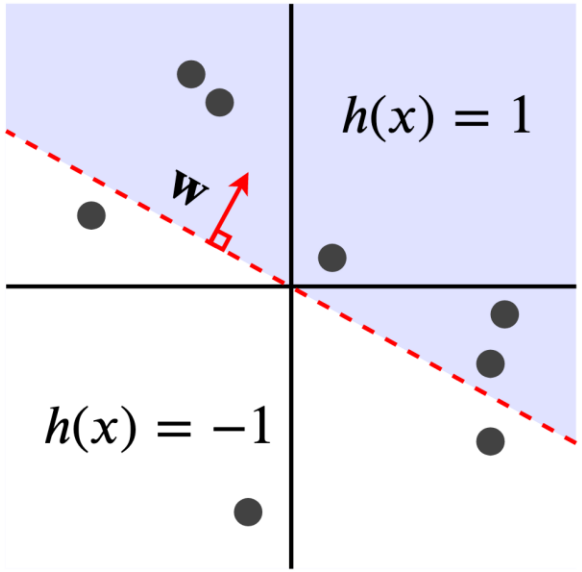
To bit sample, randomly choose an index
This is sensitive to Hamming distance



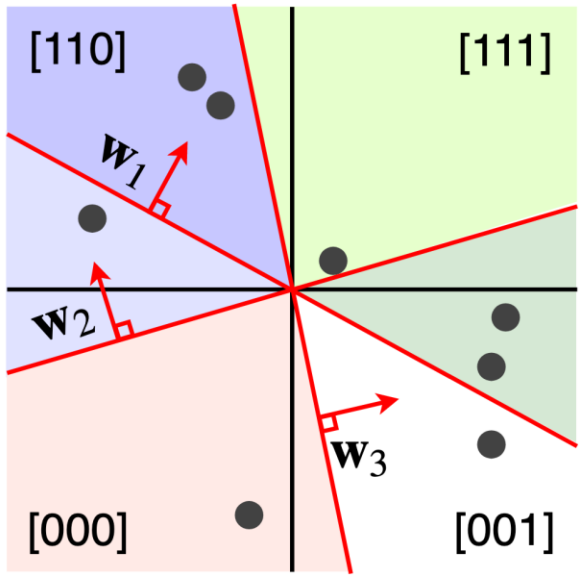
https://randorithms.com/2019/09/19/Visual-LSH.html

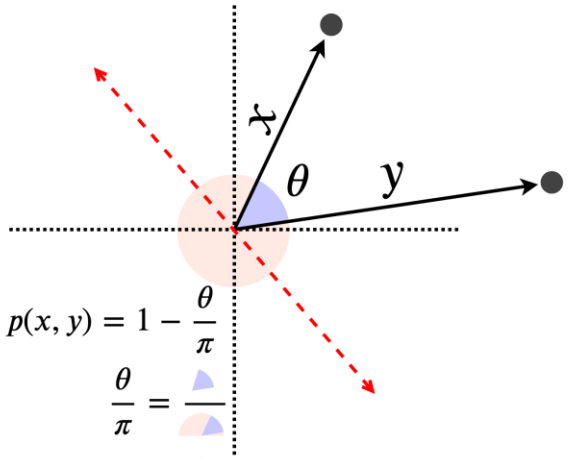
Proyecciones Aleatorias con signo

1 SRP



3 SRP





Las proyecciones aleatorias con signo (SRP) también generan valores binarios, pero SRP es sensible a la **distancia angular.**

Para construir una función hash SRP para un vector **x**, generamos un vector Gaussiano distribuido **w** con el mismo tamaño de x

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

$$p(\mathbf{x}, \mathbf{y}) = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi}$$

https://randorithms.com/2019/09/19/Visual-LSH.html

Proyecciones Aleatorias con signo

Supongamos los siguientes puntos:

- $p_1 = (1, 2)$
- $p_2 = (2, 3)$
- $p_3 = (3, 1)$
- $p_4 = (10, 10)$
- $p_5 = (9, 8)$

Proyecciones aleatorias

1. $r_1 = (1, 1)$
2. $r_2 = (1, -1)$
3. $r_3 = (-1, 1)$

Proyección 1: $r_1 = (1, 1)$

Punto	Proyección	Hash
p_1	$1 + 2 = 3$	+1
p_2	$2 + 3 = 5$	+1
p_3	$3 + 1 = 4$	+1
p_4	$10 + 10 = 20$	+1
p_5	$9 + 8 = 17$	+1

Proyección 2: $r_2 = (1, -1)$

Punto	Proyección	Hash
p_1	$1 - 2 = -1$	-1
p_2	$2 - 3 = -1$	-1
p_3	$3 - 1 = 2$	+1
p_4	$10 - 10 = 0$	+1
p_5	$9 - 8 = 1$	+1

https://randorithms.com/2019/09/19/Visual-LSH.html

Proyecciones Aleatorias con signo

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Supongamos los siguientes puntos:

- $p_1 = (1, 2)$
- $p_2 = (2, 3)$
- $p_3 = (3, 1)$
- $p_4 = (10, 10)$
- $p_5 = (9, 8)$

Proyecciones aleatorias

1. $r_1 = (1, 1)$
2. $r_2 = (1, -1)$
3. $r_3 = (-1, 1)$

Proyección 3: $r_3 = (-1, 1)$

Punto	Proyección	Hash
p_1	$-1 + 2 = 1$	+1
p_2	$-2 + 3 = 1$	+1
p_3	$-3 + 1 = -2$	-1
p_4	$-10 + 10 = 0$	+1
p_5	$-9 + 8 = -1$	-1

Tabla Hash Final

Hash	Puntos
$(+1, -1, +1)$	p_1, p_2
$(+1, -1, +1)$	p_3
$(+1, +1, -1)$	p_4
$(+1, -1, -1)$	p_5

Solo P1 y P2 Son similares de acuerdo a las proyecciones

https://randorithms.com/2019/09/19/Visual-LSH.html

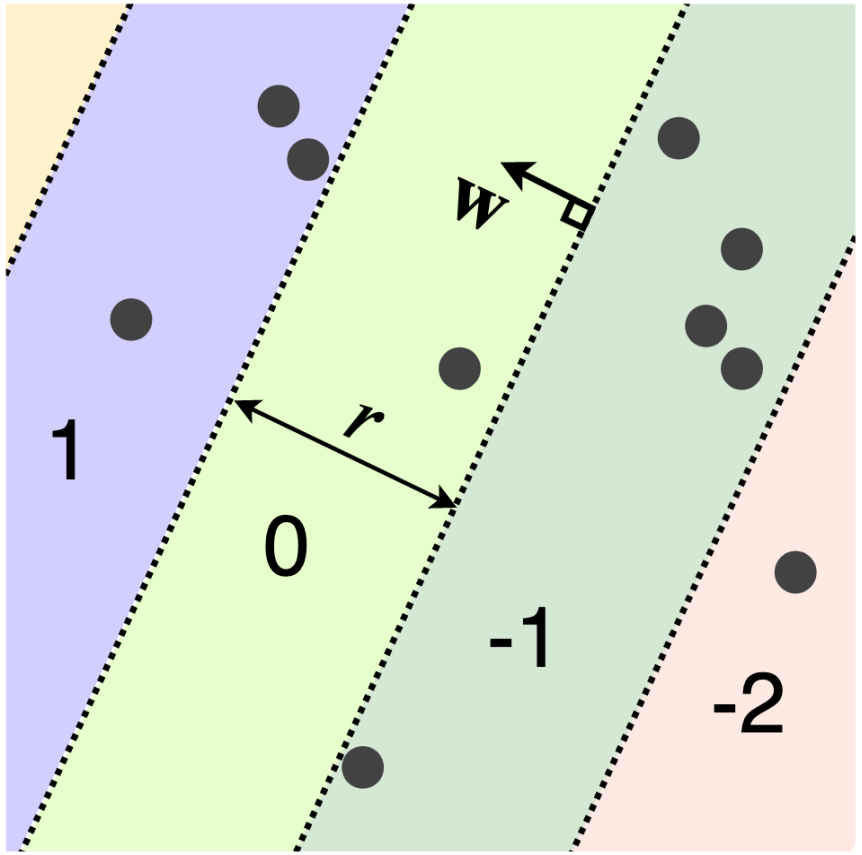
LSH euclidiano y manhattaniano

Las funciones LSH para las distancias euclidianas (L2) y de Manhattan (L1) también se basan en proyecciones aleatorias que cortan en pedazos.Cada pedazo se convierte en un contenedor hash. La función hash es idéntica a SRP, pero redondeamos en lugar de tomar el signo de la proyección.

Esto produce contenedores hash que se repiten en la dirección de **w** en lugar de un único límite de decisión como con SRP.

$$h(\mathbf{x}) = \left\lfloor \frac{\mathbf{e}\mathbf{l}^\top \mathbf{x} + b}{a} \right\rfloor$$

A diferencia de las funciones LSH anteriores, esta función hash se asigna a todo el **conjunto de números enteros**



https://randorithms.com/2019/09/19/Visual-LSH.html

LSH euclidiano

Supongamos que tenemos los siguientes puntos en un espacio bidimensional:

- $p_1 = (1, 1)$
- $p_2 = (2, 2)$
- $p_3 = (4, 4)$
- $p_4 = (10, 10)$

1. Proyección aleatoria con un vector $r_1 = (0.5, 0.5)$
2. Proyección aleatoria con un vector $r_2 = (-0.5, 0.5)$

Calculamos el producto punto entre cada punto y el vector de proyección:

- $p_1 \cdot r_1 = 1 \times 0.5 + 1 \times 0.5 = 1$
- $p_2 \cdot r_1 = 2 \times 0.5 + 2 \times 0.5 = 2$
- $p_3 \cdot r_1 = 4 \times 0.5 + 4 \times 0.5 = 4$
- $p_4 \cdot r_1 = 10 \times 0.5 + 10 \times 0.5 = 10$

Proyección 2: $r_2 = (-0.5, 0.5)$

- $p_1 \cdot r_2 = 1 \times (-0.5) + 1 \times 0.5 = 0$
- $p_2 \cdot r_2 = 2 \times (-0.5) + 2 \times 0.5 = 0$
- $p_3 \cdot r_2 = 4 \times (-0.5) + 4 \times 0.5 = 0$
- $p_4 \cdot r_2 = 10 \times (-0.5) + 10 \times 0.5 = 0$

Hash	Puntos
(1, 0)	p_1
(2, 0)	p_2
(4, 0)	p_3
(10, 0)	p_4

LSH Euclidiano: Se basa en la distancia directa en línea recta entre los puntos. Es útil para espacios donde la noción de cercanía sigue la geometría euclidiana.

https://randorithms.com/2019/09/19/Visual-LSH.html

LSH manhattaniano

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Proyección 1

Para cada punto, calculamos la suma de las coordenadas:

- $p_1 = (1, 1) \Rightarrow 1 + 1 = 2$
- $p_2 = (2, 2) \Rightarrow 2 + 2 = 4$
- $p_3 = (4, 4) \Rightarrow 4 + 4 = 8$
- $p_4 = (10, 10) \Rightarrow 10 + 10 = 20$

Proyección 2

Calculamos la diferencia entre las coordenadas:

- $p_1 = (1, 1) \Rightarrow |1 - 1| = 0$
- $p_2 = (2, 2) \Rightarrow |2 - 2| = 0$
- $p_3 = (4, 4) \Rightarrow |4 - 4| = 0$
- $p_4 = (10, 10) \Rightarrow |10 - 10| = 0$

Paso 3: Tabla Hash (Manhattan)

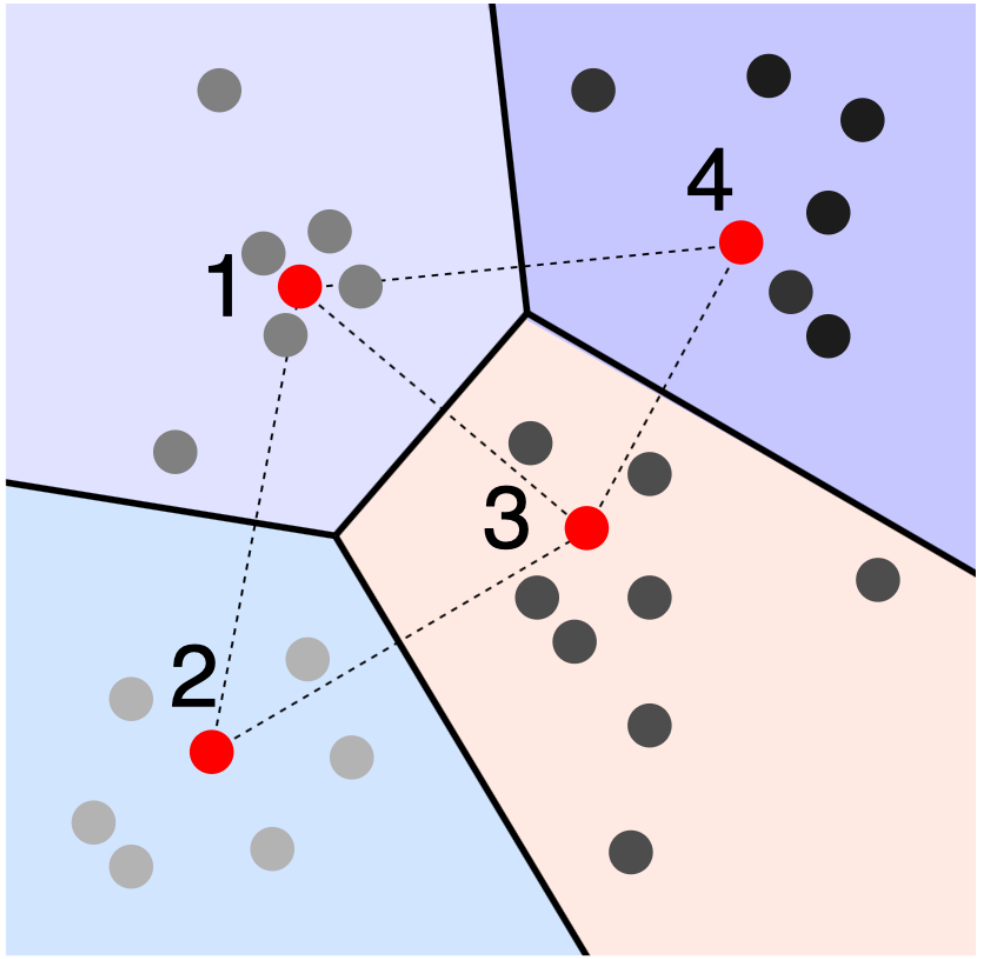
Como antes, agrupamos los puntos según sus hashes:

Hash	Puntos
(2, 0)	p_1
(4, 0)	p_2
(8, 0)	p_3
(20, 0)	p_4

LSH Manhattaniano: Mide las distancias en una "cuadrícula" (el número de movimientos necesarios para ir de un punto a otro). Es útil cuando los datos están organizados en una malla o cuando las restricciones físicas están presentes (como calles en una ciudad).

https://randorithms.com/2019/09/19/Visual-LSH.html

Agrupamiento LSH



La agrupación de LSH es un ejemplo de una función LSH aprendida o dependiente de los datos. La idea es encontrar un conjunto de centros de agrupamien que hacen un buen trabajo de aproximación del conjunto de datos, que se muestra en rojo. Estos centros se pueden encontrar utilizando la agrupación de k-medias, la agrupación convexa o cualquier otro método de agrupación. Una vez que tenemos los centros de agrupación, asignamos un valor entero a cada uno.

$$h(x) = \underset{1 \leq i \leq |\mathcal{C}|}{\arg\min}\, d(x, c_i)$$

https://randorithms.com/2019/09/19/Visual-LSH.html

En síntesis

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

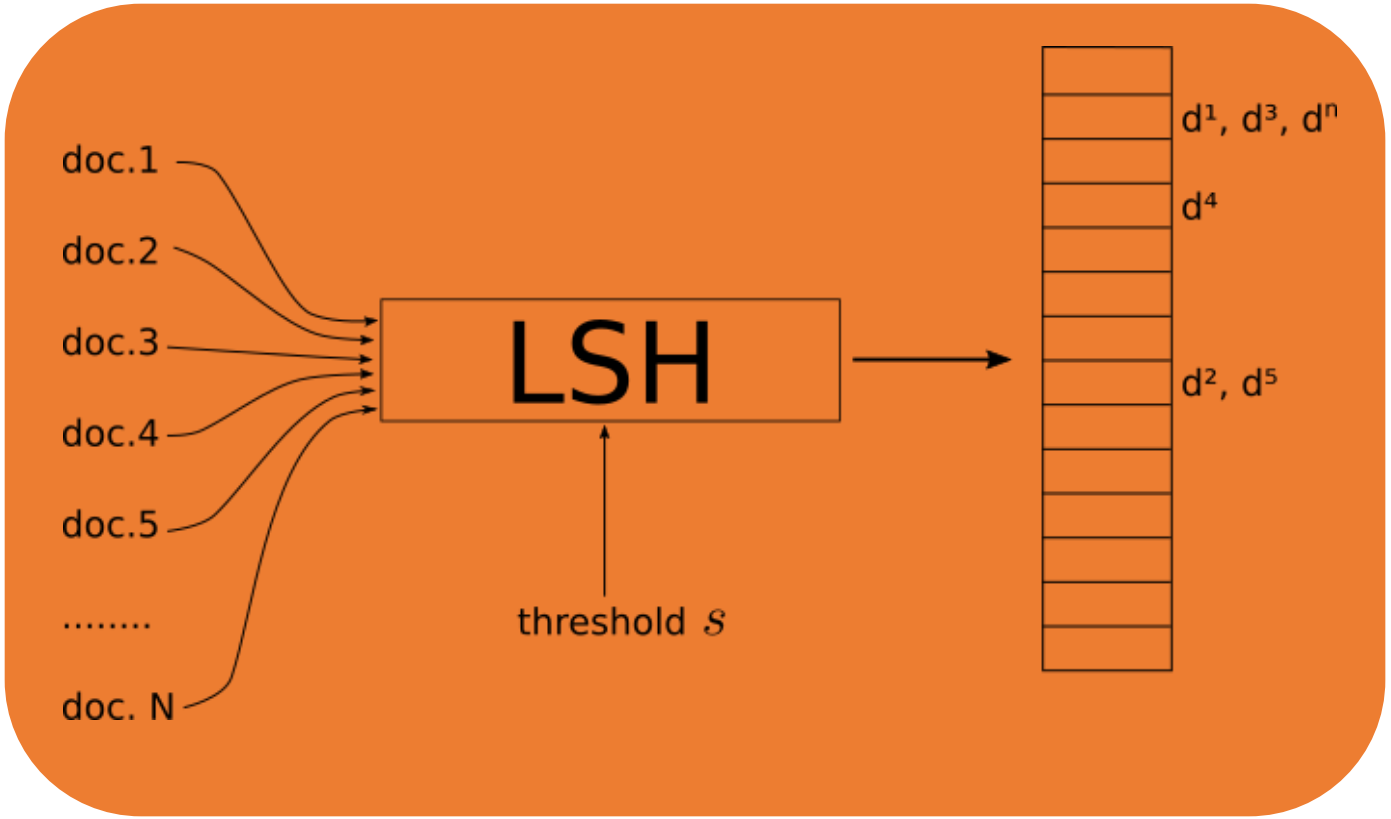
Métrica de Distancia	Familia LSH	Aplicaciones
Distancia Euclidiana (L2)	LSH con proyección aleatoria	Búsqueda de vecinos en alta dimensionalidad (imágenes, datos de sensores)
Distancia Manhattan (L1)	LSH ajustado para L1	Modelos de datos de texto, señales, datos de series temporales
Similitud de Coseno	LSH basado en hiperplanos	Recuperación de texto, procesamiento del lenguaje natural (NLP), recuperación de imágenes
Distancia Jaccard	MinHash	Detección de duplicados, análisis de conjuntos, búsqueda de documentos
Distancia Hamming	LSH para bits seleccionados	Comparación de secuencias binarias, redes de comunicación, corrección de errores

https://randorithms.com/2019/09/19/Visual-LSH.html

Aplicaciones de LSH

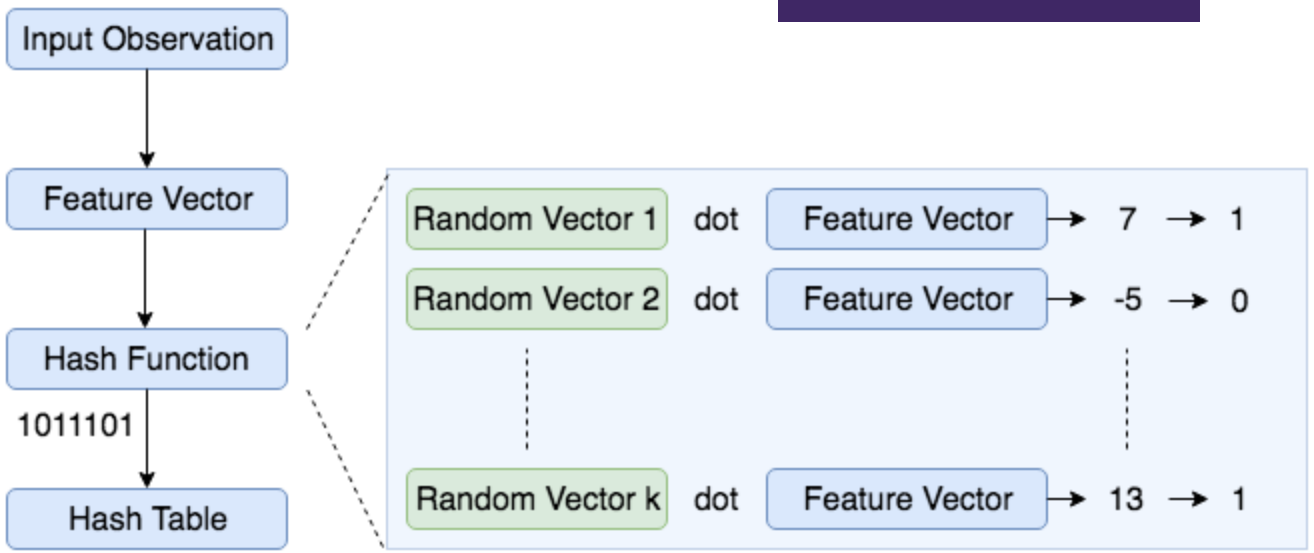
Detección de duplicados en documentos

- En grandes bases de datos de texto, LSH se aplica para identificar documentos similares o detectar plagio de manera eficiente.
- Permite comparar fragmentos de texto en lugar de realizar comparaciones entre documentos completos.



Procesamiento de Audio

Aplicaciones como Shazam utilizan LSH para encontrar canciones coincidentes en grandes bases de datos de música, identificando canciones con características acústicas similares a partir de vectores de tono de una base de datos musical



Sistemas de recomendación

Ayuda a recomendar productos, películas o canciones similares basándose en las preferencias de los usuarios.

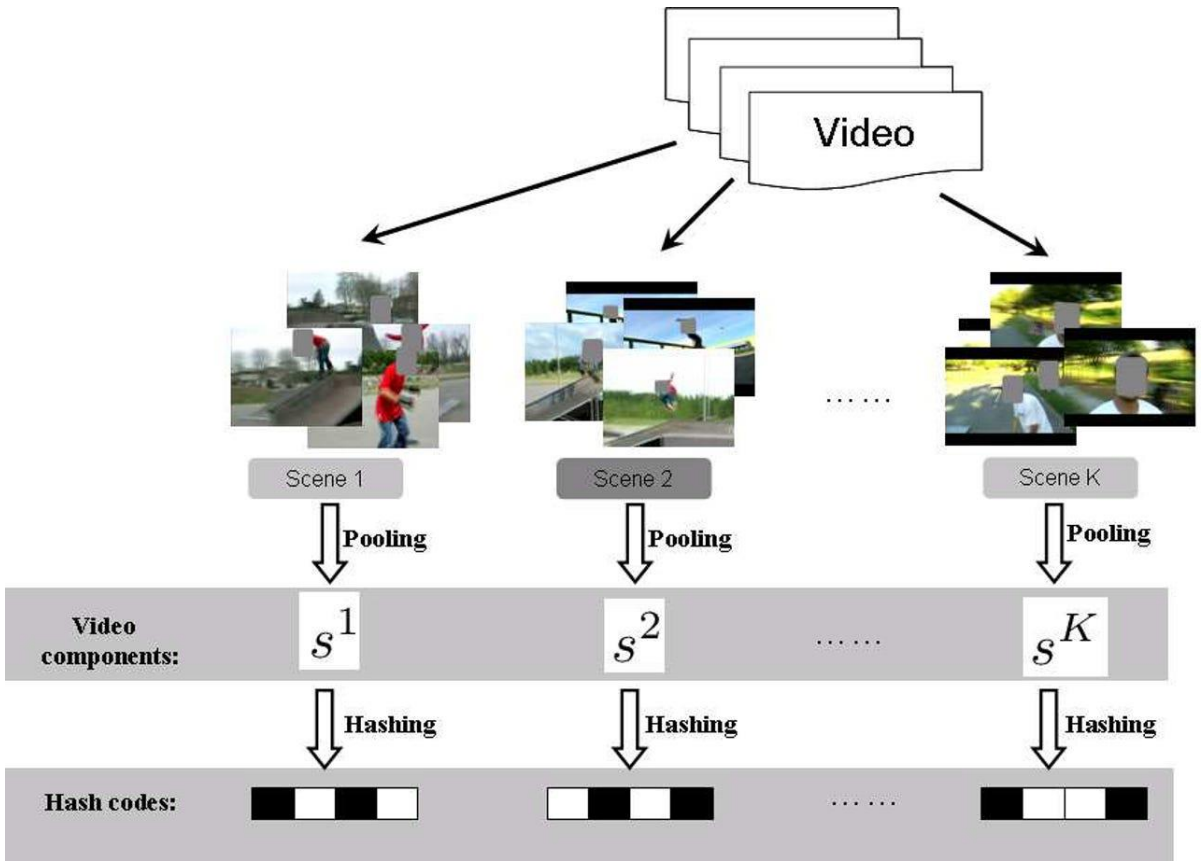
Permite encontrar patrones en datos dispersos (por ejemplo, preferencias de usuarios) para generar recomendaciones personalizadas.

Netflix: Recomienda películas o series similares según calificaciones o tiempo de visualización.

Spotify: Sugiere canciones con características acústicas similares (ritmo, tempo, instrumentos).

Amazon: Propone productos similares basados en atributos como categoría o precio.

Huella digital de vídeo

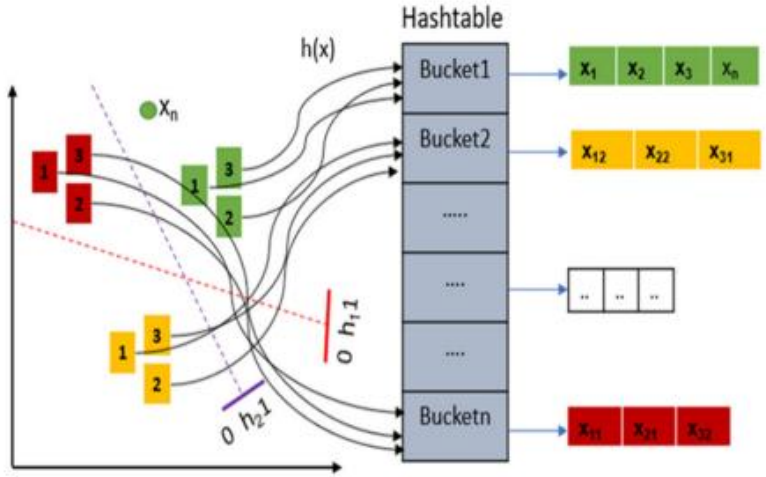


Permite comparar grandes volúmenes de contenido audiovisual de manera eficiente. LSH facilita la búsqueda rápida de vídeos similares al reducir el espacio de búsqueda y agrupar vídeos que comparten características visuales y acústicas.

- **Detección de copias**: Identificación de duplicados de vídeos en plataformas como YouTube.
- **Monitoreo de contenido**: Detección de fragmentos reutilizados en diferentes vídeos.
- **Búsqueda de vídeos**: Encontrar rápidamente vídeos similares a partir de su contenido, sin necesidad de comparar cada fotograma.

Huella digital de imágenes

- **Identificación de contenido duplicado**
- Detectar y eliminar imágenes que han sido copiadas sin autorización en sitios web o redes sociales.
- Permitir a los usuarios buscar imágenes similares.
- **Organización automática de archivos**
- Protección de la propiedad intelectual.
- **Reconocimiento facial**
- **Filtrado de contenido inapropiado**

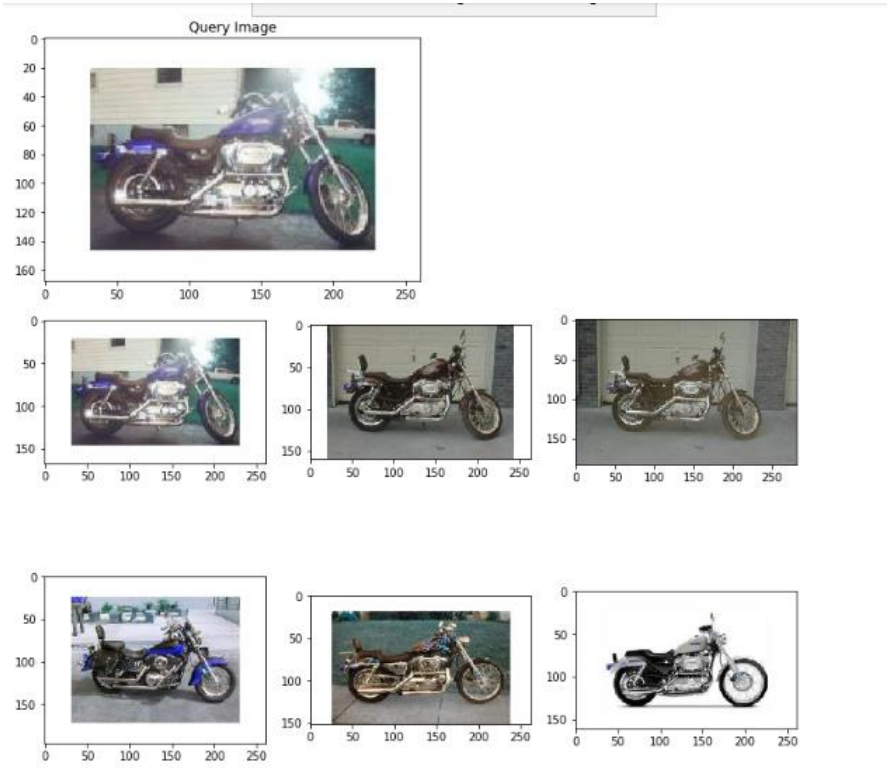


LSH en Imágenes

Toma de huellas dactilares de imágenes para la detección de duplicados casi exactos

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

- Contamos con múltiples bases de datos de imágenes y nuestro objetivo es identificar si existen imágenes muy similares dentro de una misma base de datos. La tarea es optimizar la recuperación de imágenes.
- La implementación de **Locality-Sensitive Hashing (LSH)** es una opción ideal para este propósito. LSH permite representar imágenes mediante códigos hash de baja dimensionalidad, facilitando así la comparación rápida y eficiente de imágenes.



Toma de huellas dactilares de imágenes para la detección de duplicados casi exactos

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

- El hash de imágenes es el proceso de examinar el contenido de una imagen y luego construir un valor que identifica de forma única una imagen en función de este contenido.
- Dada una imagen de entrada, aplicaremos una función hash y calcularemos un "hash de imagen" en función de la apariencia visual de la imagen. Las imágenes que son "similares" también deberían tener hashes que sean "similares".

Fundamento matemático LSH

LSH - IMAGENES

- El enfoque LSH que estamos explorando consiste en un proceso de cuatro pasos.

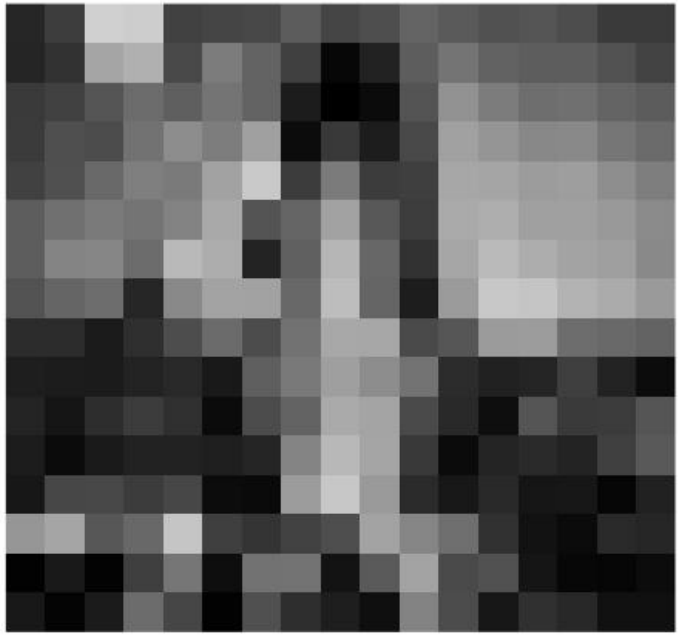
1. **Preprocesamiento de imágenes**
2. **Aplicación de LSH**
3. **Generación de pares candidatos**
4. **Comparación de Similitud**

Preprocesamiento de imágenes

La idea es comprimir las imágenes en firmas más pequeñas, preservando al mismo tiempo la similitud entre ambas.

(a) Original 5335x3557 image

(b) Resized to 17x16, colour



(c) Resized to 17x16, greyscale

dHash

Busca calcular la diferencia entre valores de intensidad de luz adyacentes horizontalmente.

$$H_{\text{fila}}(i) = \begin{cases} 1 & \text{si } p(i, j+1) \geq p(i, j) \\ 0 & \text{si } p(i, j+1) < p(i, j) \end{cases}$$

Sea $p(i, j)$ el valor de intensidad de gris del píxel en la posición (i, j) de la imagen

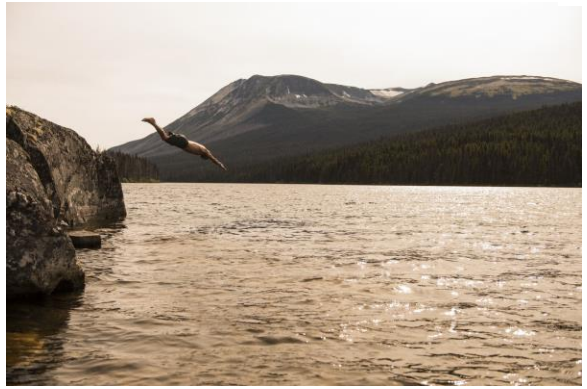


Imagen original

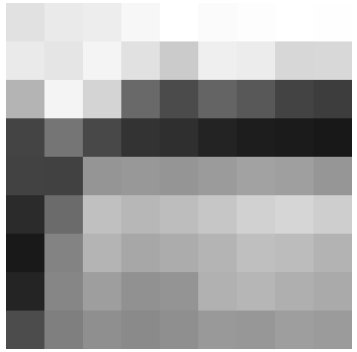
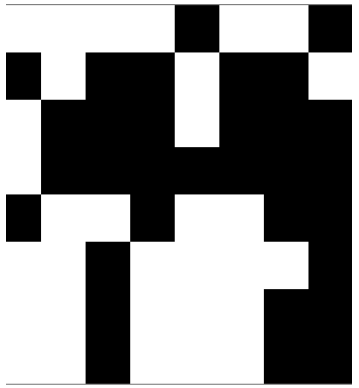


Imagen redimensionada y en escala de grises



Firma

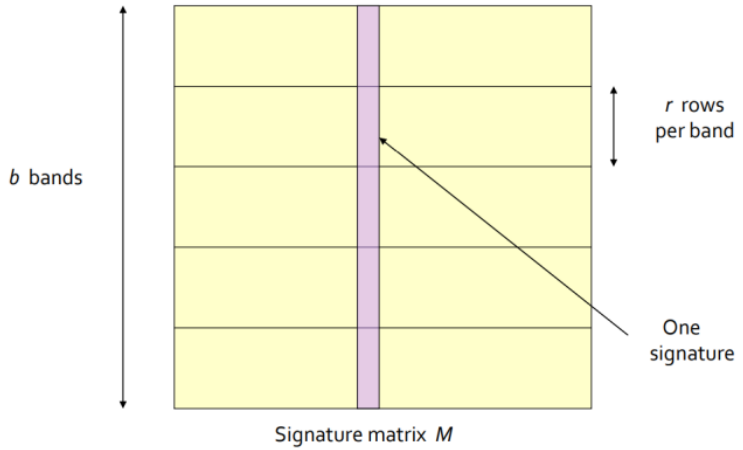
Aplicación de LSH

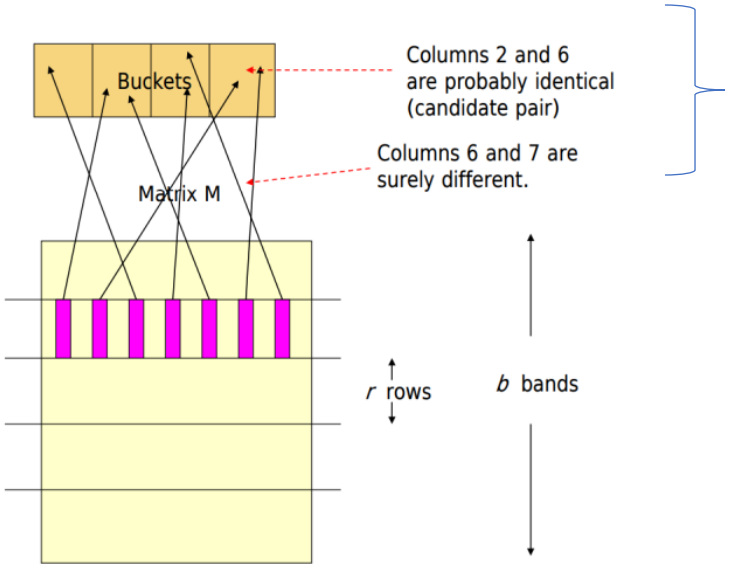
División en bandas

- Las firmas se dividen en bandas, donde cada banda tiene un número específico de filas. Para cada imagen, se toma cada banda y se convierte a bytes.

Construcción de "Hash Buckets"

- Se almacenan las imágenes en "buckets" basados en el hash de cada banda. Las imágenes con el mismo hash se agrupan juntas.





Pares candidatos

Distancia Hamming

La **distancia de Hamming** es una métrica que se utiliza para medir la diferencia entre dos cadenas de igual longitud. Específicamente, cuenta el número de posiciones en las que los elementos correspondientes son diferentes.

$$d_H(s_1, s_2) = \sum_{i=1}^{n} \delta(s_1[i], s_2[i])$$

n es la longitud de las cadenas.

$\delta(a, b)$ es una función que devuelve 1 si $a \neq b$ y 0 si $a = b$.

Se compara la similitud con un umbral definido. Si es mayor que el umbral, el par se considera como imágenes casi duplicadas.

$$S = 1 - \frac{d_H}{k^2}$$

Donde:

- S es la similitud entre las dos imágenes (valor entre 0 y 1).
- d_H es el número de bits diferentes entre las dos firmas.
- k^2 es la longitud total de la firma (número total de bits en la firma).

¿ Cómo minimizar los falsos positivos?

FALSOS POSITIVOS

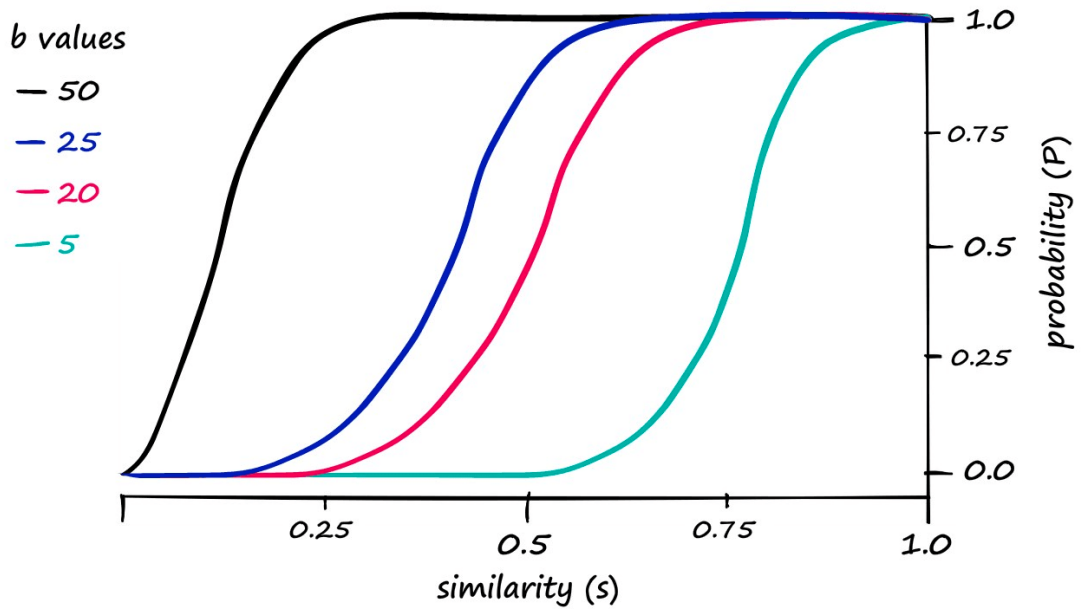
La sensibilidad de la detección de pares de candidatos en un sistema (LSH) puede influenciarse ajustando los parámetros b (número de bandas) y r (número de filas por banda).

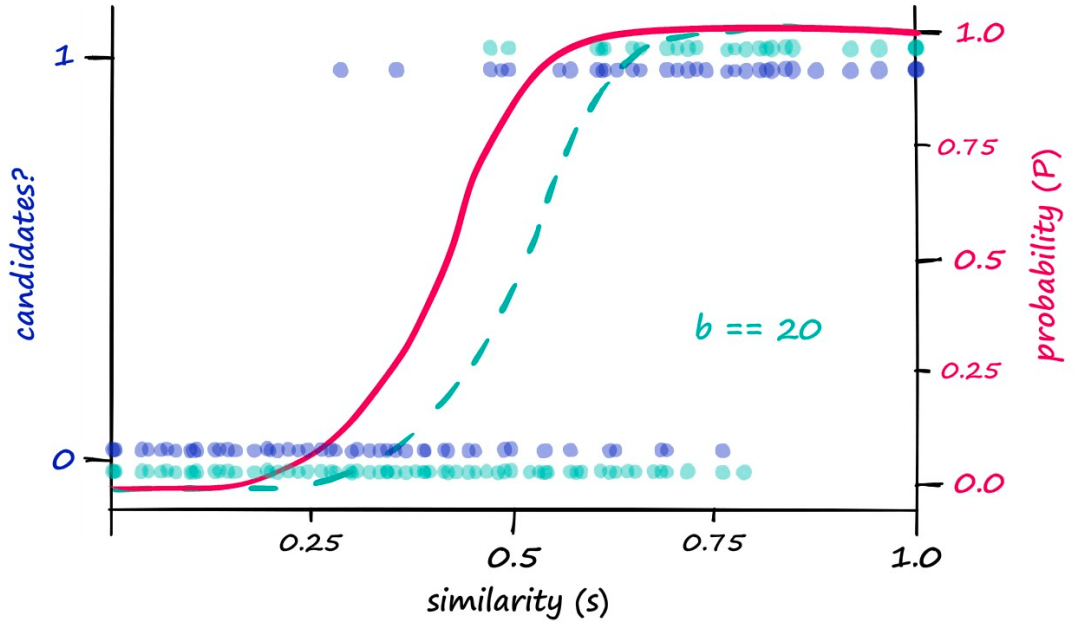
Si se aumenta el número de bandas, cada banda contendrá menos filas de la firma, lo que puede aumentar la probabilidad de que dos imágenes no similares se clasifiquen erróneamente como candidatos (falsos positivos).

$$P = 1 - (1 - s^r)^b$$

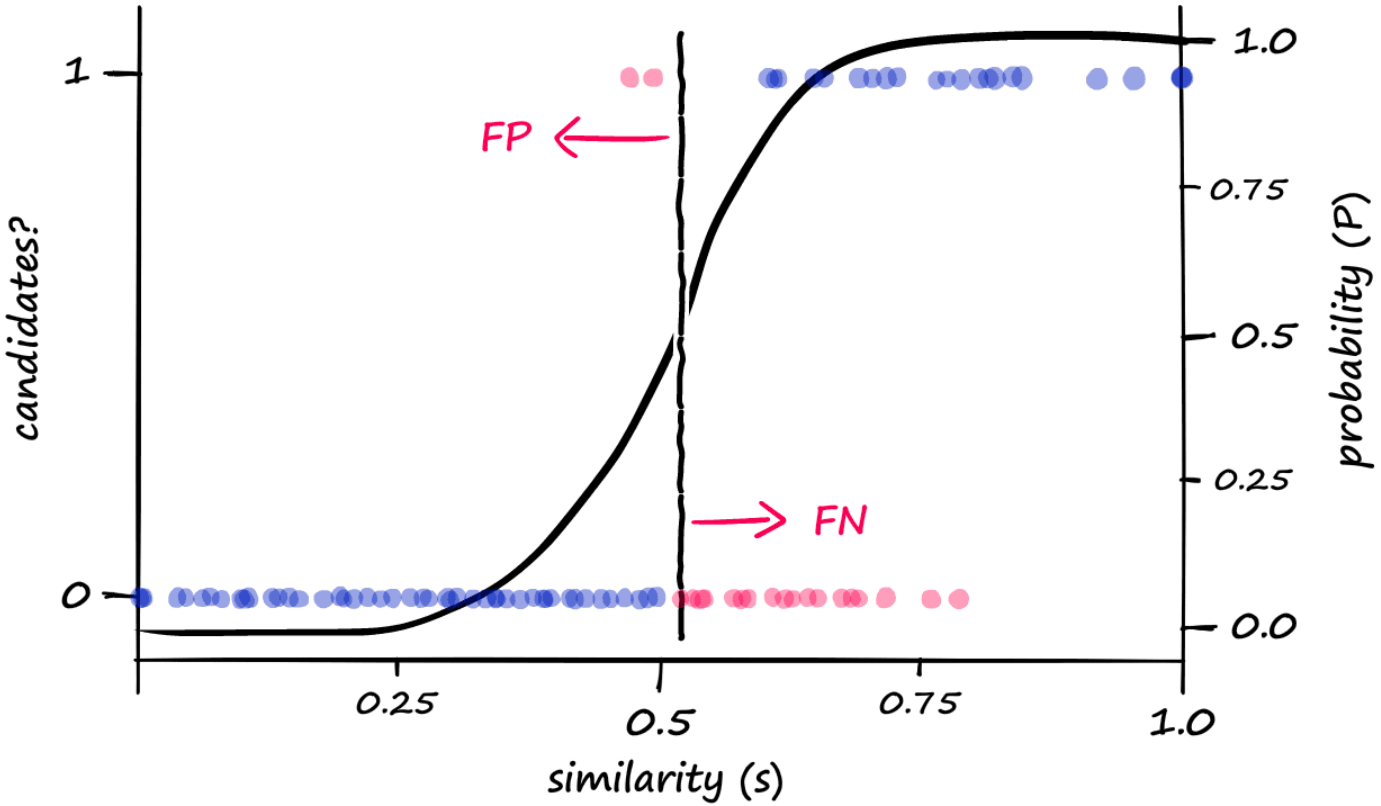
Probabilidad (P) de que un par sea identificado como par candidato dada una puntuación de similitud (s), número de bandas (b) y número de filas en cada banda (r).

FALSOS POSITIVOS





FALSOS POSITIVOS



Aumentar b (desplazarse hacia la izquierda) aumenta los FP.

Al mismo tiempo, disminuye los FN.

Aplicación LSH - Práctico

Datos

Datasets

Pardon our dust! We are in the process of transitioning our datasets to new hosting services.

As of July 2022, our current status is as follows:

- Some of our datasets are available here.
- Some of our datasets are listed below.
- All other datasets are available by:
  - (*Faster*) submitting an issue or a pull request here
  - (*Slower*) sending an email to Laure Deslisle (laure.delisle@caltech.edu), Rogério Guimarães (rogerio@caltech.edu), Suzanne Stathatos (sstathat@caltech.edu). Please include "Vision Lab Request" in the subject of your email.

Project Pages for Datasets

- CUB-200-2011
- Caltech Camera Traps
- Caltech 10k Web Faces
- Caltech Mouse Social Interaction Dataset 2021 (CalMS21)
- FlyTracker

Other Datasets

- Caltech 101

https://www.vision.caltech.edu/datasets/

Datos

Caltech 101

Li, Fei-Fei[1]; Andreeto, Marco[1]; Ranzato, Marc'Aurelio[1]; Perona, Pietro[1]

Show affiliations

Citation

Style APA

Li, F.-F., Andreeto, M., Ranzato, M., & Perona, P. (2022). Caltech 101 (1.0) [Data set]. CaltechDATA. https://doi.org/10.22002/D1.20086

Description

Pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc'Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels. We have carefully clicked outlines of each object in these pictures, these are included under the 'Annotations.tar'. There is also a MATLAB script to view the annotations, 'show_annotations.m'.

https://data.caltech.edu/records/mzrjq-6wc02

LSH – Medida de distancia coseno

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Keras-Reverse-Image-Search-Using-LSH

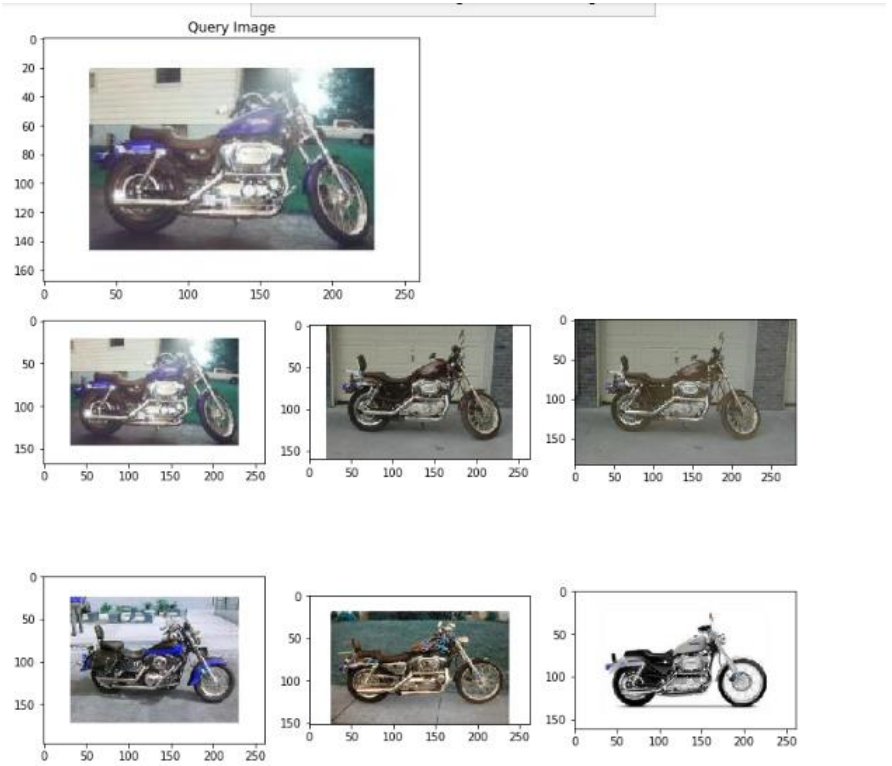
Keras Reverse Image Search using Locality Sensitive Hashing on Caltech 101 Dataset

For downloading the Dataset:

http://www.vision.caltech.edu/Image_Datasets/Caltech101/

Steps Followed:

1 Loading the data
2 Sorted images based on category.
2 Used xception and take the output from last dense layer.
3 Make a feature vector of all the training data.
4 For the test data make the feature vector and to check similarity used LSH and Cosine Similarity.



https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia coseno

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

```python
def result_vector_cosine(model, feature_vector, new_img):
    new_feature = model.predict(new_img)
    new_feature = np.array(new_feature).flatten()

    N_result = 6  # Número de imágenes similares a recuperar
    nbrs = NearestNeighbors(n_neighbors=N_result, metric="cosine").fit(feature_vector)
    distances, indices = nbrs.kneighbors([new_feature])

    # Convertir distancias de coseno a porcentaje de similitud
    similarity_percentages = 100 * (1 - distances)
    print(similarity_percentages)
    # Retornar los índices de las imágenes similares y los porcentajes de similitud
    return indices, similarity_percentages
```

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia coseno

```
N_result = 6  # Número de imágenes similares a recuperar
nbrs = NearestNeighbors(n_neighbors=N_result, metric="cosine").fit(feature_vector)
distances, indices = nbrs.kneighbors([new_feature])
```

- `N_result = 6` : Se define que se quieren encontrar las **6 imágenes más similares**.
- Se crea una instancia de `NearestNeighbors`, que es un algoritmo para encontrar los **vecinos más cercanos** en función de una métrica de distancia, en este caso, la **distancia coseno**.
- Se ajusta el modelo `NearestNeighbors` con el conjunto de vectores de características (`feature_vector`) de las imágenes ya calculadas.
- `kneighbors([new_feature])` : Se utiliza este método para encontrar las imágenes más cercanas a la nueva imagen (consulta). Este método retorna:
  - `distances` : Las distancias coseno de las 6 imágenes más cercanas.
  - `indices` : Los índices de esas imágenes dentro del conjunto de datos.

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia Hamming

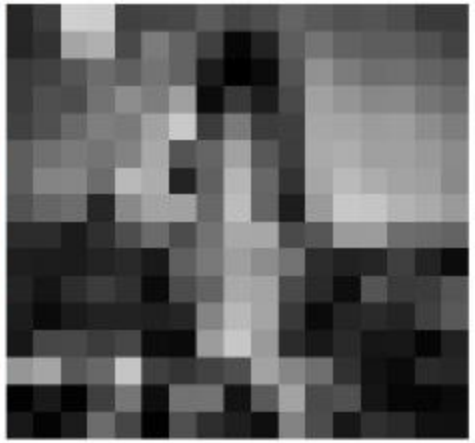
LSH for near-duplicate image detection

This proof-of-concept uses **Locality Sensitive Hashing** for near-duplicate image detection and was inspired by Adrian Rosebrock's article Fingerprinting Images for Near-Duplicate Detection. At the end of the article, the author proposes to use K-d trees or VP trees to achieve real near-duplicate detection in a scalable way. This page explains how to achieve that with Locality Sensitive Hashing instead.

> Note: Instructions on how to run the proof-of-concept code can be found at the bottom of this page.

(a) Original 5335x3557 image

(b) Resized to 17x16, colour



(c) Resized to 17x16, greyscale

(a) Original - 100%

(b) Waldo interference - 97.27%

(c) Bottom cropped - 77.73%

LSH – Medida de distancia Hamming

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

```python
def calculate_signature(image_file: str, hash_size: int) -> np.ndarray:
    """
    Calculate the dhash signature of a given file

    Args:
        image_file: the image (path as string) to calculate the signa
        hash_size: hash size to use, signatures will be of length has

    Returns:
        Image signature as Numpy n-dimensional array or None if the f
    """
    pil_image = Image.open(image_file).convert("L").resize(
                        (hash_size+1, hash_size),
                        Image.Resampling.LANCZOS)
    dhash = imagehash.dhash(pil_image, hash_size)
    signature = dhash.hash.flatten()
    pil_image.close()
    return signature
```

1. Convierte la imagen a escala de grises para reducir la complejidad de las comparaciones.
2. Redimensiona la imagen a una matriz pequeña de píxeles.
3. Compara la luminancia de píxeles adyacentes:
   - Si un píxel es más claro que el siguiente, se almacena un 1.
   - Si es más oscuro, se almacena un 0.
4. El resultado es una matriz binaria que representa la estructura visual de la imagen.
5. Esta matriz binaria se aplana y se convierte en un vector de bits (el **hash**).

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia Hamming

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

¿ Qué es el DHash?

El **diferential hash (dHash)** se basa en calcular las diferencias en los valores de luminancia de píxeles adyacentes. Es un tipo de **perceptual hashing** que se utiliza para medir la **similitud visual** entre imágenes. A diferencia de un hash criptográfico que está diseñado para detectar cambios pequeños en los datos, el dHash está diseñado para que imágenes visualmente similares tengan hashes cercanos, incluso si no son idénticas en cuanto a datos (por ejemplo, si una imagen es redimensionada o ligeramente modificada).

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia Hamming

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Asignación a cubos de LSH

```
for i in range(bands):
    signature_band = signature[i*rows:(i+1)*rows]
    signature_band_bytes = signature_band.tobytes()
    if signature_band_bytes not in hash_buckets_list[i]:
        hash_buckets_list[i][signature_band_bytes] = list()
    hash_buckets_list[i][signature_band_bytes].append(fh)
```

- La firma de hash de cada imagen se divide en **bandas**. Cada banda representa un subconjunto de la firma total.

- La banda se convierte en una secuencia de bytes (tobytes()), que se utiliza como clave en los cubos de LSH.

- Si la clave (banda en bytes) no existe en el cubo correspondiente, se crea una nueva entrada con una lista vacía.

- Luego, el archivo de imagen se agrega a la lista correspondiente a su clave de banda.

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia Hamming

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Construcción de pares de candidatos

```
candidate_pairs = set()
for hash_buckets in hash_buckets_list:
    for hash_bucket in hash_buckets.values():
        if len(hash_bucket) > 1:
            hash_bucket = sorted(hash_bucket)
            for i in range(len(hash_bucket)):
                for j in range(i+1, len(hash_bucket)):
                    candidate_pairs.add(
                        tuple([hash_bucket[i], hash_bucket[j]])
                    )
```

- Una vez que las imágenes están asignadas a sus cubos de LSH, se identifican los posibles **pares de candidatos**: imágenes que comparten la misma clave de banda y, por lo tanto, pueden ser similares.

- Se construyen todos los pares de candidatos que caen en el mismo cubo y se agregan a un conjunto (`set`) para evitar duplicados.

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

LSH – Medida de distancia Hamming

Comparación de pares de candidatos

```
for cpa, cpb in candidate_pairs:
    hd = sum(np.bitwise_xor(np.unpackbits(signatures[cpa]), np.unpackbits(signatures[c
    similarity = (hash_size**2 - hd) / hash_size**2
    if similarity > threshold:
        near_duplicates.append((cpa, cpb, similarity))
```

- Los pares de candidatos se comparan calculando su distancia de Hamming (HD). Esto se hace aplicando una operación XOR bit a bit (np.bitwise_xor()) entre las firmas de las dos imágenes.
- La distancia de Hamming es el número de posiciones en las que los bits de las dos firmas difieren.

LSH – Medida de distancia Hamming

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Ordenación de duplicados por similitud

```
near_duplicates.sort(key=lambda x:x[2], reverse=True)
return near_duplicates
```

Este algoritmo utiliza Locality Sensitive Hashing (LSH) para **agrupar firmas de imágenes similares** en **cubetas** de hash. La clave está en dividir las firmas de hash en bandas, donde cada banda se convierte en una clave para asignar las imágenes a cubos. Las imágenes que caen en el mismo cubo se consideran candidatas a ser comparadas más a fondo.

https://github.com/TanyaChutani/Keras-Reverse-Image-Search-Using-LSH-and-Cosine-Similarity

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Gracias

ingenieria.udea.edu.co
Fecha y ciudad

Una Facultad abierta
y transformadora