Procesamiento de Lenguaje Natural

Traducción automática y búsqueda

Dra. Helena Gómez Adorno

helena.gomez@iimas.unam.mx

Dra. Gemma Bel

gbele@iingen.unam.mx

Correo del curso:

pln.cienciadedatos@gmail.com

Asistente:

Luis Ramon Casillas

Contenido

Traducción automática

“hola” → “bonjour”

Búsqueda de documentos

“Puedo obtener” un reembolso?”

“¿Cuál es tu política de devoluciones?”

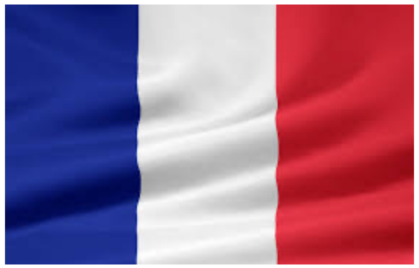
¿Puedo recuperar mi dinero?

Contenido

- Transformación de vectores
- K-vecinos mas cercanos
- Tablas Hash
- Dividir espacio de vectores en regiones
- Hash sensible a la localidad
- Vecinos mas cercanos aproximados

Revisión de traducciones

$\xrightarrow{?}$



Español
- [-2, -4, -8]
- …
- [2, 1, 0]

Francés
- [3, 2, 7]
- …
- [9, 33, 12]

Revisión de traducciones

?

Gato

Español
[-2, -4, -8]
…
[2, 1, 0]

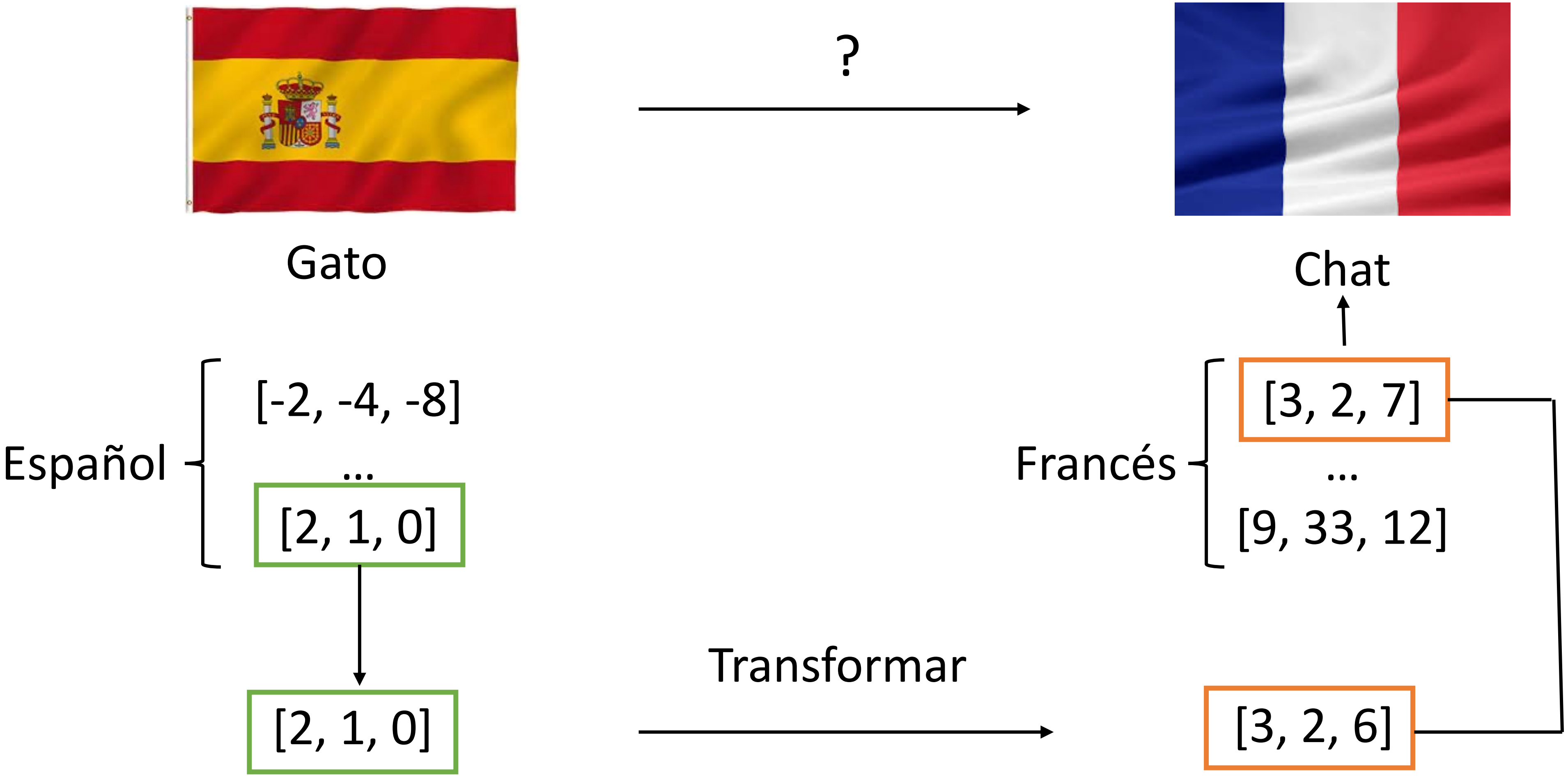
Francés
[3, 2, 7]
…
[9, 33, 12]

[2, 1, 0]

Transformar

[3, 2, 6]

Revisión de traducciones



Revisión de traducciones

?

Gato

Chat

Español
- [-2, -4, -8]
- …
- [2, 1, 0]

Francés
- [3, 2, 7]
- …
- [9, 33, 12]

[2, 1, 0]

Transformar

Con una matriz

[3, 2, 6]

Transformación de vectores

```
R = np.array([[2,0],
              [0,-2]])

x = np.array([[1,1]])

np.dot(x,R)
```

```
array([[2,-2]])
```

Alineación de vectores de palabras

$$XR \approx Y$$

[vector “gato”]
[… vector]

X

[vector “chat”]
[… vector]

Y

Alineación de vectores de palabras

$$XR \approx Y$$

$$\underset{\mathbf{X}}{\begin{pmatrix} \text{[vector “gato”]} \\ \text{[... vector]} \\ \text{[vector “zebra”]} \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} \text{[vector “chat”]} \\ \text{[... vector]} \\ \text{[vector “zébresse”]} \end{pmatrix}}$$

Alineación de vectores de palabras

$$XR \approx Y$$

$$\underbrace{\begin{pmatrix} \text{[vector “gato”]} \\ \text{[... vector]} \\ \text{[vector “zebra”]} \end{pmatrix}}_{\mathbf{X}} \qquad \underbrace{\begin{pmatrix} \text{[vector “chat”]} \\ \text{[... vector]} \\ \text{[vector “zébresse”]} \end{pmatrix}}_{\mathbf{Y}}$$

Subconjunto del vocabulario total

Pasos para aprender R

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Pasos para aprender R

initialize R

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Pasos para aprender R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Pasos para aprender R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss$$

gradient

Pasos para aprender R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss \qquad \text{gradient}$$

$$R = R - \alpha g \qquad \text{update}$$

Norma Frobenius

$$\| \mathbf{XR} - \mathbf{Y} \|_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

Norma Frobenius

$$\| \mathbf{XR} - \mathbf{Y} \|_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Norma Frobenius al cuadrado

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

$$\|\mathbf{A}\|_F^2 = 16$$

Gradiente

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR}Loss = \frac{2}{m}\left(\mathbf{X}^T(\mathbf{XR} - \mathbf{Y})\right)$$

Encontrar la traducción

“hola”

Matriz R

[transformado]

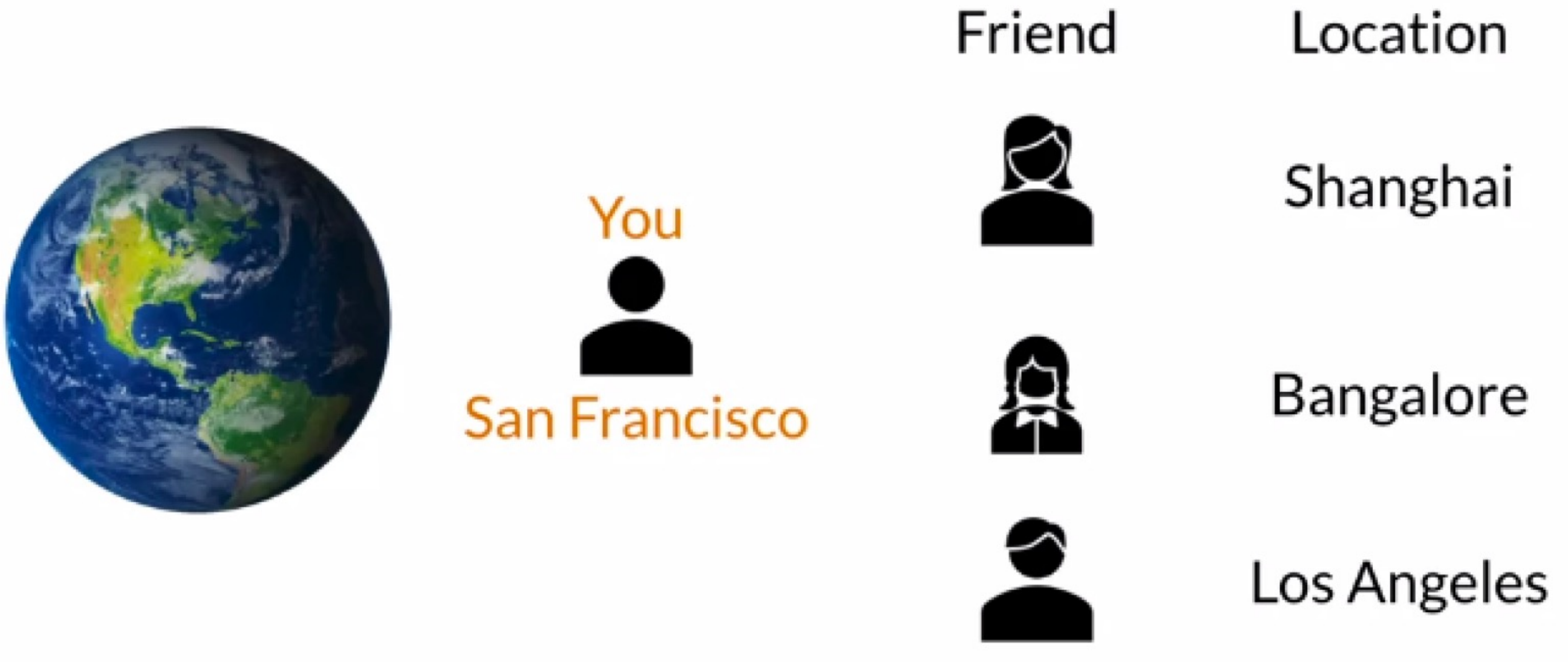
Encontrar palabras similares

“salut”

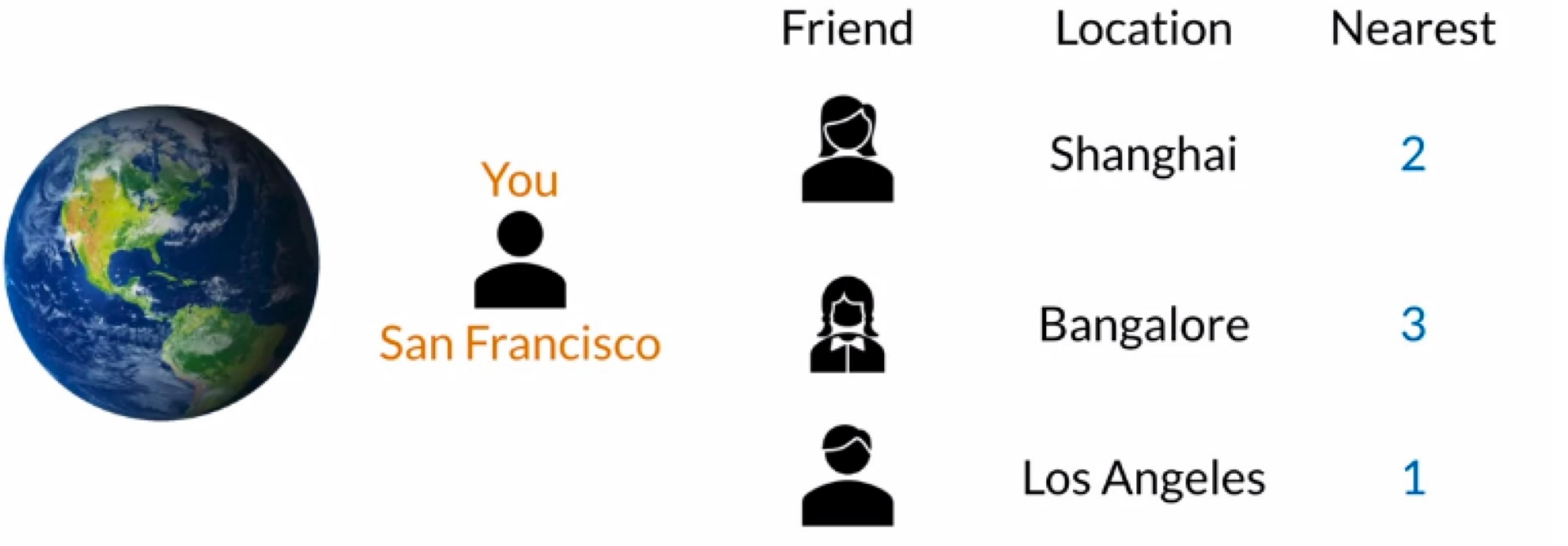
“bonjour”

Vecinos más cercanos

Vecinos más cercanos

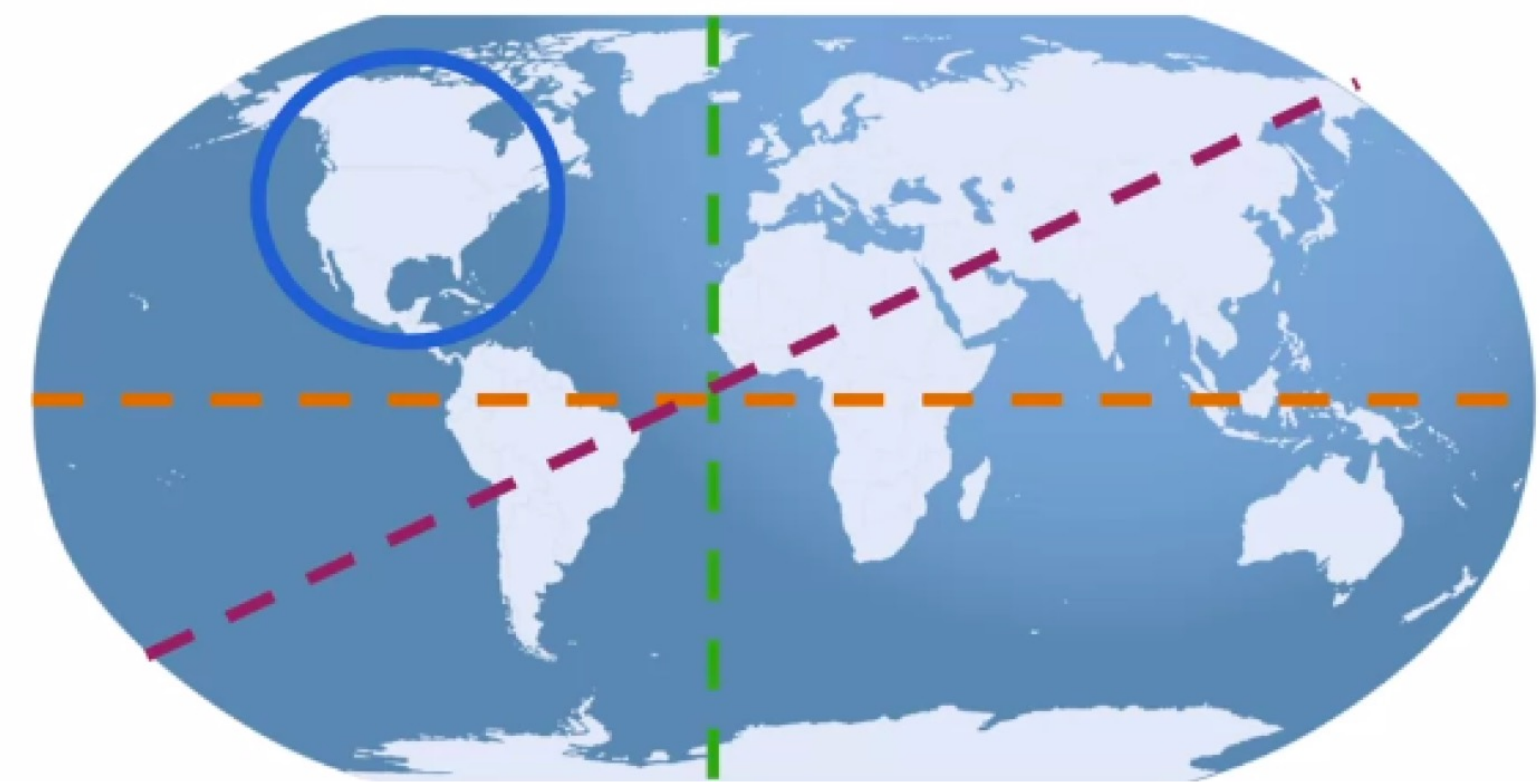


Vecinos más cercanos



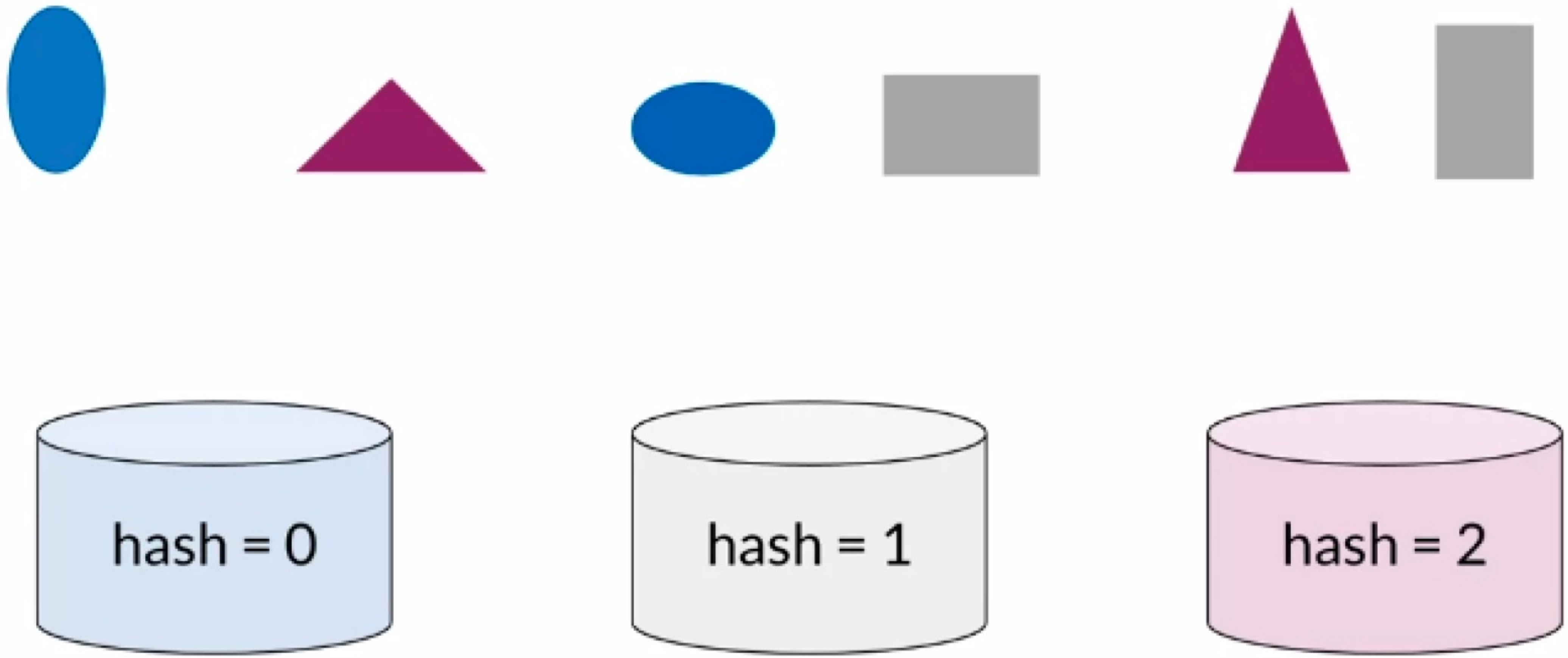
Vecinos más cercanos

Vecinos más cercanos

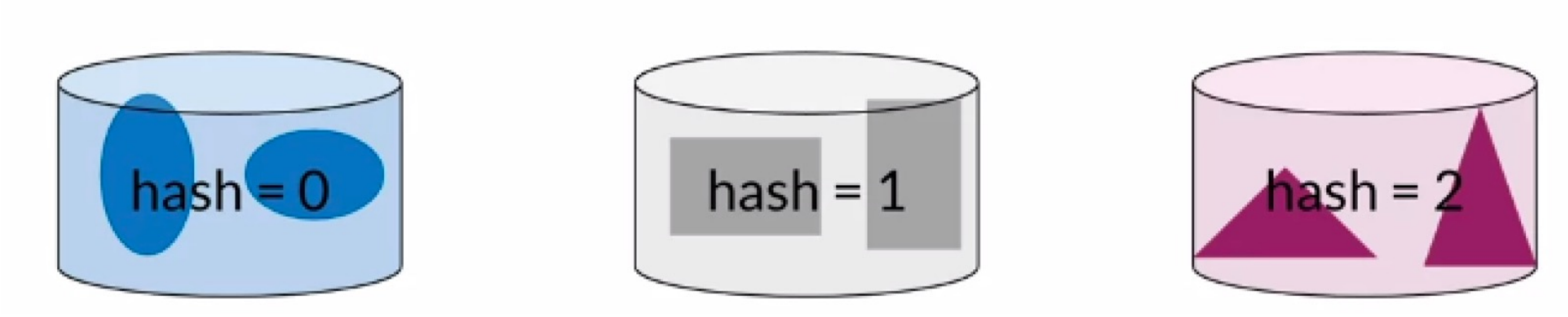


Tablas hash

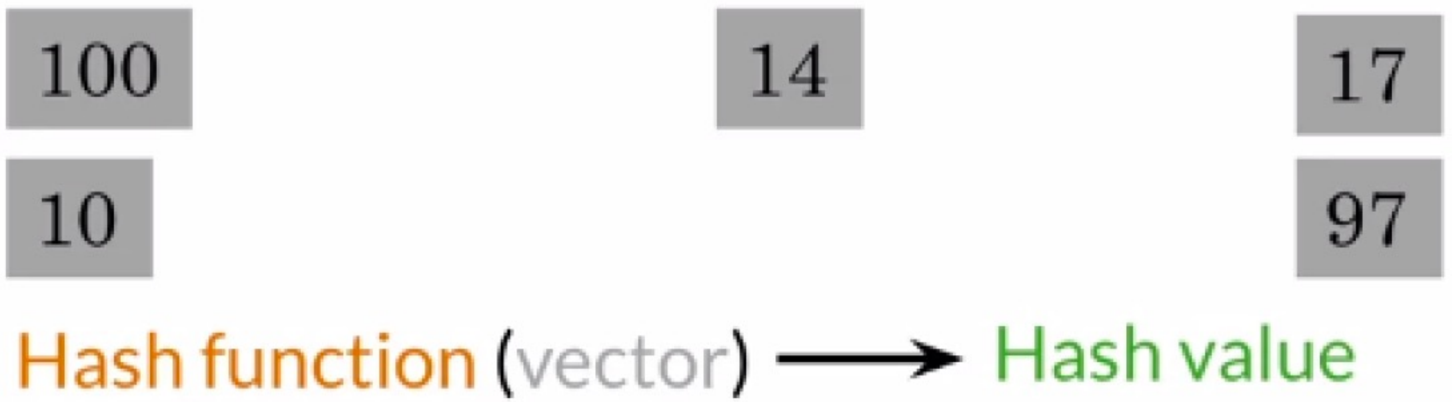
Tablas hash



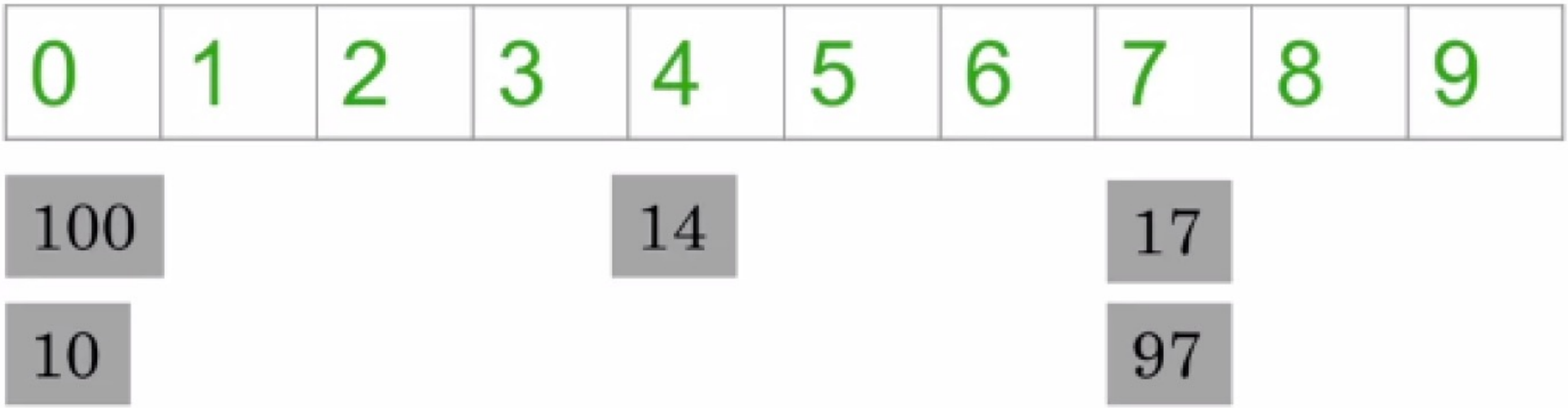
Tablas hash



Tablas hash

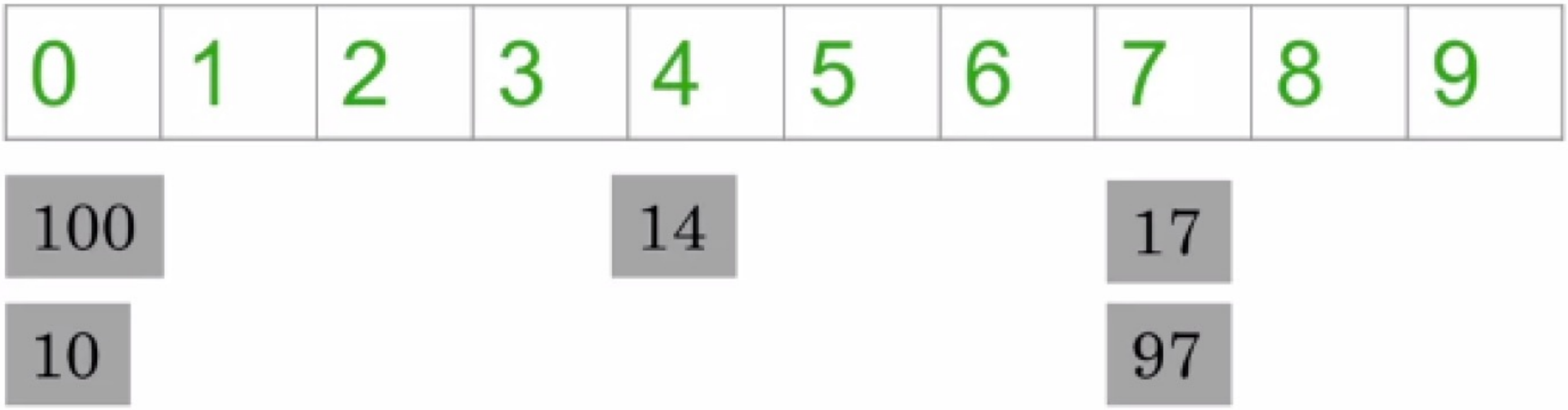


Tablas hash



Función hash (vector)= → valor hash

Tablas hash



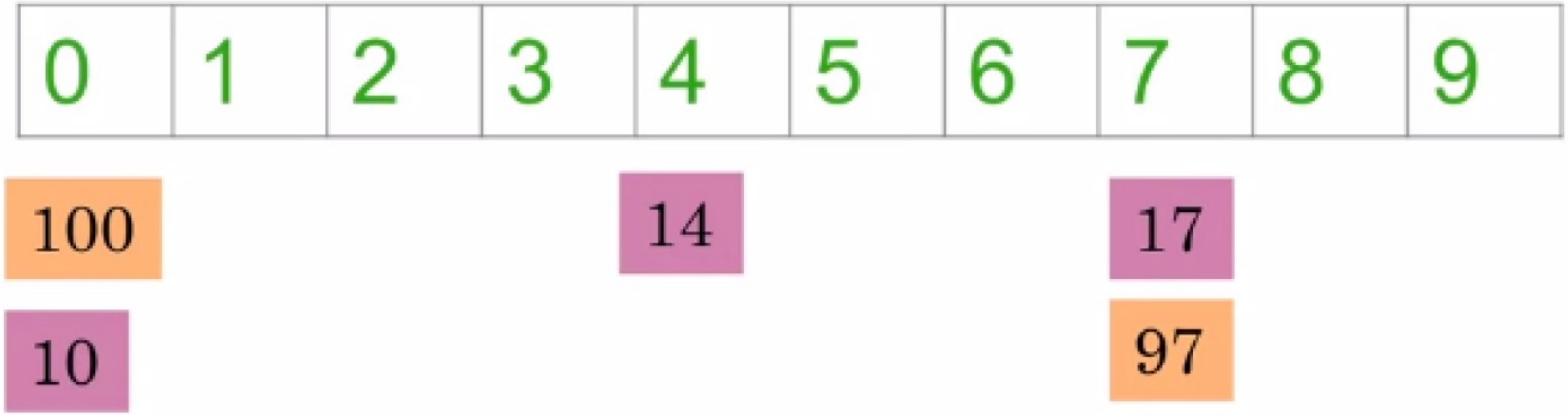
Función hash (vector)= → valor hash

Valor hash= vector % número de cubetas

Creación de tabla hash básica

```python
def basic_hash_table(value_l,n_buckets):

    def hash_function(value_l,n_buckets):
        return int(value) % n_buckets

    hash_table = {i:[] for i in range(n_buckets)}
    for value in value_l:
        hash_value = hash_function(value,n_buckets)
        hash_table[hash_value].append(value)

    return hash_table
```

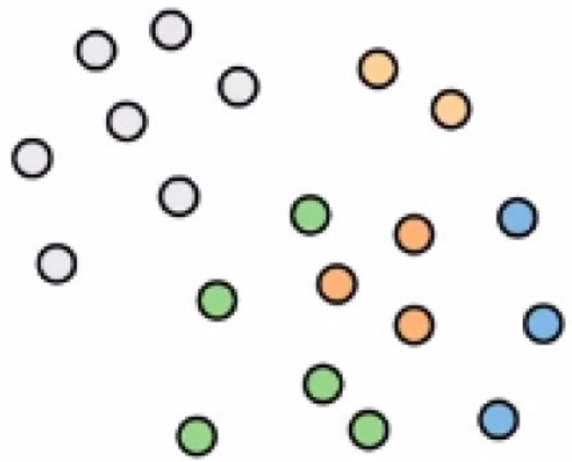
Función hash



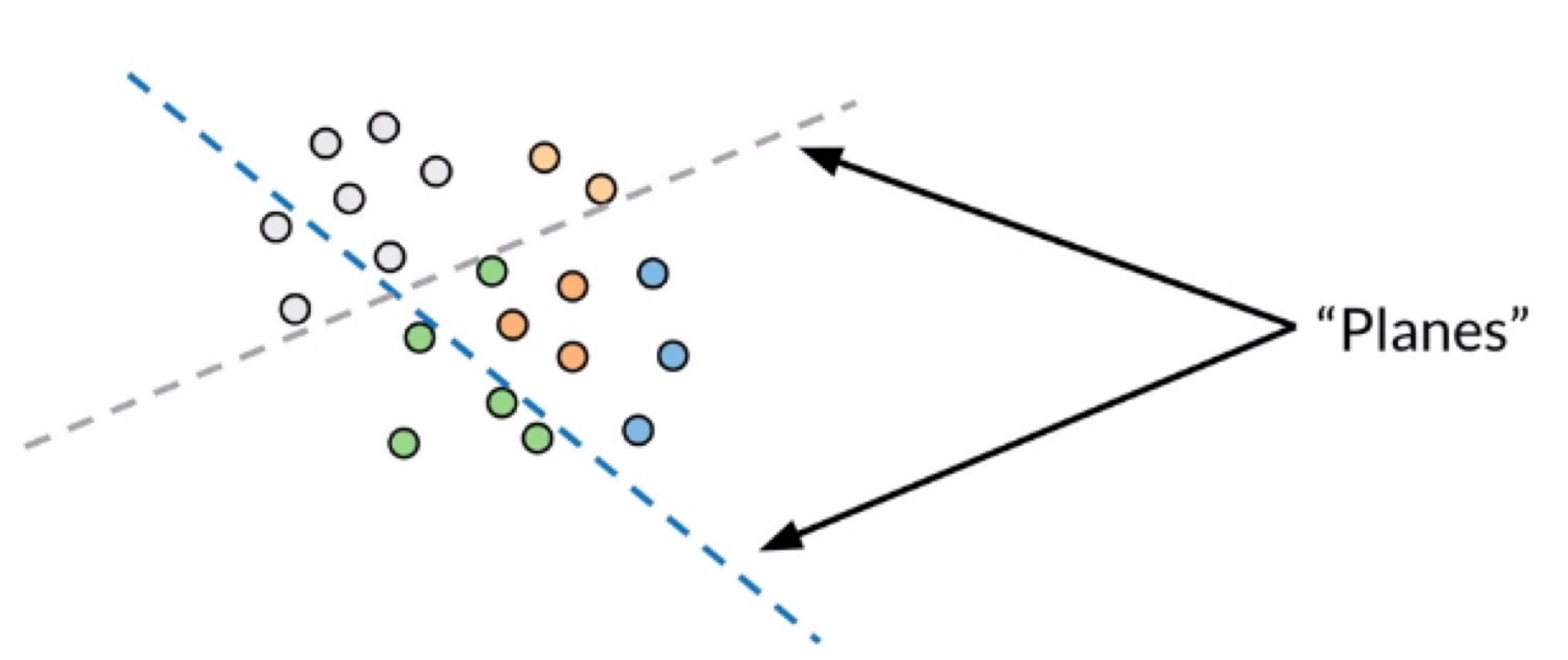
Función hash por ubicación?

Idealmente queremos que las palabras (números) cercanas queden en la misma cubeta.
Solución: Hashing sensible a la localidad (locality sensitive)

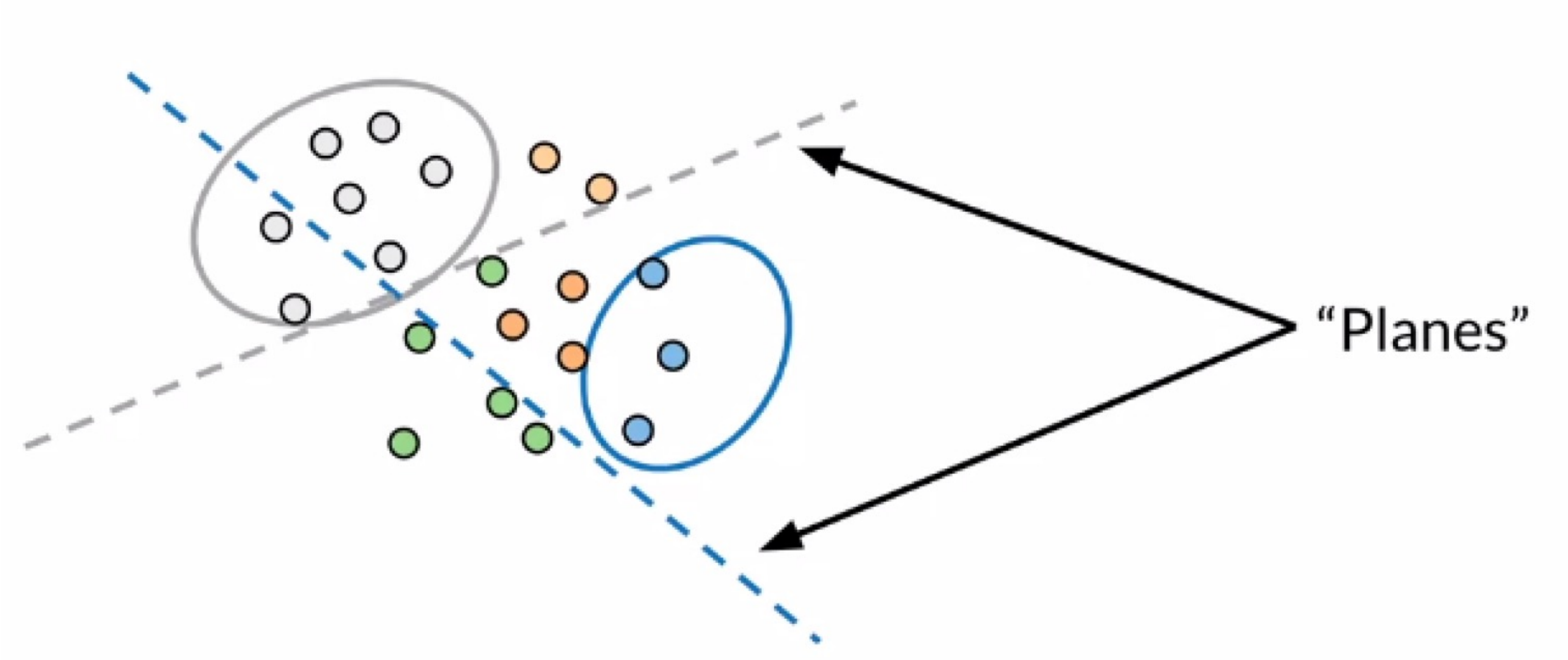
Locality sensitive hashing



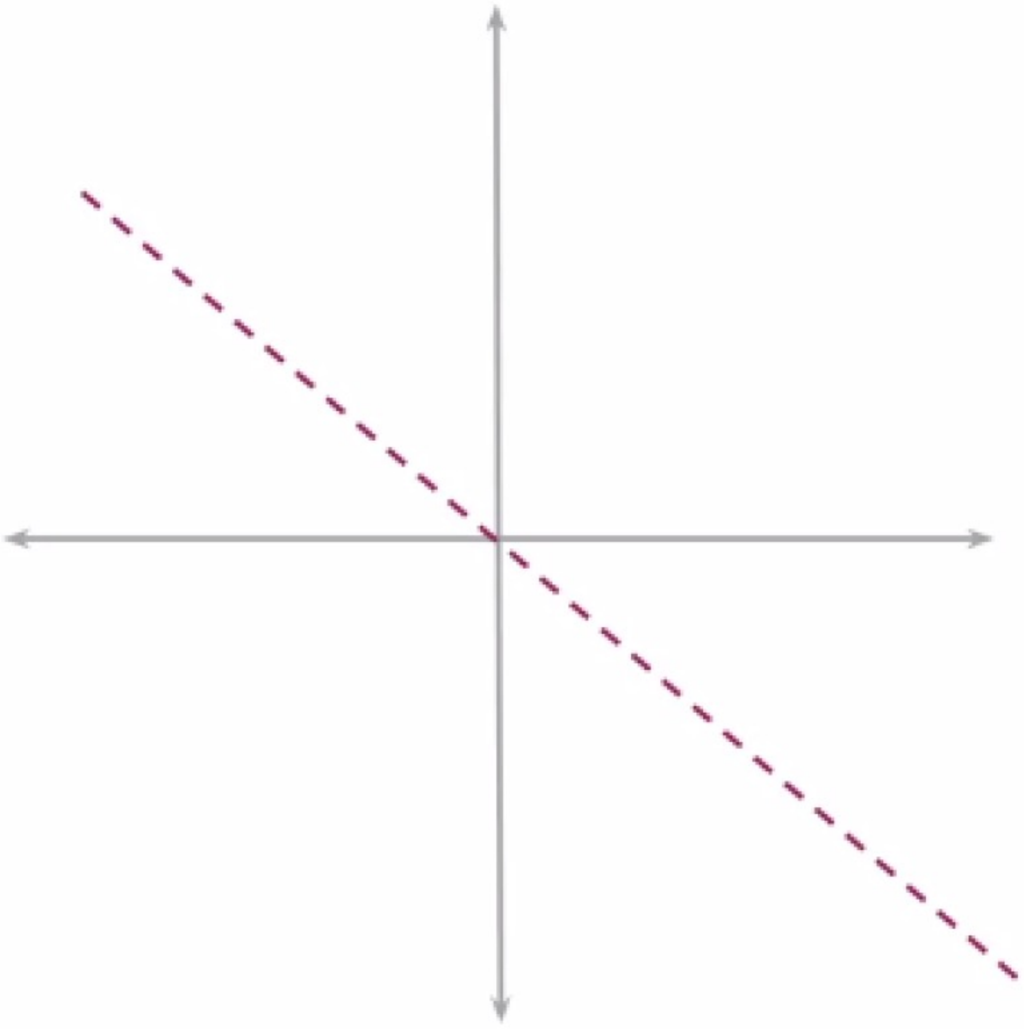
Locality sensitive hashing



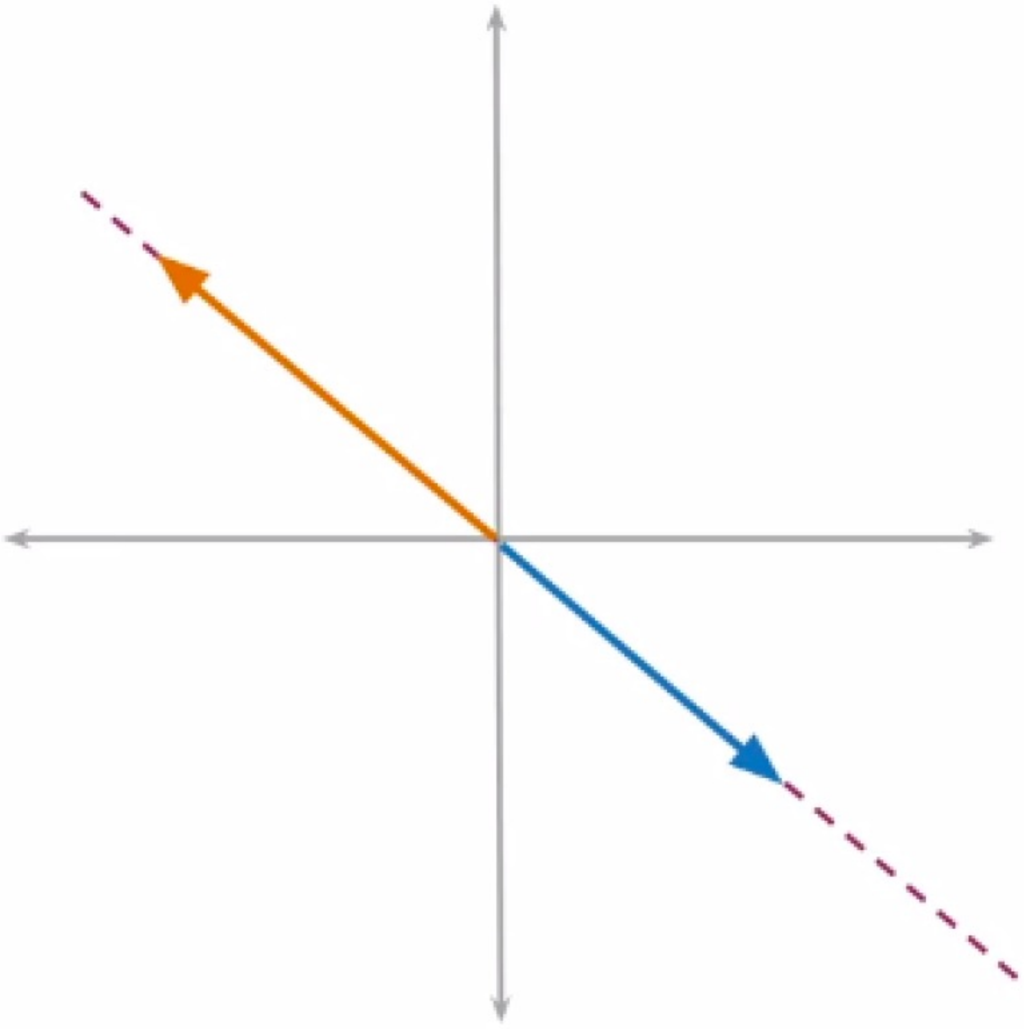
Locality sensitive hashing



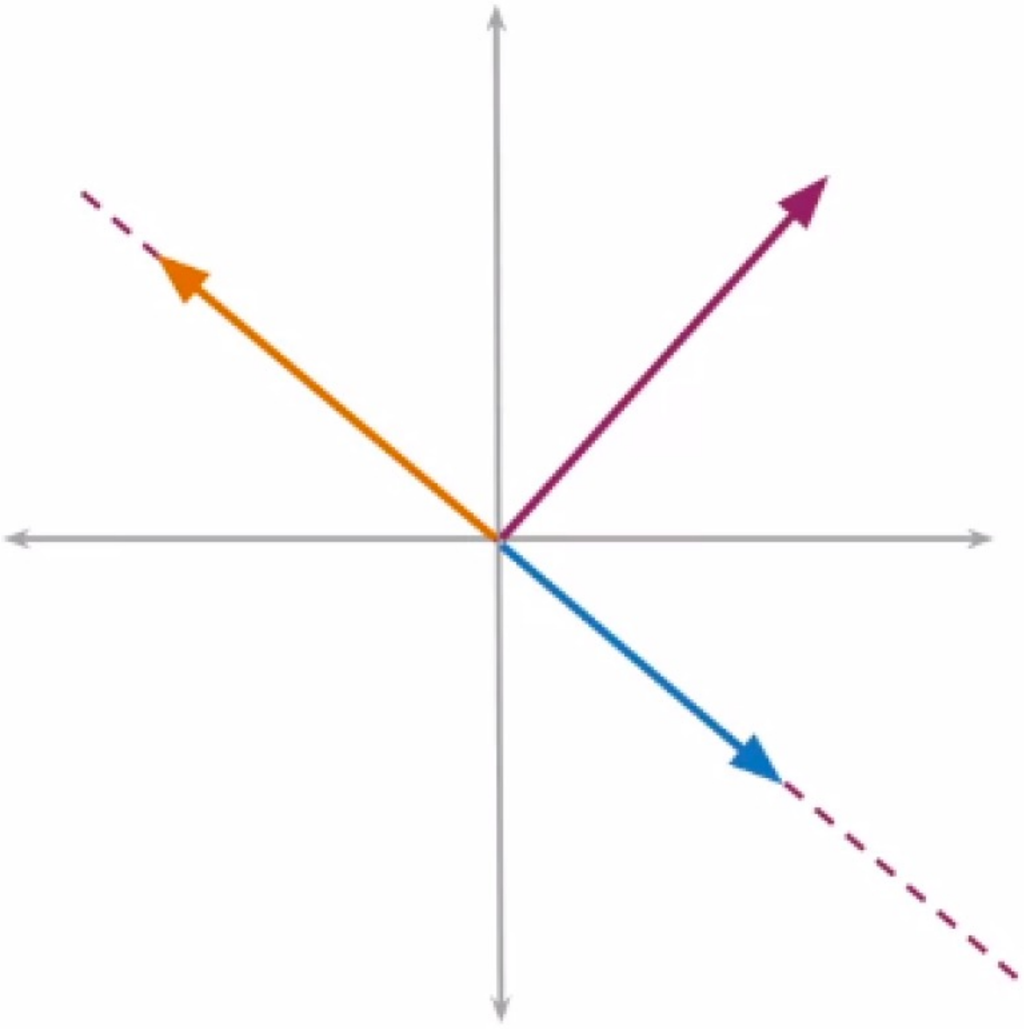
Planos



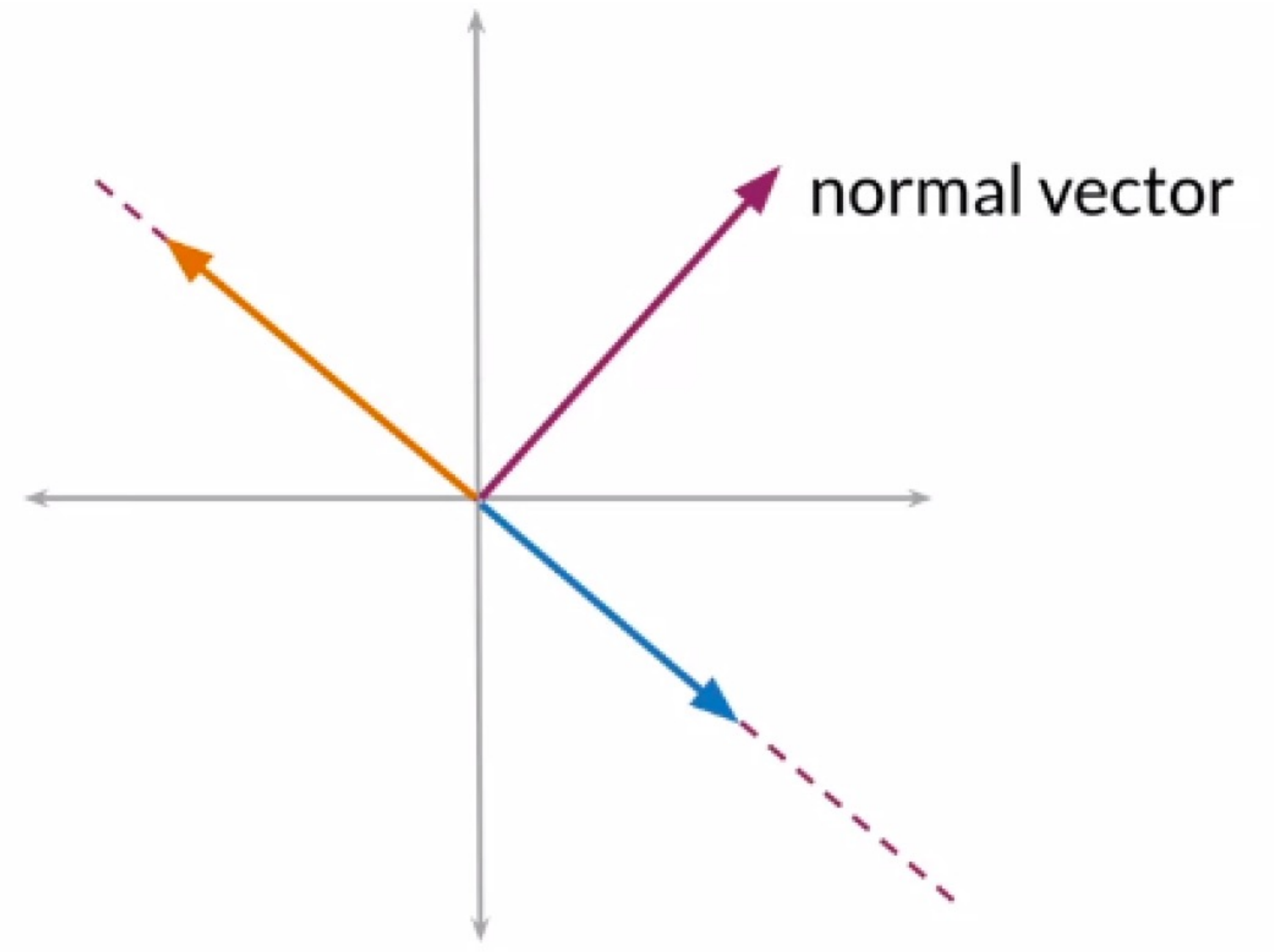
Planos



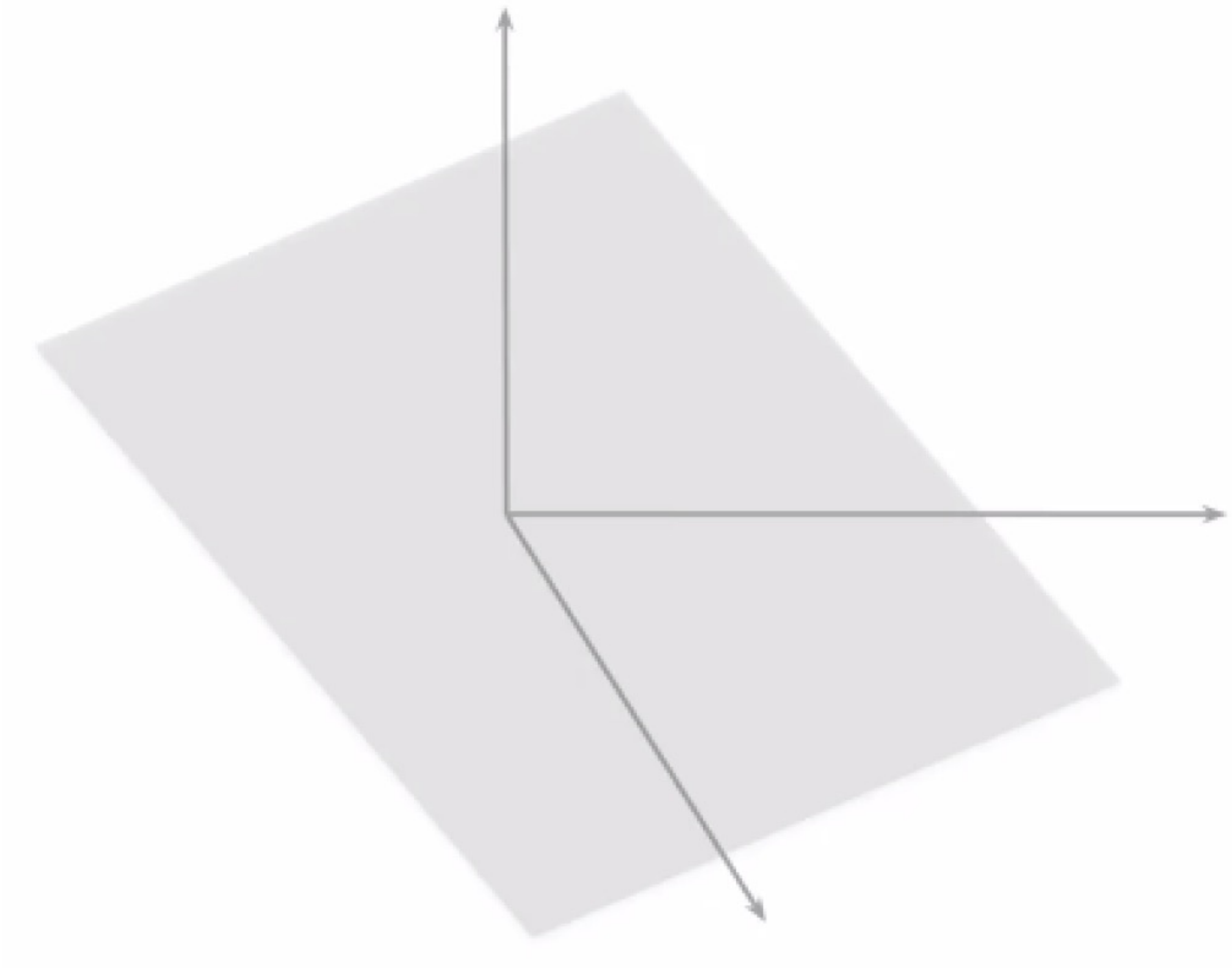
Planos



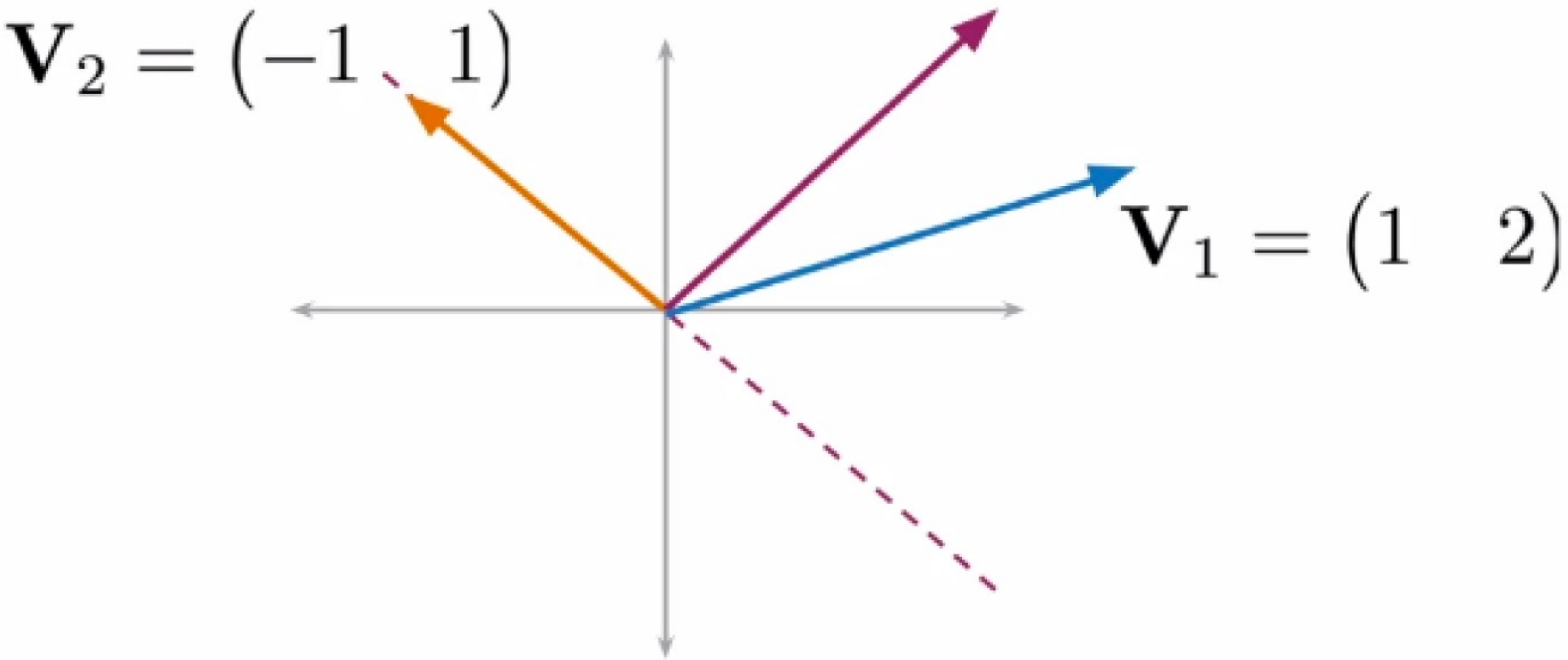
Planos



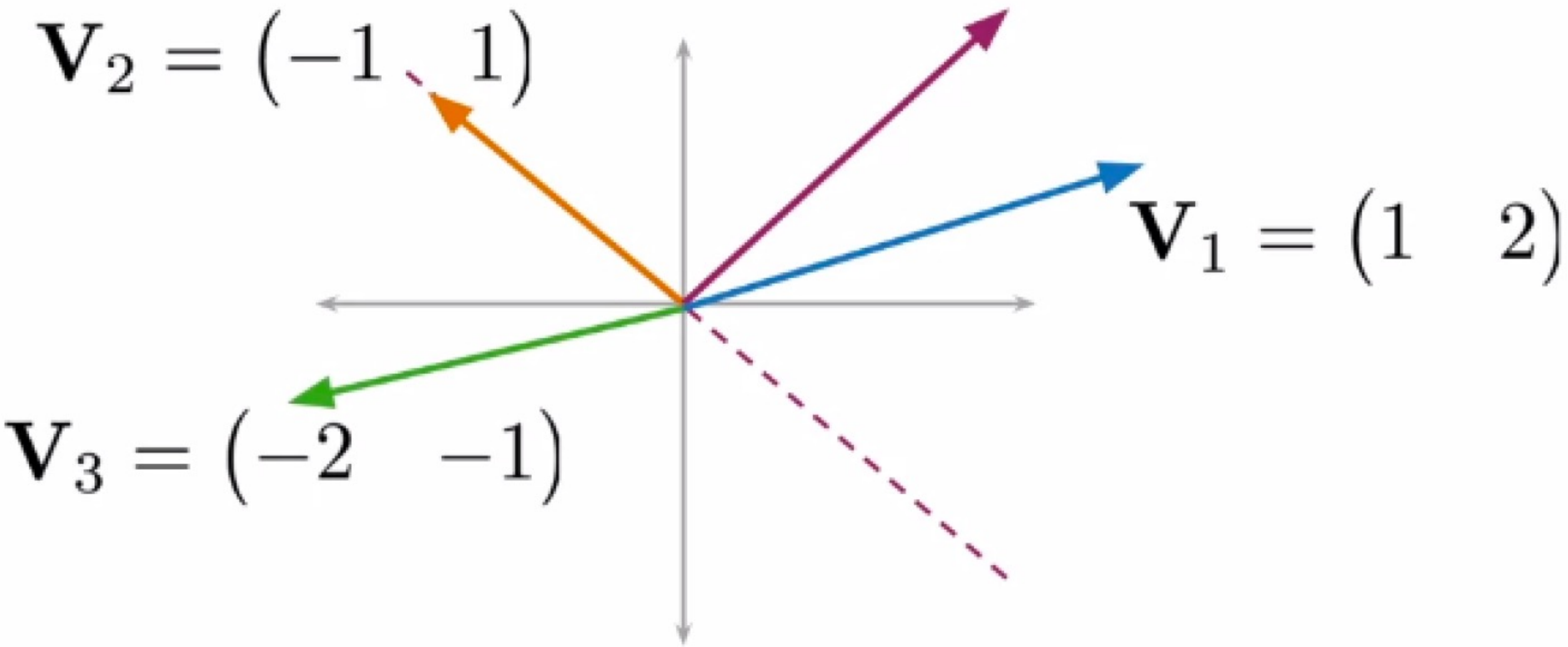
Planos



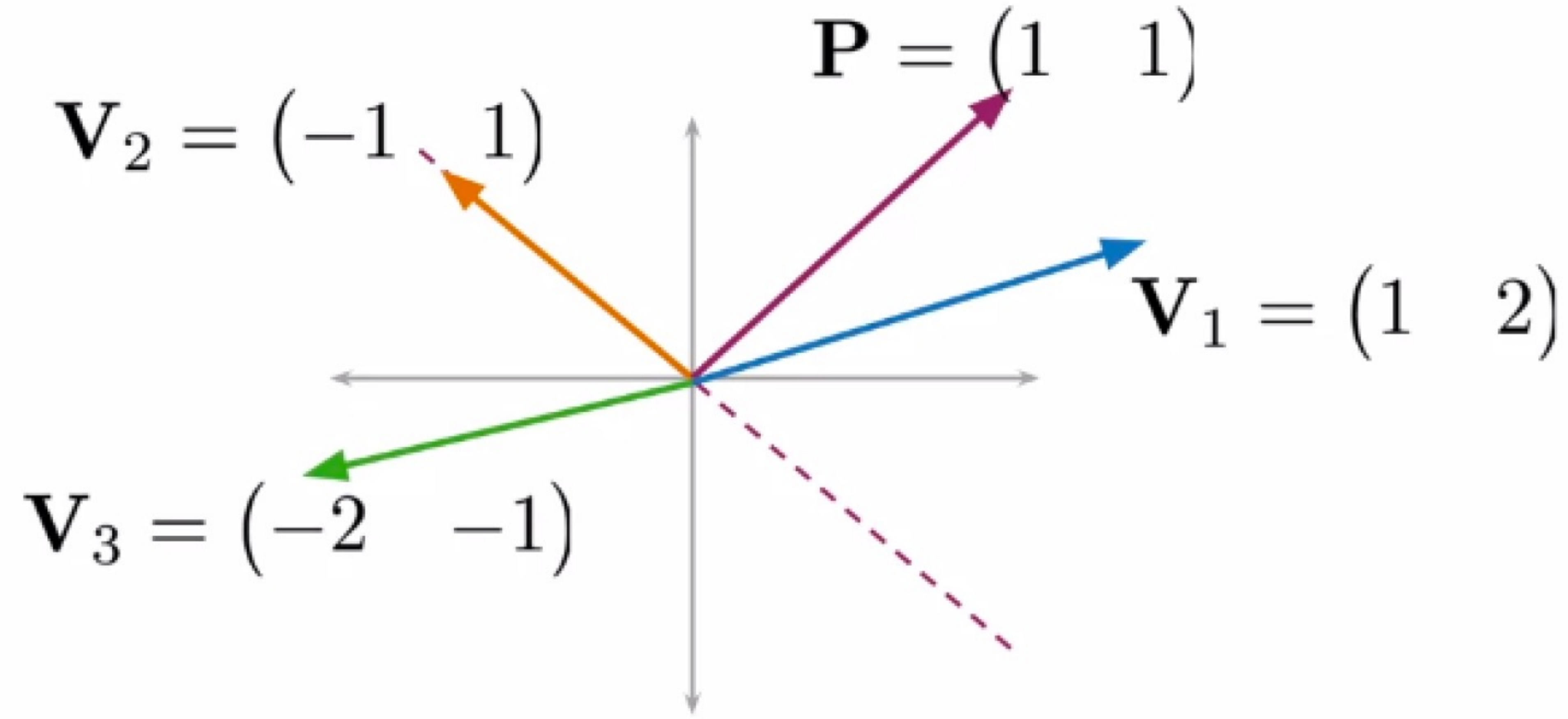
Cuál lado del plano?



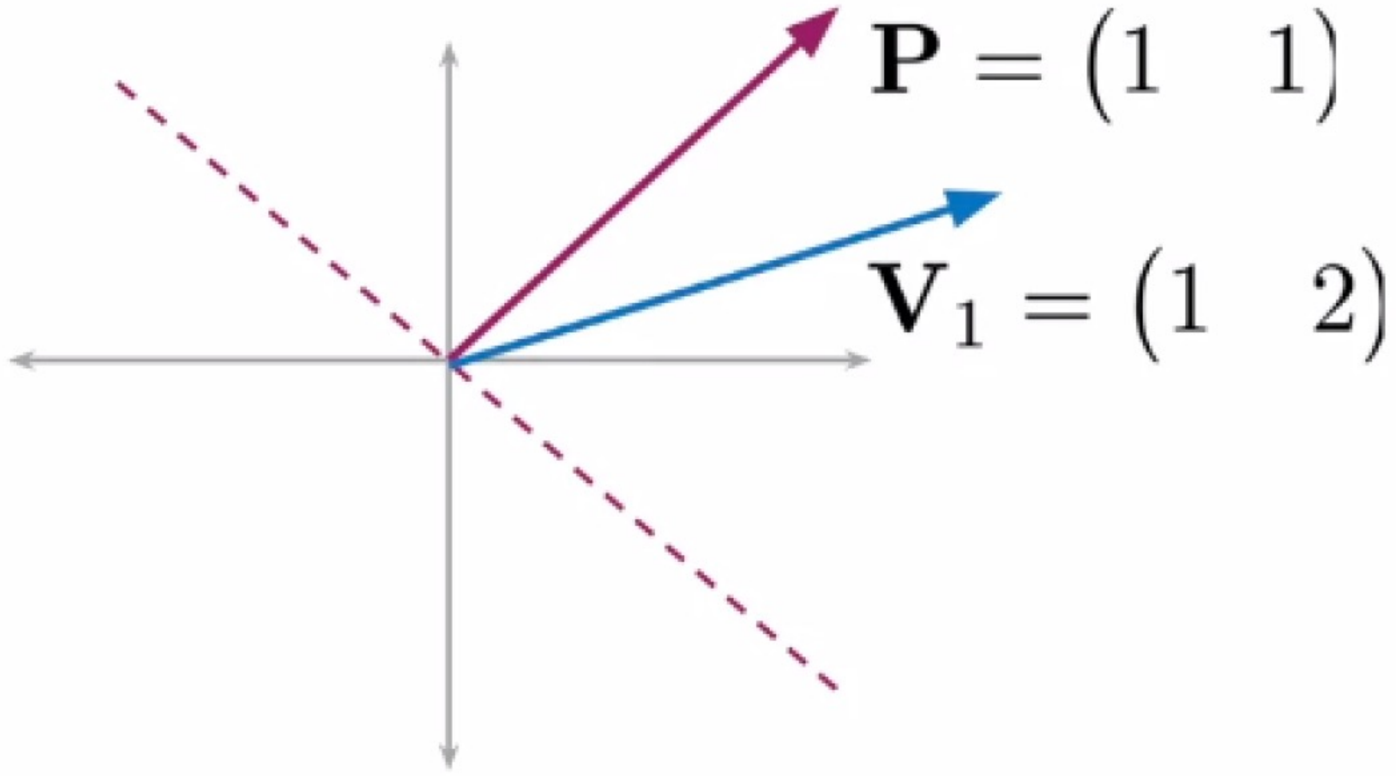
Cuál lado del plano?



Cuál lado del plano?



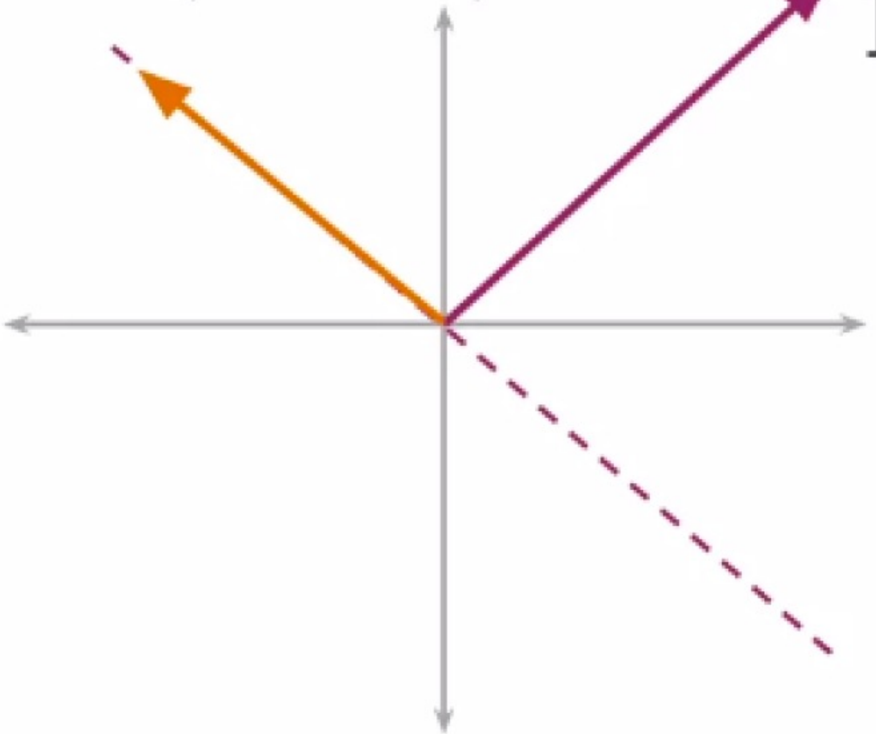
Cuál lado del plano?



$$\mathbf{P}\mathbf{V}_1^T = 3$$

Cuál lado del plano?

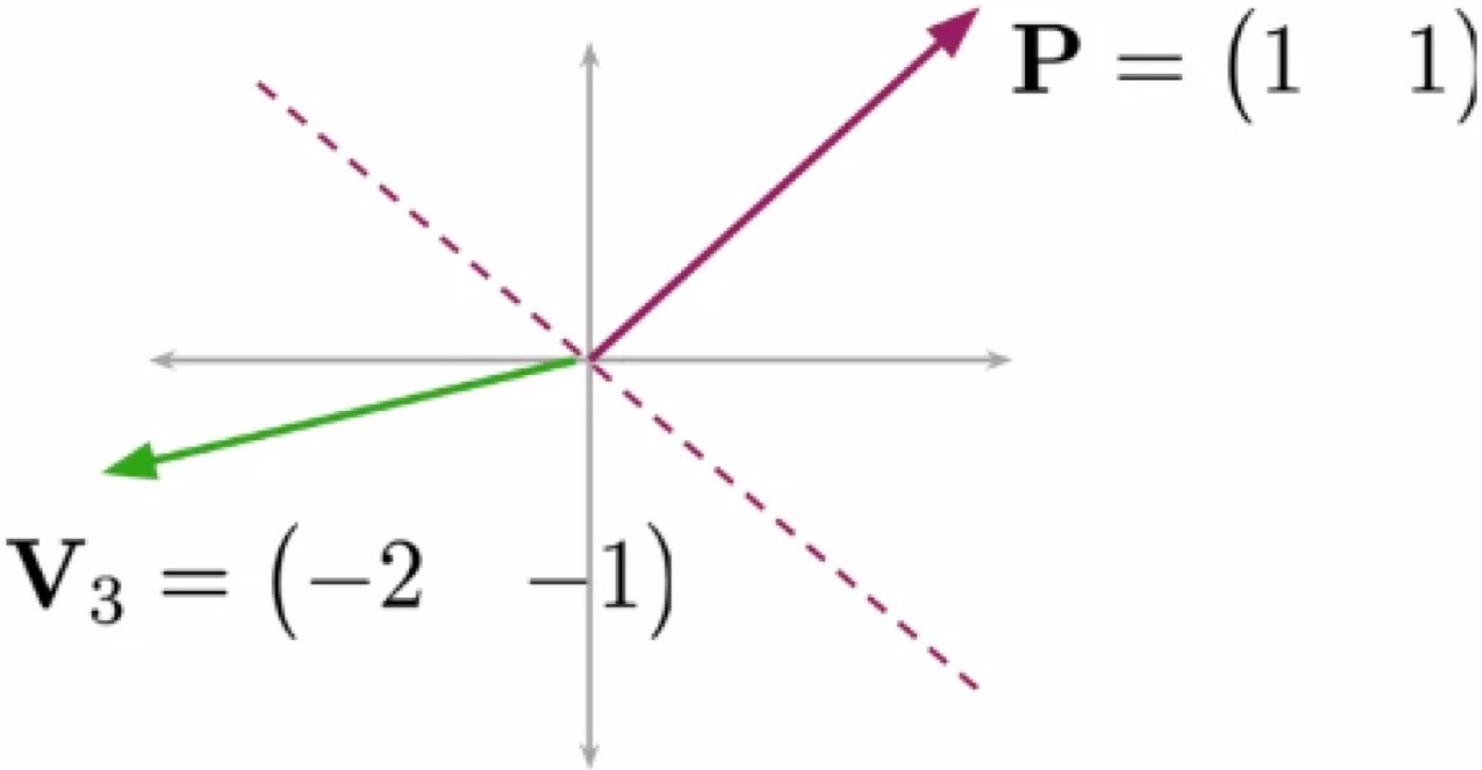
$$\mathbf{V}_2 = \begin{pmatrix} -1 & 1 \end{pmatrix}$$



$$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

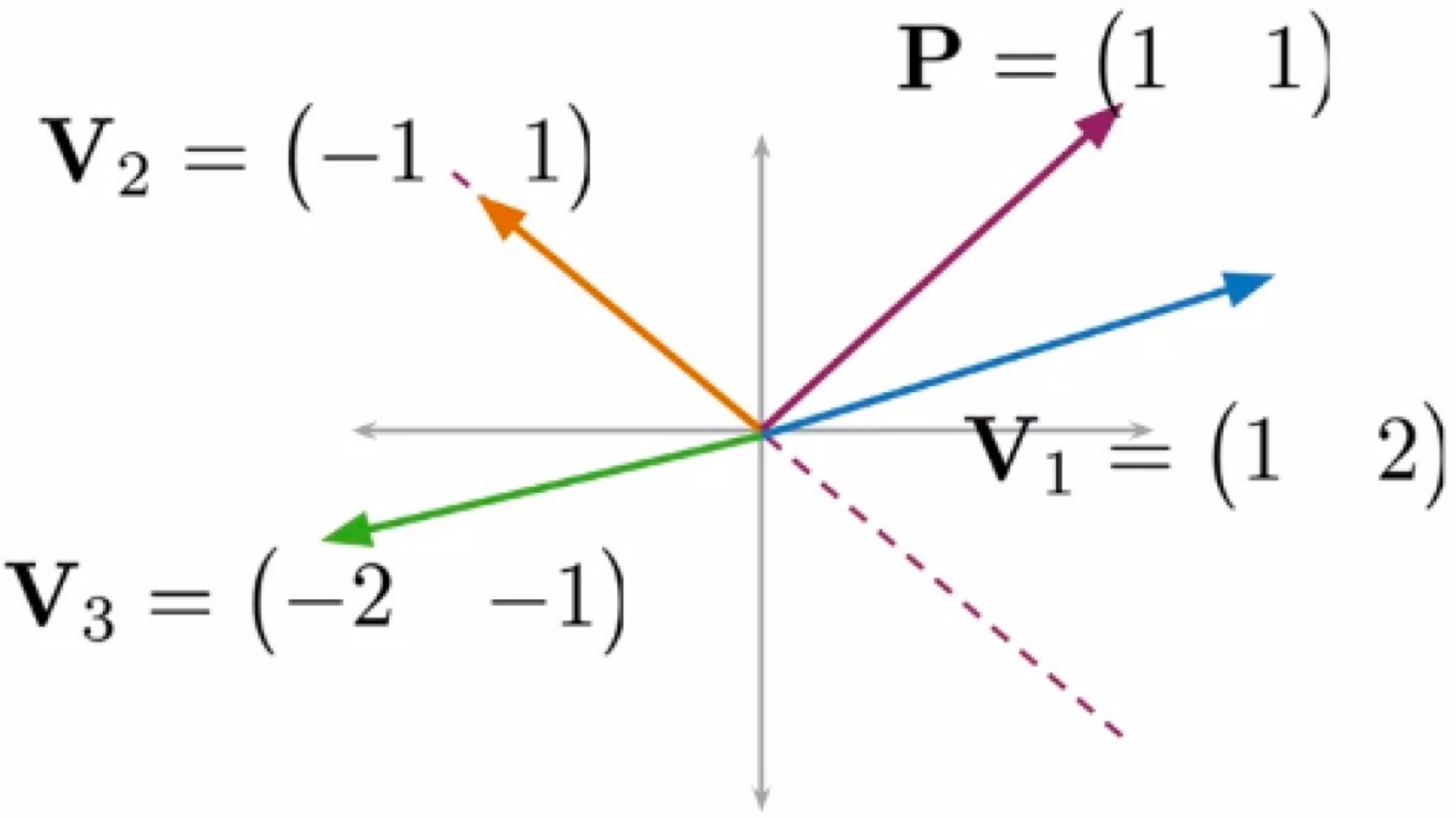
$$\mathbf{P}\mathbf{V}_2^T = 0$$

Cuál lado del plano?



$$\mathbf{P}\mathbf{V}_3^T = -3$$

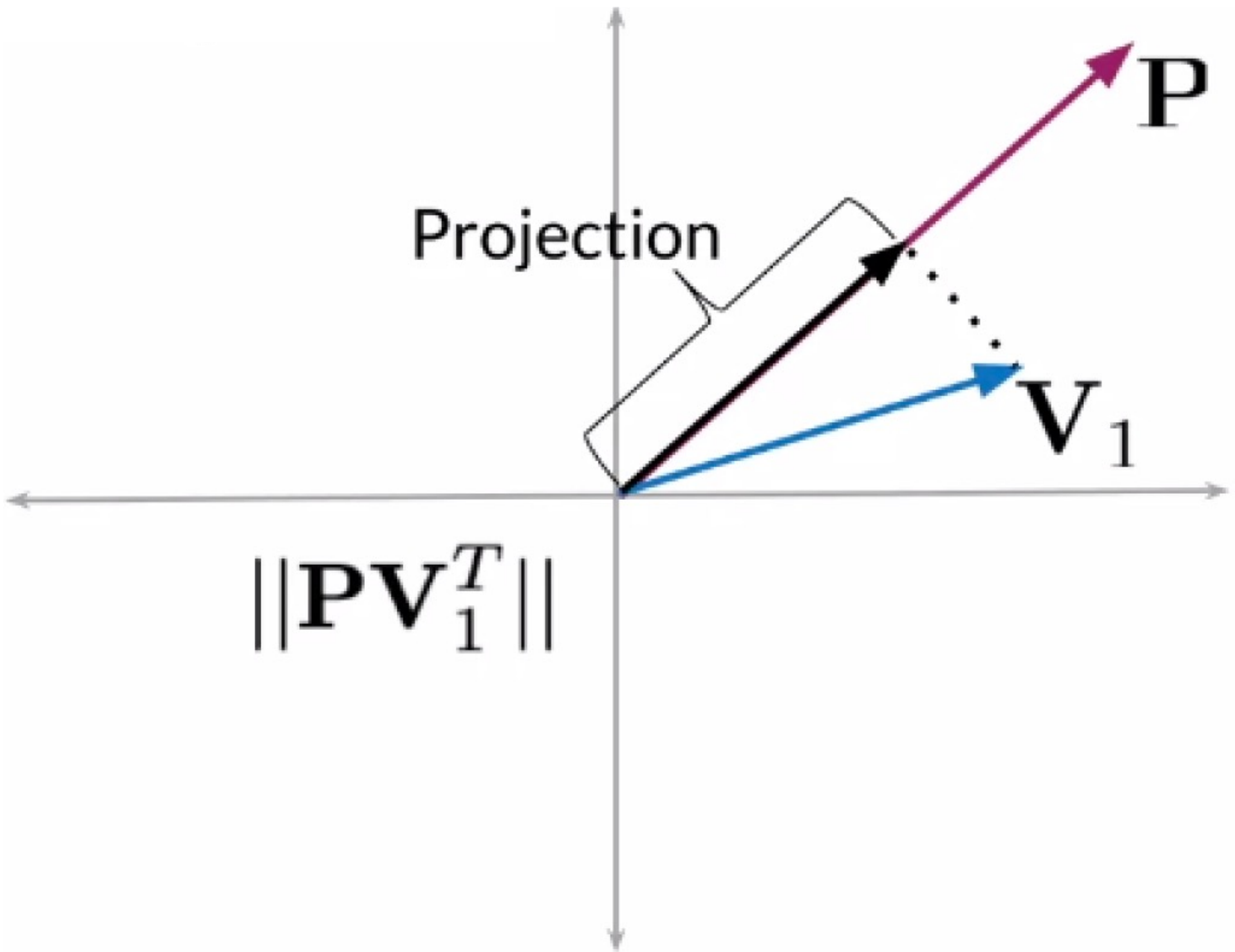
Cuál lado del plano?



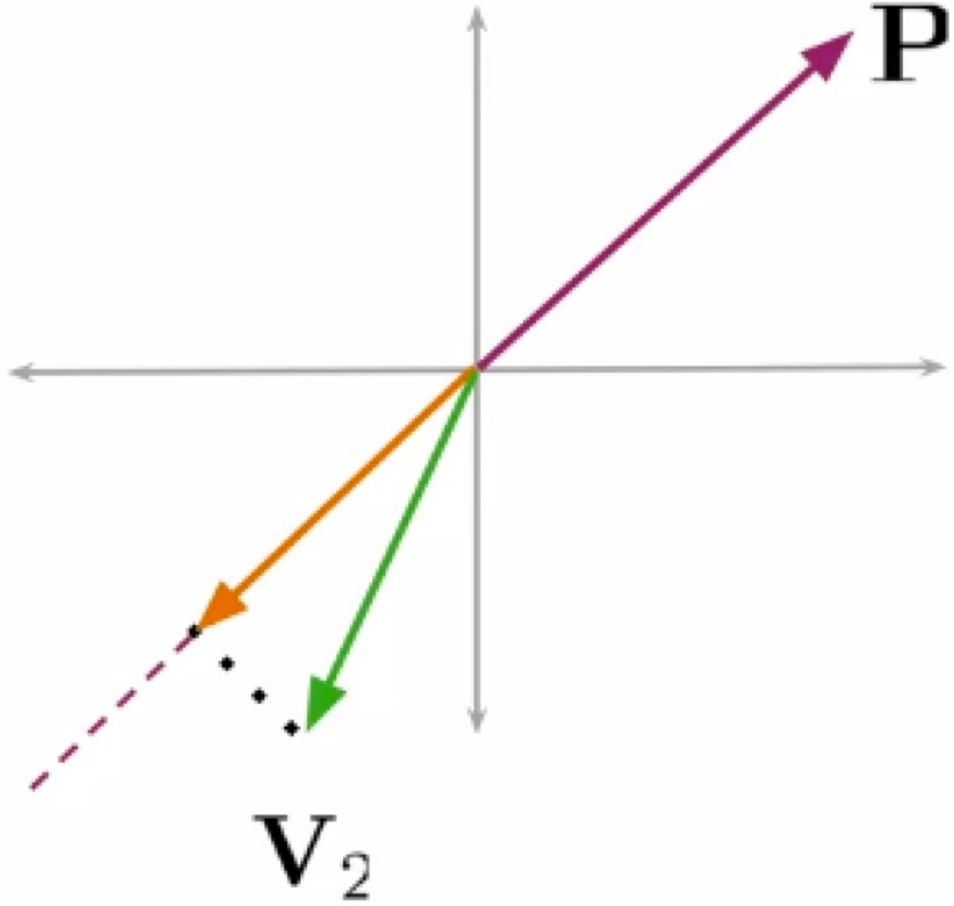
$$\mathbf{PV}_1^T = 3$$
$$\mathbf{PV}_2^T = 0$$
$$\mathbf{PV}_3^T = -3$$

Nótese los signos

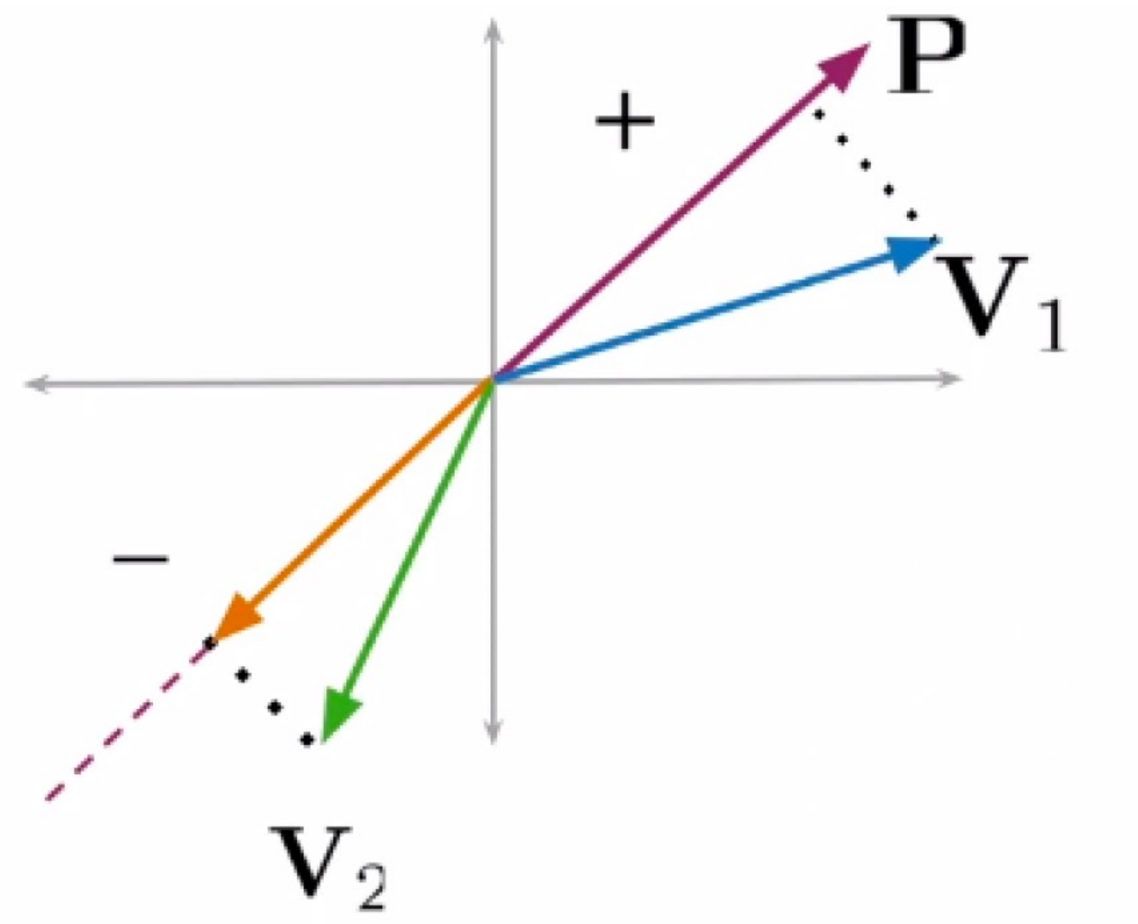
Visualizando un producto punto



Visualizando un producto punto



Visualizando un producto punto



El signo indica dirección

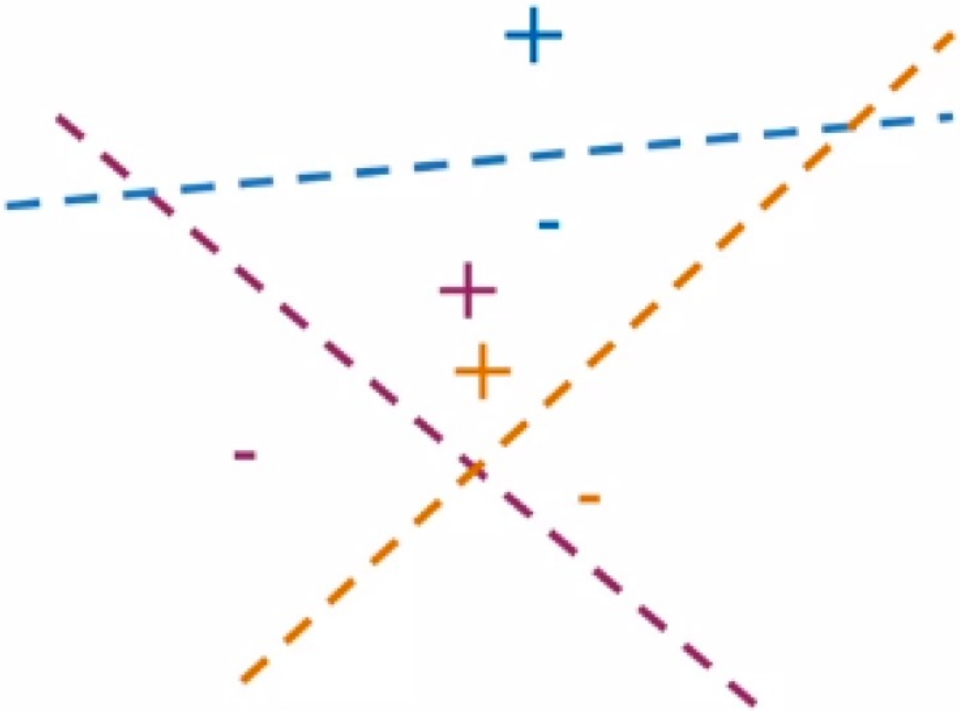
¿Cuál lado del plano?

```python
def side_of_plane(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    sign_of_dot_product_scalar= np.asscalar(sign_of_dot_product)
    return sign_of_dot_product_scalar
```

Múltiples planos

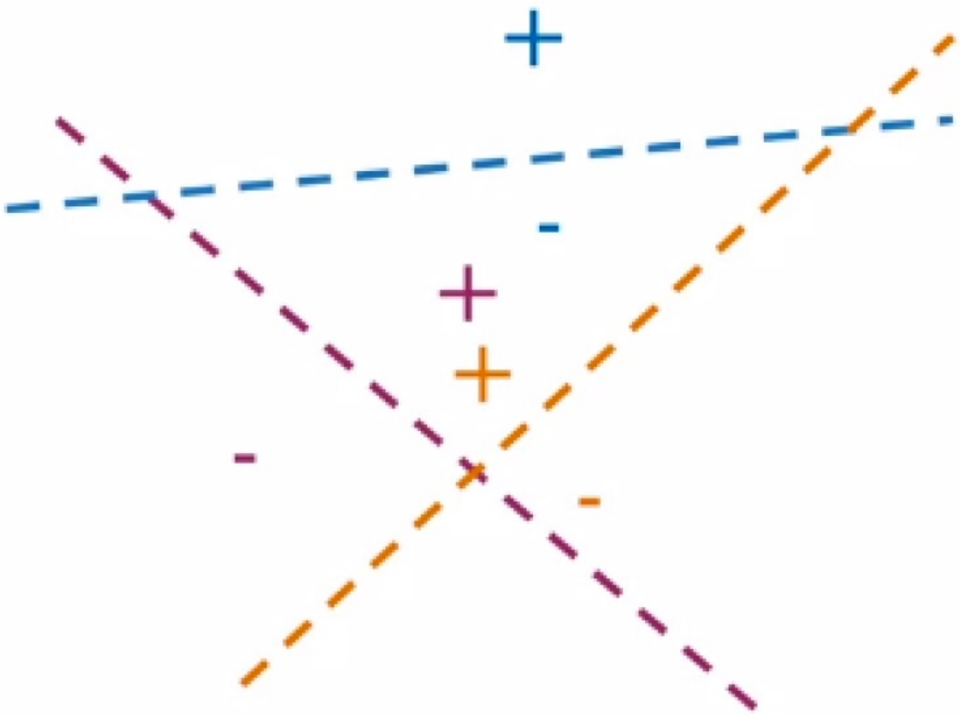
Múltiples planos → Productos puntos → Valores Hash

Múltiples planos, valor hash único?



$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

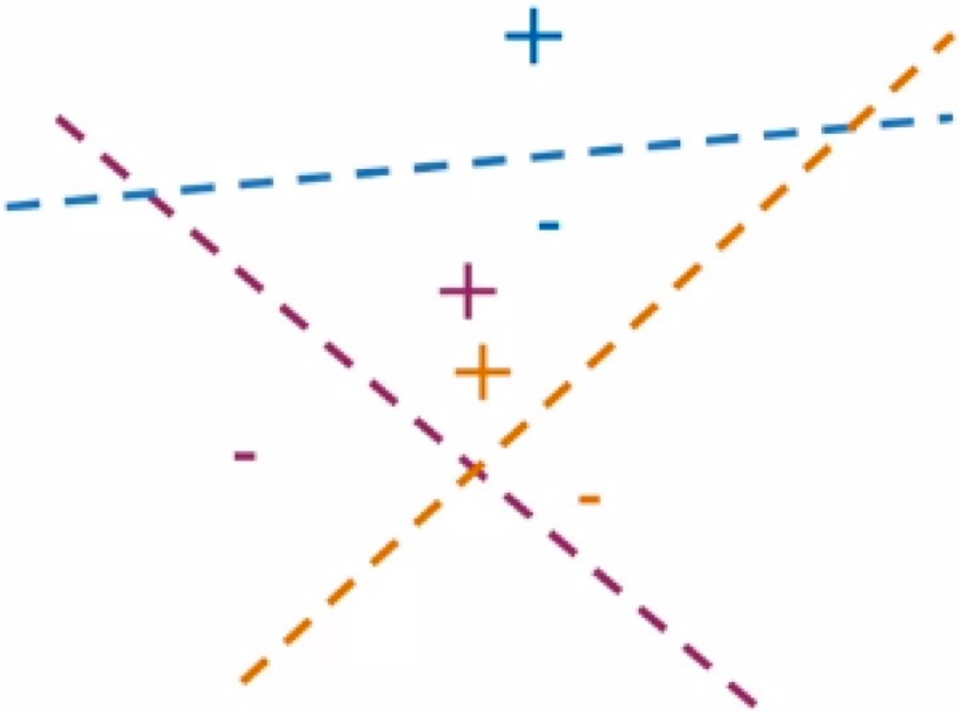
Múltiples planos, valor hash único?



$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

Múltiples planos, valor hash único?

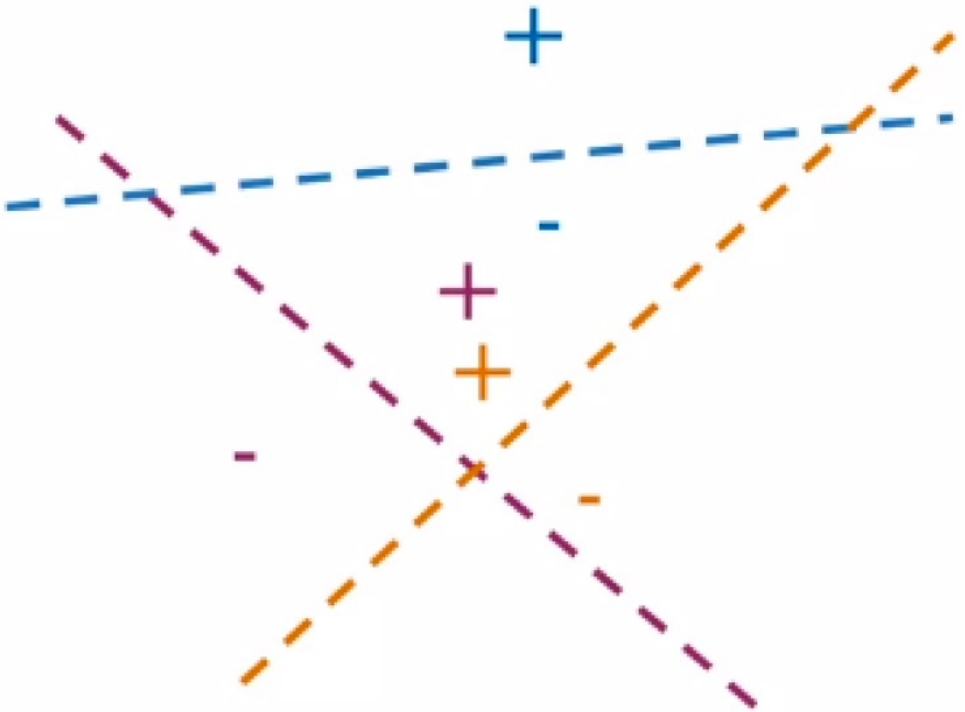


$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

$$\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0$$

Múltiples planos, valor hash único?



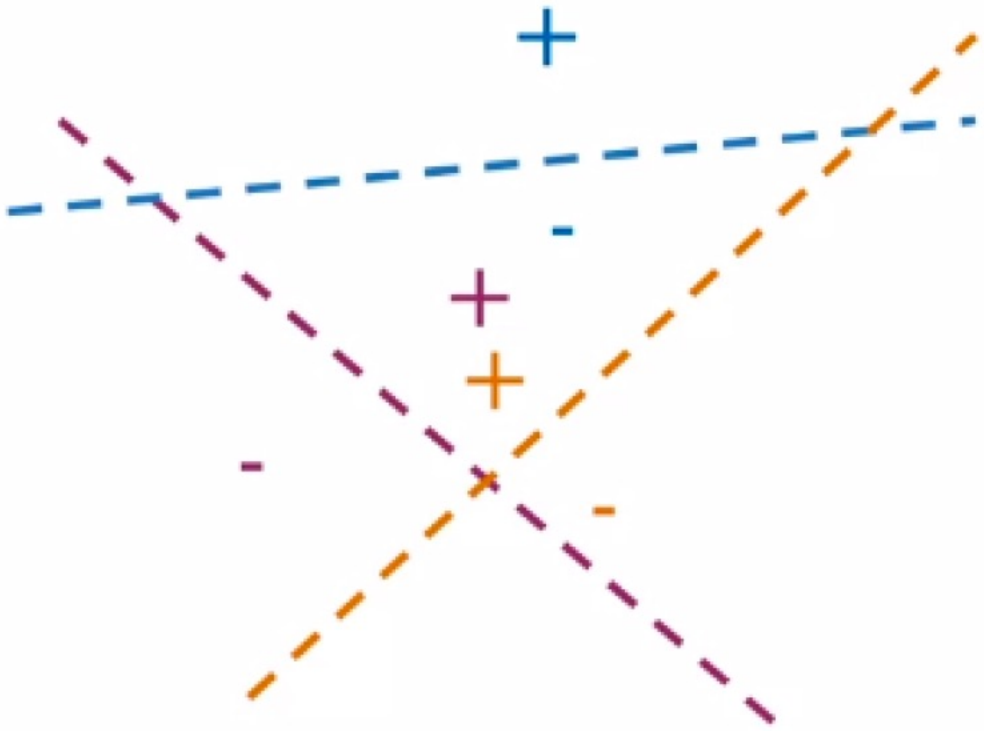
$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

$$\boxed{\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1}$$

$$\boxed{\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0}$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$
$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$
$$= 3$$

Múltiples planos, valor hash único!



$$sign_i \geq 0, \rightarrow h_i = 1$$

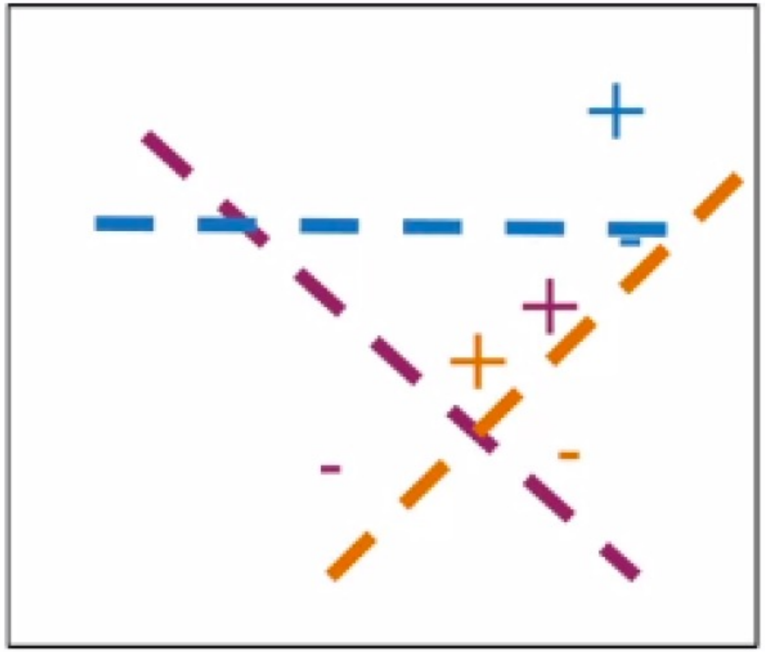
$$sign_i < 0, \rightarrow h_i = 0$$

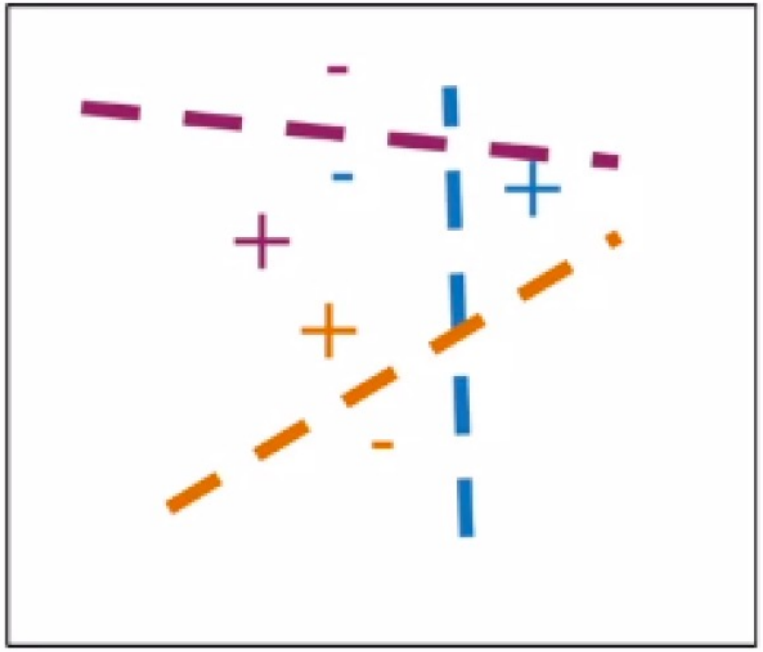
$$\text{hash} = \sum_{i}^{H} 2^i \times h_i$$

Múltiples planos, valor hash único!!

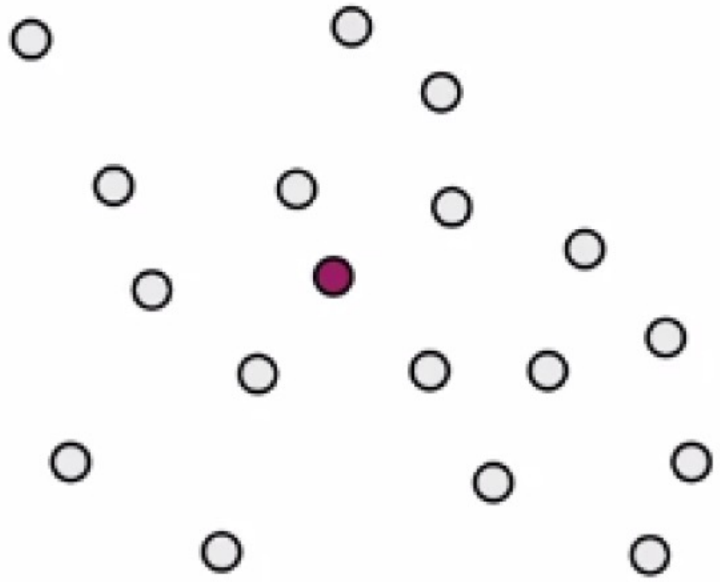
```
def hash_multiple_plane(P_l,v):

    hash_value = 0

    for i, P in enumerate(P_l):
        sign = side_of_plane(P,v)
        hash_i = 1 if sign >=0 else 0
        hash_value += 2**i * hash_i

    return hash_value
```

Planos aleatorios



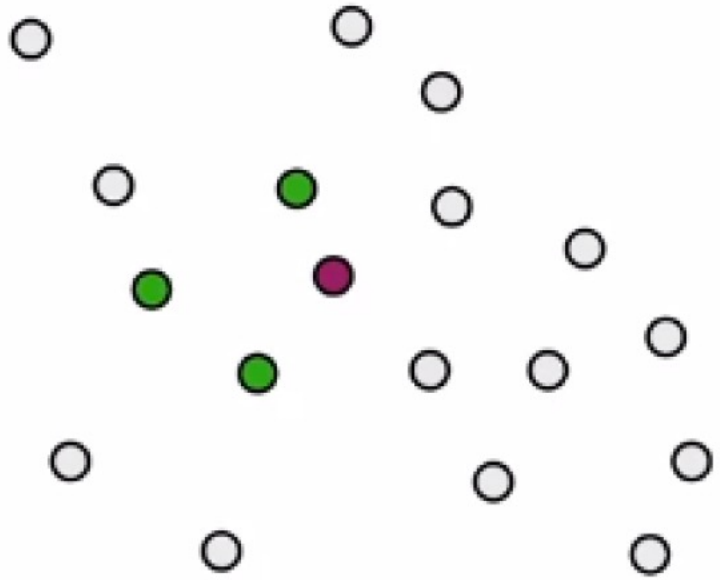


Múltiples conjuntos de planos aleatorios



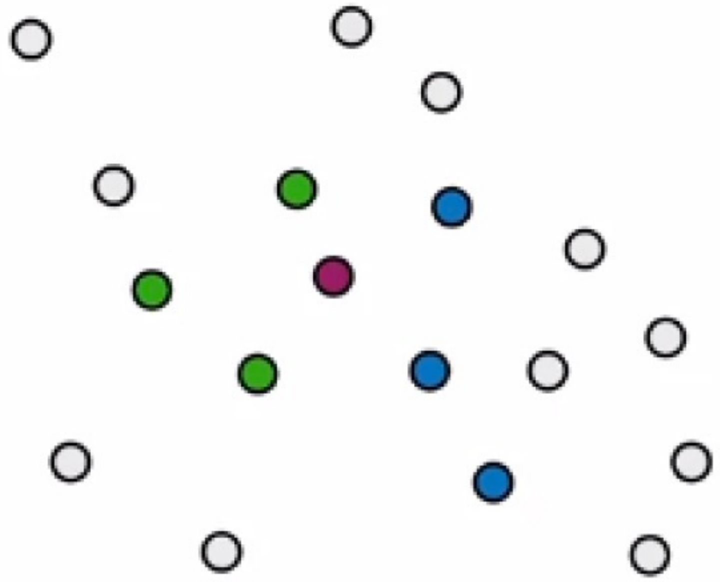
Supongamos que estamos tratando de encontrar el vecino más cercano para el vector rojo (punto).

Múltiples conjuntos de planos aleatorios



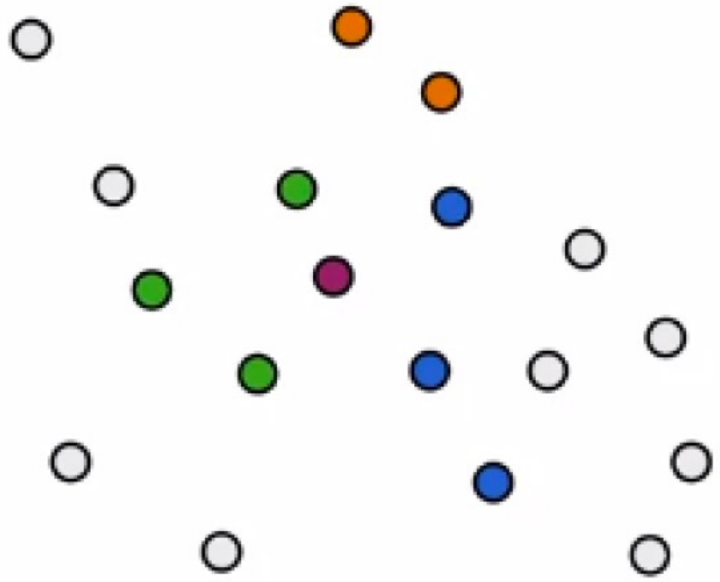
La primera vez, el plano nos da los puntos verdes

Múltiples conjuntos de planos aleatorios



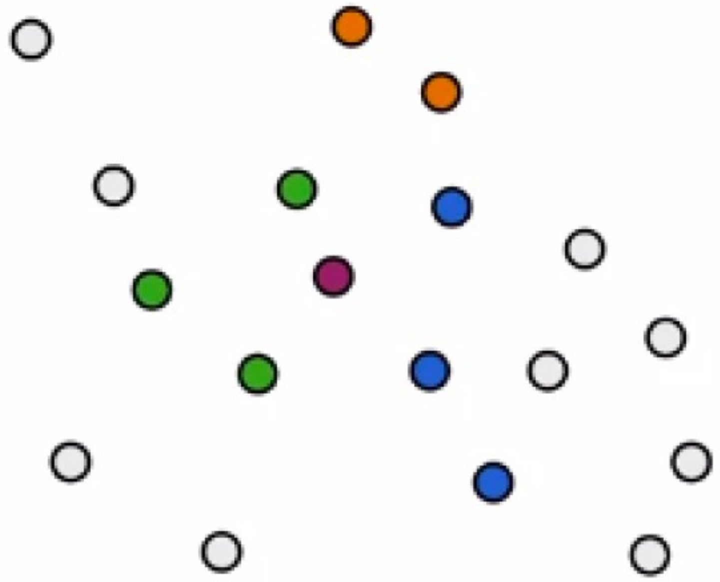
Luego, volvemos a ejecutar el lado del plano con otro plano aleatorio y el algoritmo me da los puntos azules

Múltiples conjuntos de planos aleatorios



La tercera vez, obtuvimos los puntos naranjas

Múltiples conjuntos de planos aleatorios



Si seguimos realizando este proceso, es muy probable que obtengamos todos los vecinos cercanos

Vecinos cercanos aproximados

Hacer un conjunto de planos aleatorios

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

Hacer un conjunto de planos aleatorios

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

```
def side_of_plane_matrix(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    return sign_of_dot_product

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

```
array([[1.]
       [1.]
       [1.]])
```

Representación de documentos

I love learning!	[?, ?, ?]
I	[1, 0, 1]
love	[-1, 0, 1]
learning	[1, 0, 1]

Representación de documentos

I love learning!	[?, ?, ?]
I	[1, 0, 1]
	+
love	[-1, 0, 1]
	+
learning	[1, 0, 1]
	=
I love learning!	[1, 0, 3]