Microsoft

LIVE

Python + IA

Microsoft

LIVE

Python + IA

- 🧠 3/11: LLMs
- ↖ 3/13: Vector embeddings
- 🔍 3/18: RAG
- 🖼️ 3/19: Modelos de Vision
- 🛠️ 3/25: Salidas estructuradas
- 🛡️ 3/27: Calidad y Seguridad

Regístrate @ aka.ms/PythonIA/series

Microsoft

Python + IA

↖ Embeddings vectoriales

aka.ms/pythonia/embeddings/diapositivas

Gwyneth Peña-Siguenza

Python Cloud Advocate

aka.ms/madebygps

Hoy cubriremos...

- ¿Qué son los embeddings vectoriales?
- Espacio de similitud vectorial
- Búsqueda vectorial
- Métricas de distancia vectorial
- Cuantización vectorial
- Reducción de dimensionalidad

aka.ms/pythonia/embeddings/diapositivas

Embeddings vectoriales 101

Embeddings vectoriales

Un *embedding* codifica una entrada como una lista de números de punto flotante.
"perro" → [0.017198, -0.007493, -0.057982, ...]
Diferentes modelos de *embedding* generan diferentes *embeddings*, con diferentes longitudes.

Embedding modelo	Codifica	MTEB Avg.	Longitud
word2vec	Palabras		300
Sbert (Sentence-Transformers)	texto (hasta ~400 palabras)		768
OpenAI text-embedding-ada-002	texto (hasta 8191 tokens)	61.0%	1536
OpenAI text-embedding-3-small	texto (hasta 8191 tokens)	62.3%	512, 1536
OpenAI text-embedding-3-large	texto (hasta 8191 tokens)	64.6%	256, 1024, 3072
Azure AI Vision	imágenes o texto		1024

Genera un embedding con OpenAI SDK

Usa el OpenAI SDK con OpenAI.com, Azure, Ollama, o GitHub Models:

```
openai_client = openai.OpenAI(
  base_url="https://models.inference.ai.azure.com",
  api_key=os.environ["GITHUB_TOKEN"]
)
```

Genera embeddings para una o mas entradas:

```
embeddings_response = openai_client.embeddings.create(
  model ="text-embedding-3-small",
  input="hola mundo",
  dimensions=1536,
)
print(embeddings_response.data[0].embedding)
```

Notebook: generate_embedding.ipynb

Los embeddings de vectores varían entre modelos

"queen"

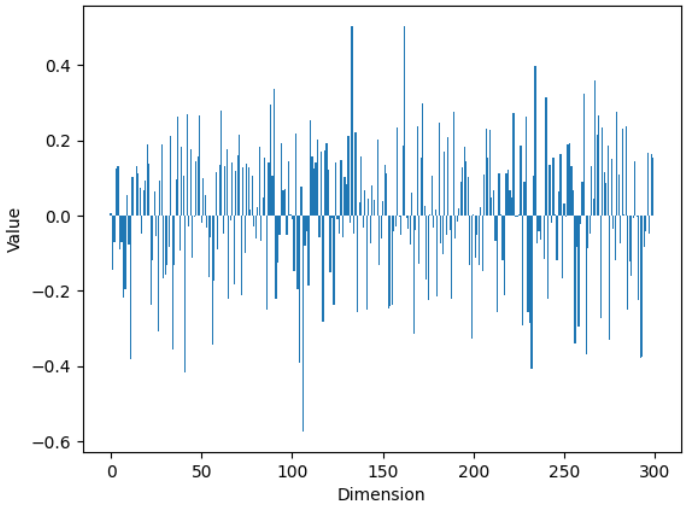
"queen"

"queen"

word2vec-google-news-300

300 dimensiones

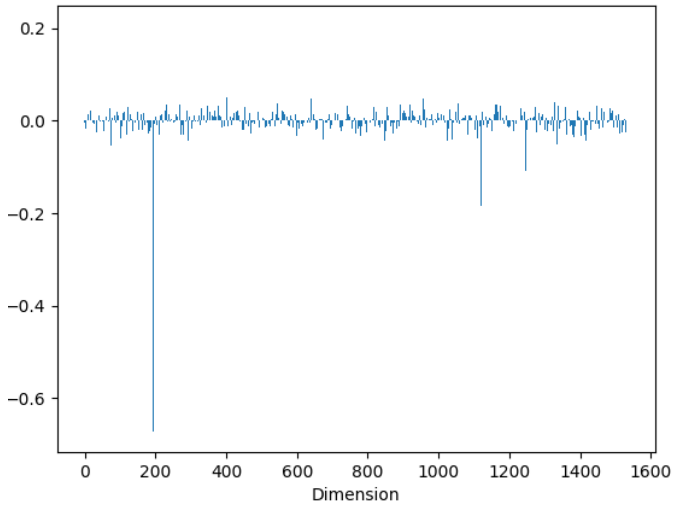
```
[0.0052490234375,
-0.1435546875,
-0.0693359375,
0.12353515625, ...]
```



text-embedding-ada-002

1536 dimensiones

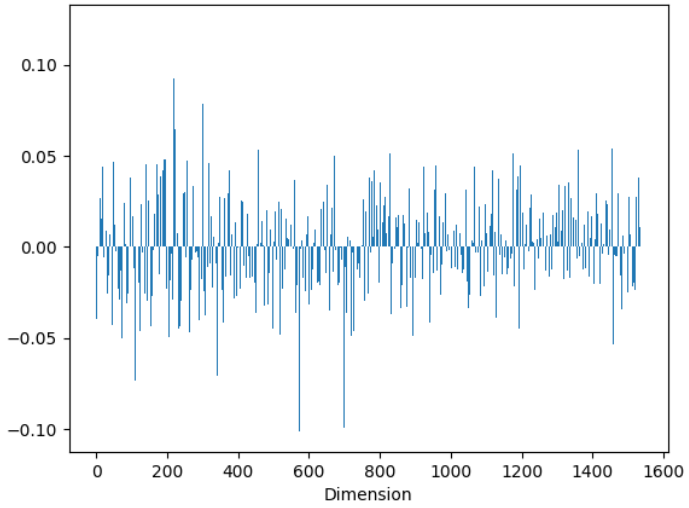
```
[-0.00449855113402009,
-0.006737332791090012,
-0.002418933203443885,
-0.018148120492696762, ...]
```



text-embedding-3-small

1536 dimensiones

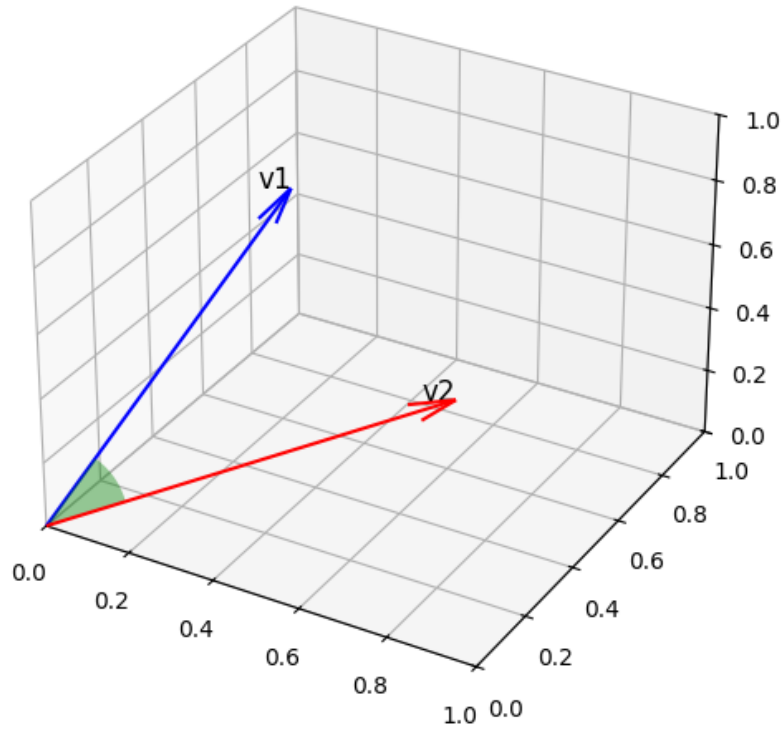
```
[0.04379640519618988,
-0.03982372209429741,
0.044741131365299225,
0.02169230207800865, ...]
```



Similiutud de vector

Calculamos embeddings para poder calcular la similitud entre las entradas. La medida de distancia más común es la similitud del coseno (cosine similiarity).

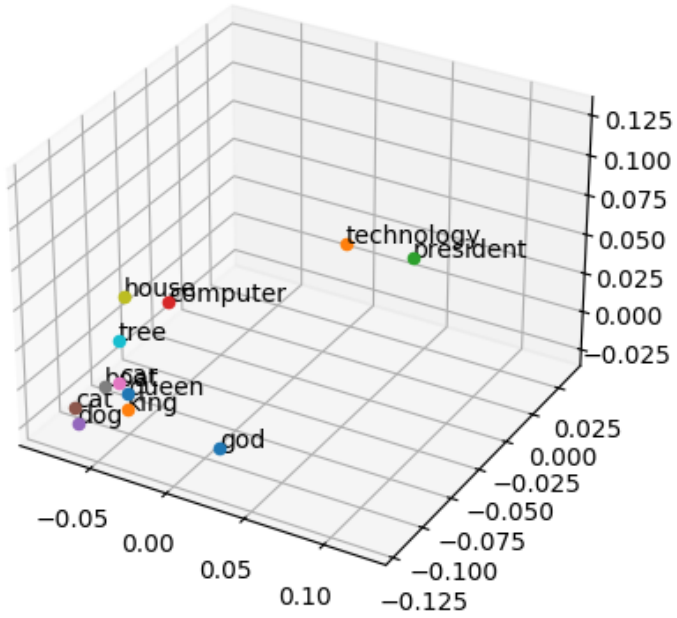
```
def cosine_similarity(v1, v2):

  dot_product = sum(
    [a * b for a, b in zip(v1, v2)])

  magnitude = (
    sum([a**2 for a in v1]) *
    sum([a**2 for a in v2])) ** 0.5

  return dot_product / magnitude
```



Notebook: [similarity.ipynb]

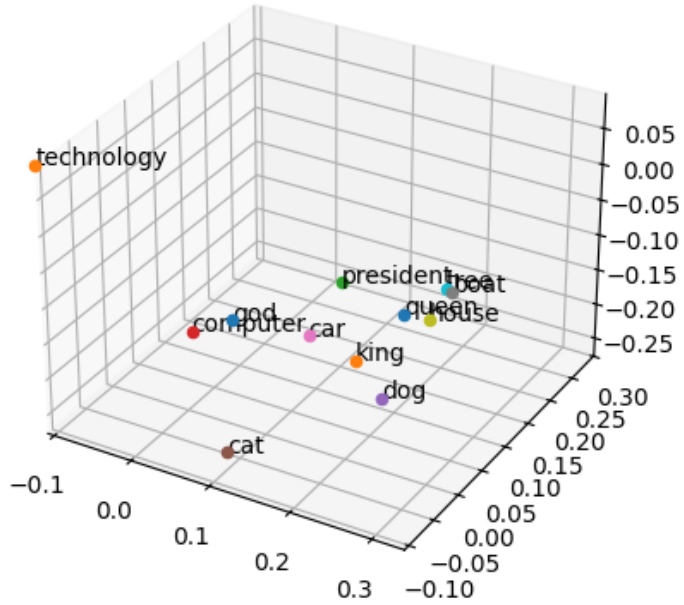
El espacio de similitud varía entre modelos

text-embedding-ada-002



palabra	coseno
dog	1.0000
animal	0.8855
god	0.8660
cat	0.8635
fish	0.8566
bird	0.8555
diet	0.8530
horse	0.8521
drug	0.8506
gun	0.8494

text-embedding-3-large (256)

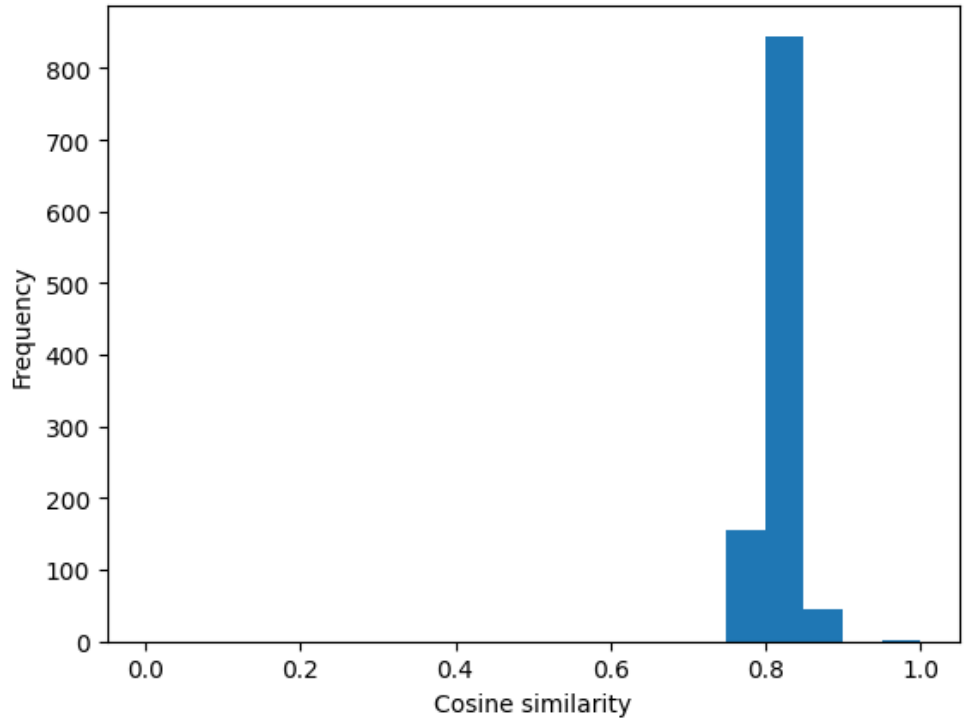


palabra	coseno
dog	1.0000
animal	0.6619
cat	0.6502
car	0.6185
horse	0.5927
boat	0.5737
dad	0.5654
post	0.5440
girl	0.5431
man	0.5410

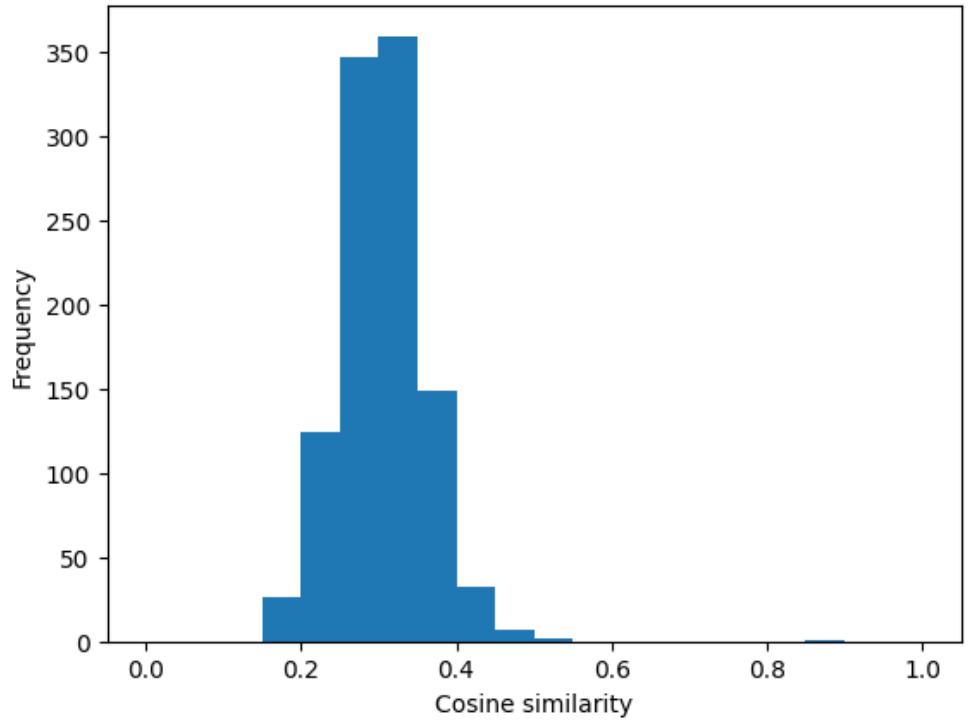
Rangos de Valores de similitud entre modelos

Similitud del coseno de ”perro" con 500 palabras más en dos modelos.

text-embedding-ada-002



text-embedding-3-large (256)



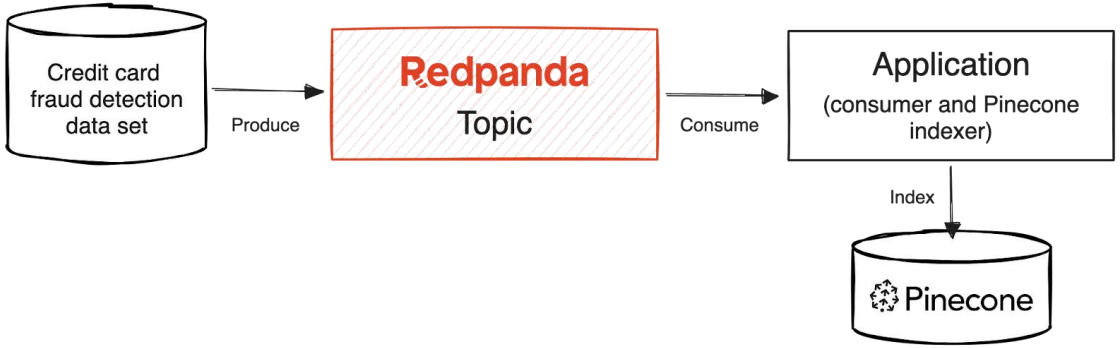
Usos empresariales de la similitud de vectores

Sistema de recomendación:

out_recipename	out_similarityscore
Apple Pie by Grandma Ople	0
Easy Apple Pie	0.05137232
Grandma's Iron Skillet Apple Pie	0.054287136

https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-recommendation-system

Detección de fraude:

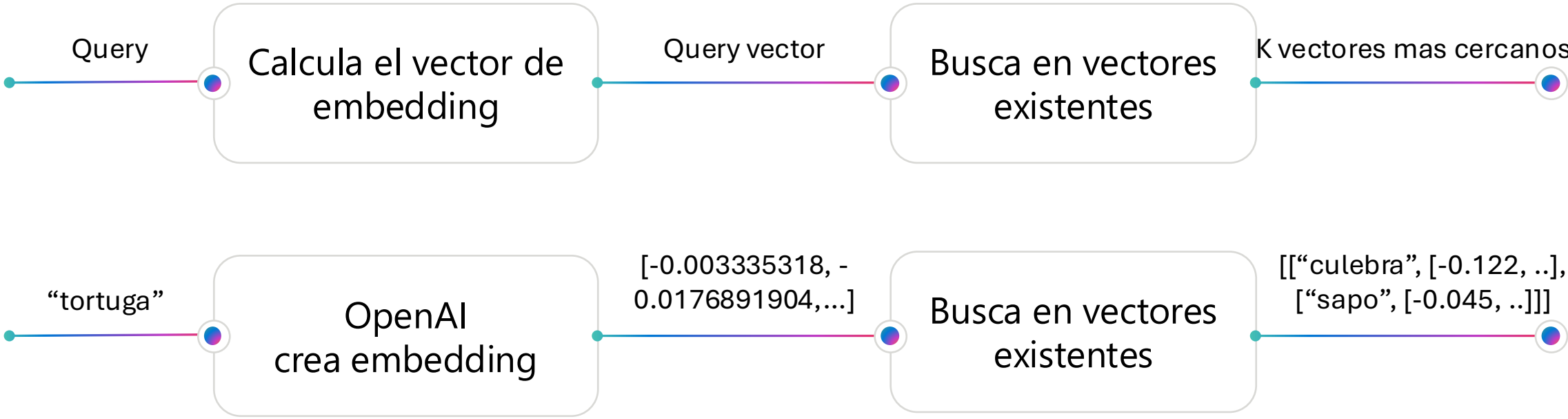


https://www.redpanda.com/blog/fraud-detection-pipeline-redpanda-pinecone

Búsqueda vectorial

Búsqueda vectorial

1. Calcula el vector de embedding para la consulta
2. Encuentra los K vectores más cercanos al vector de la consulta
   Busca de manera exhaustiva o utilizando aproximaciones



Búsqueda exhaustiva de vectores en Python

Una búsqueda exhaustiva revisa cada vector para encontrar al más cercano.

```
def exhaustive_search(query_vector, vectors):

  similarities = []
  for title, vector in vectors.items():
    similarity = cosine_similarity(query_vector, vector)
    similarities.append((title, similarity))

  similarities.sort(key=lambda x: x[1], reverse=True)

  return similarities
```

Notebook: search.ipynb

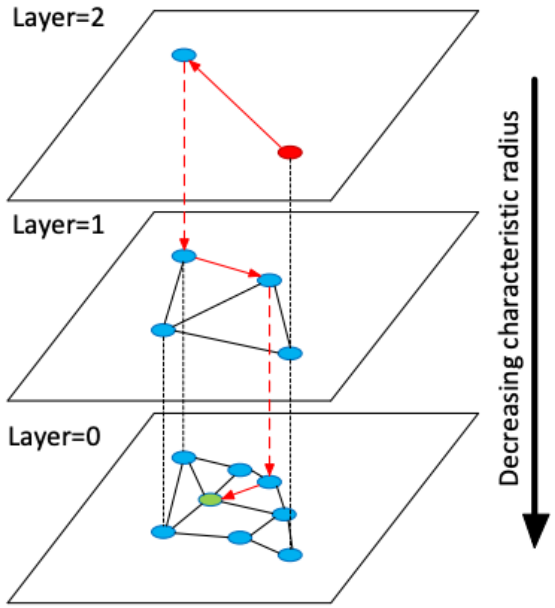
Búsqueda ANN (Approximate Nearest Neighbor)

Existen varios algoritmos de búsqueda ANN que pueden acelerar el tiempo de búsqueda.

Algoritmo	**Python librería**	**Ejemplos de bases de datos soportadas**
HNSW	hnswlib	PostgreSQL pgvector extension Azure AI Search Chromadb Weaviate
DiskANN	diskannpy	Cosmos DB
IVFFlat	faiss	PostgreSQL pgvector extension
Faiss	faiss	None, in-memory index only

HNSW: Hierarchical Navegable Small Worlds

El algoritmo HNSW es ideal para situaciones en las que tu índice pueda que se actualice con frecuencia, y su rendimiento escala de forma logarítmica incluso con índices de gran tamaño.

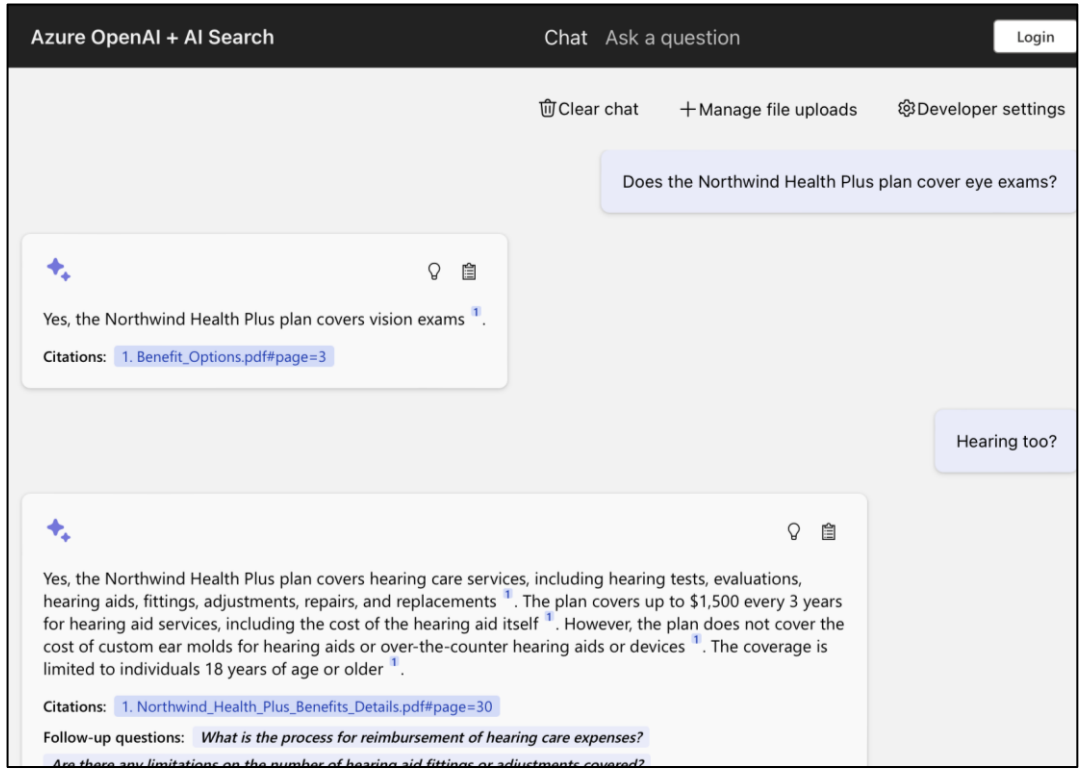



```
import hnswlib

p = hnswlib.Index(space='cosine', dim=1536)
p.init_index(
  max_elements=len(movies),
  ef_construction=200,
  M=16)

vectors = list(movies.values())
ids = list([i for i in range(len(vectors))])
p.add_items(vectors, ids)

p.set_ef(50)
```

HNSW research paper:
https://github.com/nmslib/hnswlib

Uso empresarial: Recuperación Aumentada con Generación (RAG)

La búsqueda vectorial puede mejorar considerablemente la fase de recuperación en RAG.



Azure OpenAI +
Azure AI Search +
Azure AI Vision +
Azure App Service +

Código:
aka.ms/ragchat

Demo:
aka.ms/ragchat/demo

Únete a la próxima transmisión sobre RAG el 18/3 aka.ms/PythonIA/series

Métricas de distancia vectorial

Métricas de distancia vectorial

Cuatro métricas comunes para medir la distancia entre dos vectores son:

1. Distancia Euclidiana
2. Distancia Manhattan
3. Producto Punto (escalar)
4. Distancia del coseno

La métrica que seleccionemos puede depender de si los vectores son de tipo **unit vectors.**

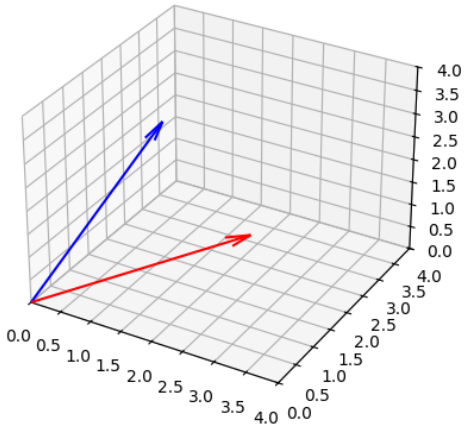
Notebook: distance_metrics.ipynb

Unit vectors

Un vector unitario es un vector cuya **magnitud** es 1.

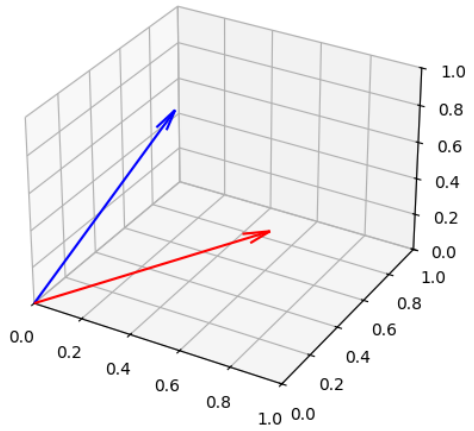
```
def magnitude(vector):
  return sum([a**2 for a in vector]) ** 0.5
```

Dos vectores con la misma magnitud de 3.7416573867739413:




```
[1, 2, 3]
[3, 1, 2]
```

Después de la normalización, dos vectores tendrán una magnitud de 1.




```
[0.26726124 0.53452248 0.80178373]
[0.80178373 0.26726124 0.53452248]
```

Distancia euclidiana

La distancia euclidiana es la distancia en línea recta entre dos puntos en el espacio euclidiano.

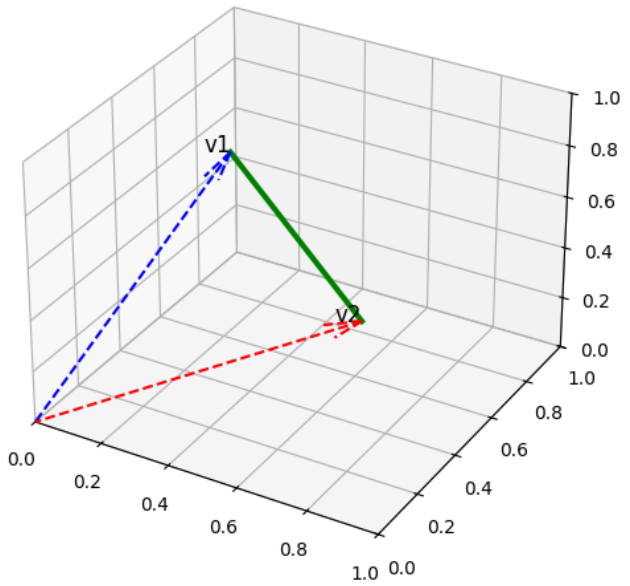
```
def euclidean(v1, v2):
  return magnitude(v1 - v2)
```

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

```
euclidean(
  [0.26726124 0.53452248 0.80178373],
  [0.80178373 0.26726124 0.53452248]
)
```

↓

```
0.655
```



Distancia de Manhattan

Es la distancia "taxicab" entre dos puntos en el espacio euclidiano.

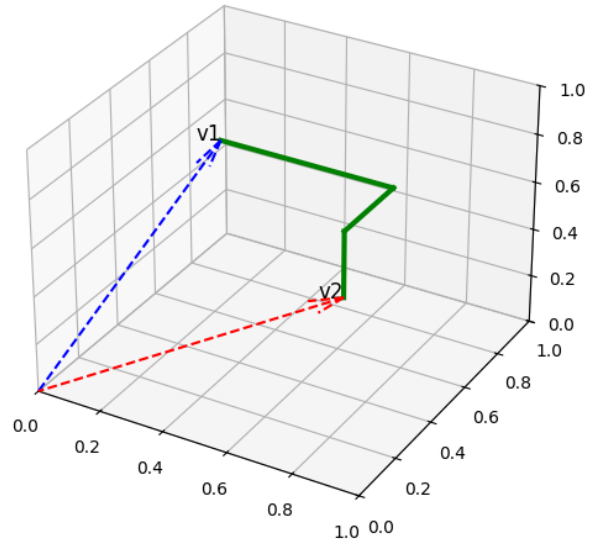
```
def manhattan(v1, v2):
  return sum(abs(a - b)
    for a, b in zip(v1, v2))
```

$$\text{Manhattan distance} = \sum_{i=1}^{n} |x_i - y_i|$$

```
manhattan(
  [0.26726124 0.53452248 0.80178373],
  [0.80178373 0.26726124 0.53452248]
)
```

↓

```
1.07
```



Producto escalar (o producto punto)

La suma de los productos de los elementos correspondientes de los vectores.

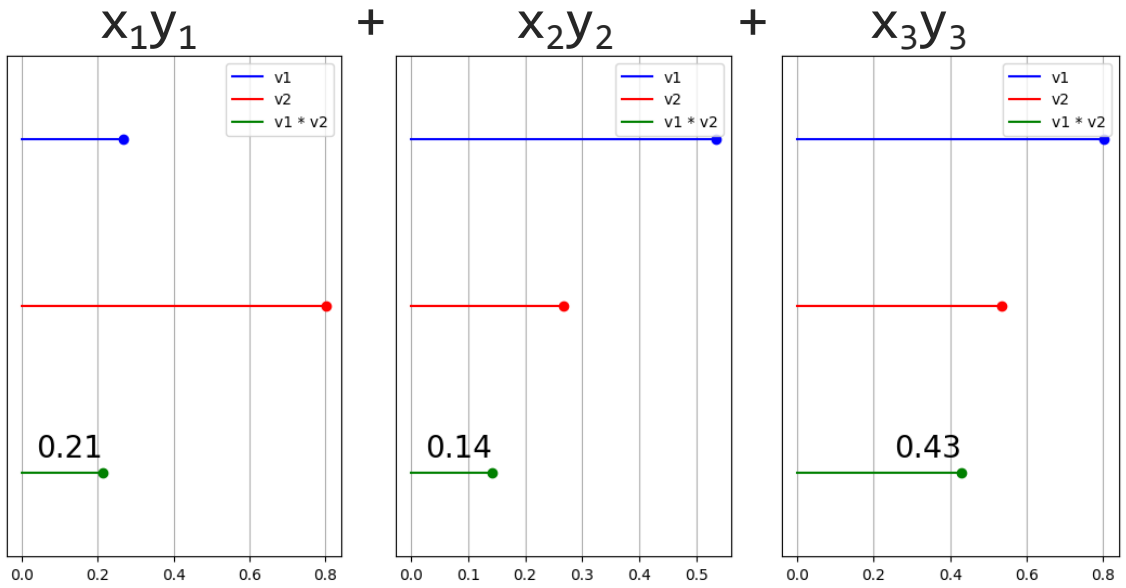
```
def dot_product(v1, v2):
  return sum(a * b
    for a, b in zip(v1, v2))
```

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + \ldots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

```
dot_product(
  [0.26726124 0.53452248 0.80178373],
  [0.80178373 0.26726124 0.53452248]
)
```

↓

0.786



Distancia del coseno

El **complemento** del coseno del ángulo entre dos vectores en el espacio euclidiano.

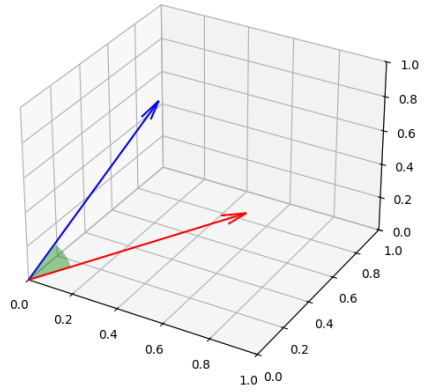
```
def cosine_similarity(v1, v2):
    return dot_product(v1, v2) /
      (magnitude(v1) * magnitude(v2))

def cosine_distance(v1, v2):
  return 1 - cosine_similarity(v1, v2)
```

$$\text{Cosine similarity} = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

$$\text{Cosine distance} = (1 - \text{Cosine similarity})$$

```
cosine_distance(
  [0.26726124 0.53452248 0.80178373],
  [0.80178373 0.26726124 0.53452248]
)
```

0.214



Distancia del coseno vs. producto escalar

Para vectores unitarios, la similitud del coseno es igual al producto escalar.

```
>> cosine_similarity(v1, v2) == dot_product(v1, v2)

True

>>> 1 - cosine_distance(v1, v2) == dot_product(v1, v2)

True
```

$$\text{Cosine similarity} = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

$$\text{Cosine distance} = (1 - \text{Cosine similarity})$$

En algunas bases de datos vectoriales, el operador de producto punto será ligeramente más rápido que los operadores de distancia del coseno, ya que no necesita calcular la magnitud.

Si tus embeddings son vectores unitarios, considera usar el producto punto como métrica. Los modelos de embedding de OpenAI actualmente generan únicamente vectores unitarios.

Cuantización vectorial

Cuantización vectorial

La mayoría de los embeddings vectoriales se almacenan como números de punto flotante (64 bits en Python). Podemos utilizar la cuantización para reducir el tamaño de los embeddings.

- **Cuantización escalar:** reduce cada número a un entero

```
[0.03265173360705376,
0.01370371412485838,
-0.01774894446134567 3,...]
```

→ `[53, 40, 20, ...]`

- **Cuantización binaria:** reduce cada número a un solo bit

```
[0.03265173360705376,
0.01370371412485838,
-0.01774894446134567 3,...]
```

→ `[1, 1, 0, ...]`

Notebook: quantization.ipynb

Cuantización escalar: El proceso

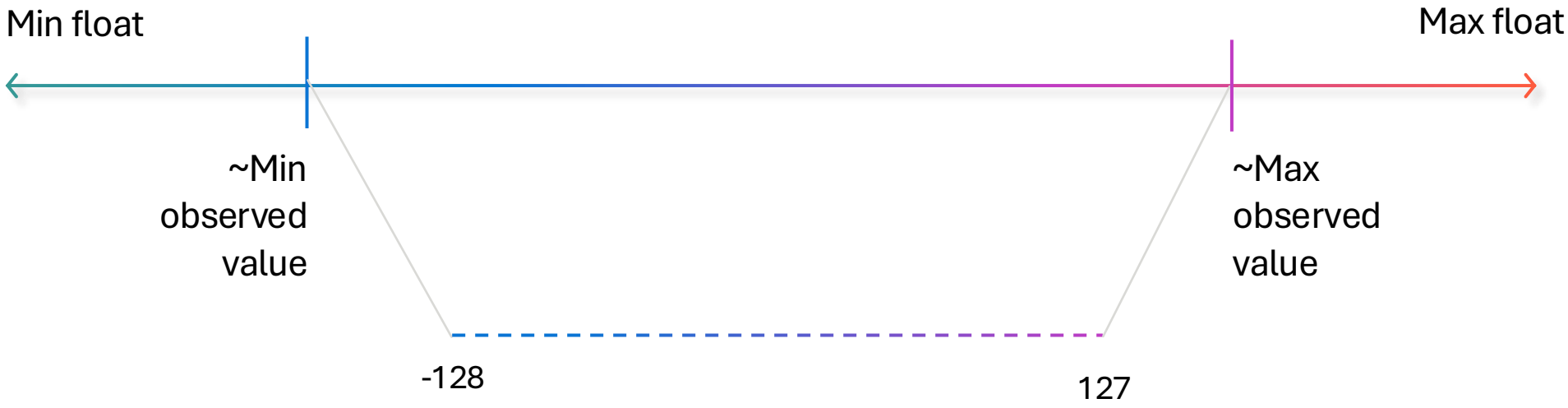
float32

```
[0.03265173360705376, 0.01370371412485838, ...]
[-0.00786194484680891, -0.01898514106869697 6, ...]
[-0.00390561786480247 97, 0.01903911307454109 2, ...]
```

int8

```
[53, 40,...]
[27, 19, ...]
[29, 44, ...]
```

1. Calcular el mínimo y máximo de todos los embeddings
2. Normalizar los valores de cada embedding al rango [0, 1]
3. Mapear los valores normalizados en intervalos enteros de -128 a +127



Cuantización escalar: Antes y Después

"Moana"

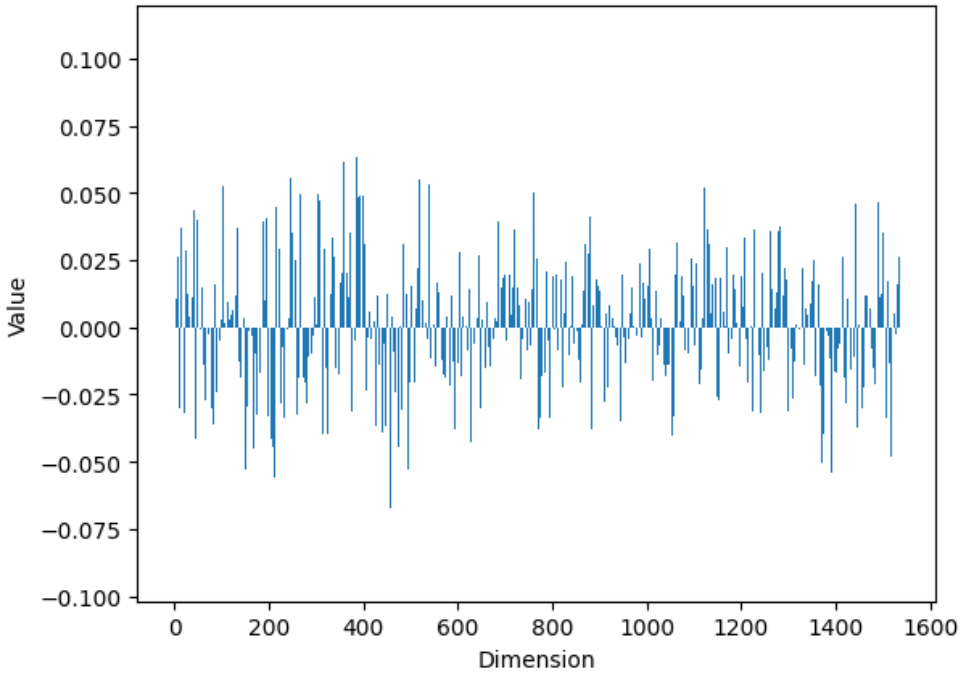
float32

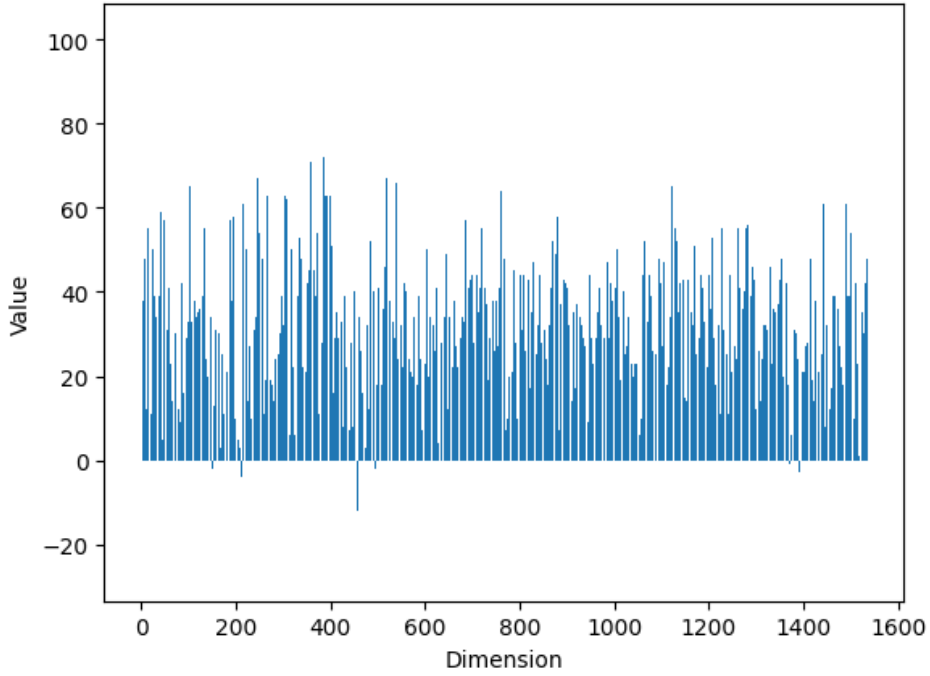
```
[0.03265173360705376,
0.01370371412485838,
-0.01774894446134567 3,...]
```

quantization

int8

[53, 40, 20, ...]





Cuantización escalar: Efectos en la similitud

float32

```
[0.03265173360705, 0.013703...]
[-0.00786194484680891, -0.0189...]
[-0.0039056178648024797, 0.0190...]
```

int8

```
[53, 40,...]
[27, 19, ...]
[29, 44, ...]
```

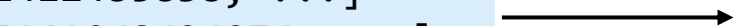
movie	similarity
Moana	1.000000
Mulan	0.546800
Lilo & Stitch	0.502114
The Little Mermaid	0.498209
Big Hero 6	0.491800
Monsters University	0.484857
The Princess and the Frog	0.471984
Finding Dory	0.471386
Maleficent	0.461029
Ice Princess	0.457817

movie	similarity
Moana	1.000000
Mulan	0.903532
The Little Mermaid	0.894227
Lilo & Stitch	0.893718
Big Hero 6	0.890959
Monsters University	0.890915
The Princess and the Frog	0.889009
Finding Dory	0.888350
Ice Princess	0.885539
Maleficent	0.885364

Cuantización binaria: El proceso

float32

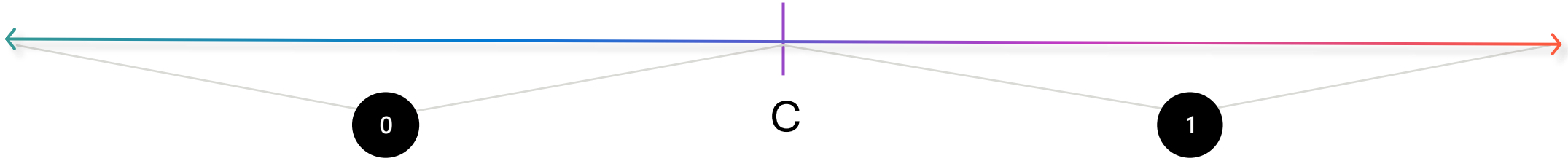
```
[0.03265173360705376, 0.01370371412485838, ...]
[-0.00786194484680891, -0.01898514106869697 6, ...]
[-0.00390561786480247 97, 0.01903911307454109 2, ...]
```



bit

```
[1, 1,...]
[0, 0, ...]
[0, 1, ...]
```

1. Escoge un centro C basándote en el promedio, una muestra o conocimientos previos.
2. Si el valor es mayor o igual a C, mapea a 1; de lo contrario, mapea a 0.



Cuantización binaria: Antes y después

"Moana"

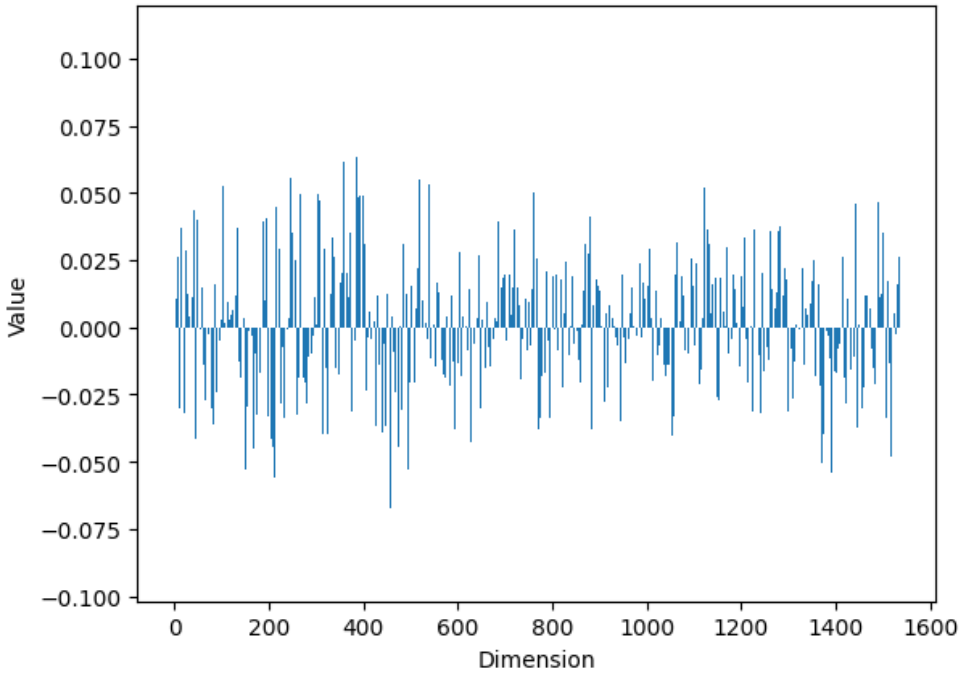
float32

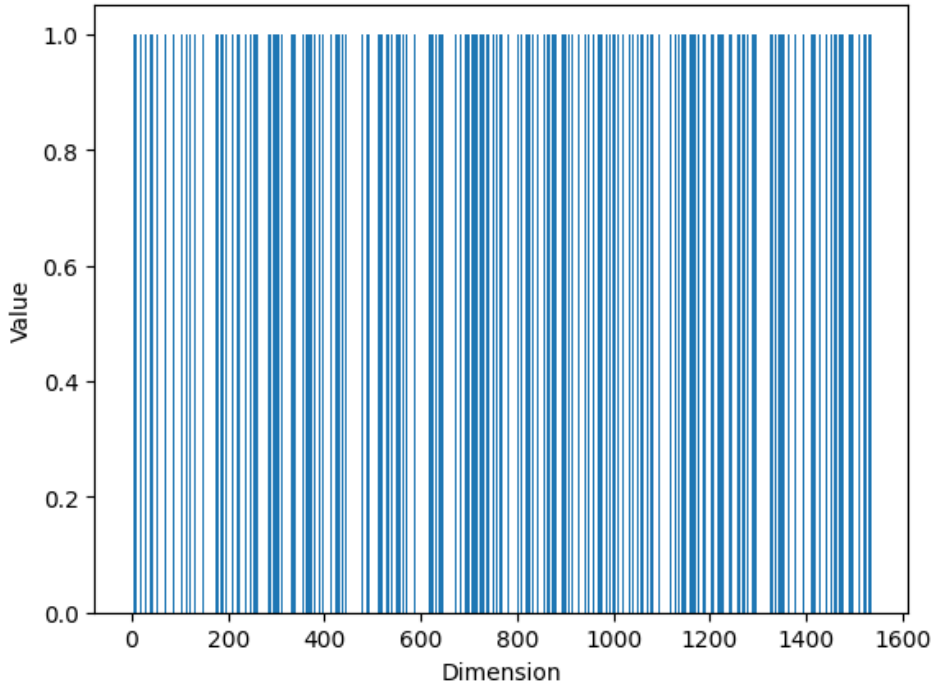
```
[0.03265173360705376,
0.01370371412485838,
-0.01774894446134567,...]
```

quantization

bit

[1, 1, 0, ...]





Cuantización binaria: Efectos sobre la similitud

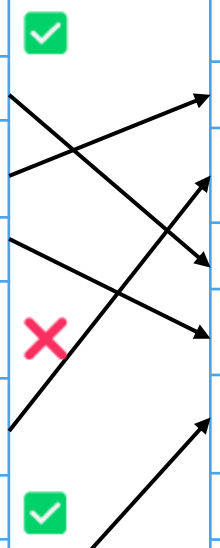
float32

```
[0.03265173360705, 0.013703...]
[-0.00786194484680891, -0.0189...]
[-0.0039056178648024797, 0.0190...]
```

bit

```
[1, 1,...]
[0, 0, ...]
[0, 1, ...]
```

movie	similarity
Moana	1.000000
Mulan	0.546800
Lilo & Stitch	0.502114
The Little Mermaid	0.498209
Big Hero 6	0.491800
Monsters University	0.484857
The Princess and the Frog	0.471984
Finding Dory	0.471386
Maleficent	0.461029
Ice Princess	0.457817



movie	similarity
Moana	1.000000
Mulan	0.686634
The Little Mermaid	0.666260
The Princess and the Frog	0.659825
Lilo & Stitch	0.657599
Big Hero 6	0.655869
Ice Princess	0.648046
Finding Dory	0.643830
The Lion King	0.643088
Maleficent	0.642270

Cuantización: Efectos en el tamaño de almacenamiento

	float32	int8	bit

	`[0.03265173360705,...]`
	`[-0.00786194484680891,...]`
	`[-0.00390561786480247,...]`
		`[53, 40,...]`
		`[27, 19, ...]`
		`[29, 44, ...]`
			`[1, 1,...]`
			`[0, 0, ...]`
			`[0, 1, ...]`

Python built-in number type	12728	12728	12728
numpy typed arrays	12400	1648	1648

Las bases de datos que soportan almacenamiento vectorial a menudo pueden ahorrar más espacio utilizando bits, mediante técnicas como el empaquetamiento de bits.

Cuantización: Efectos en el tamaño del índice en AI Search

Azure AI Search admite la cuantización como una forma de reducir el espacio necesario para el almacenamiento de vectores.

float32

```
[0.03265173360705,...]
[-0.007861944846808 91,...]
[-0.003905617864802 47,...]
```

int8

```
[53, 40,...]
[27, 19, ...]
[29, 44, ...]
```

bit

```
[1, 1,...]
[0, 0, ...]
[0, 1, ...]
```

Vector index size (MB)	1177.12	298.519	41.8636
		74.64% reducción!	96.44% reducción!

AI Search cuenta con dos ubicaciones de almacenamiento para los vectores: el índice HNSW utilizado para la búsqueda y el almacenamiento real de datos. Las estadísticas anteriores corresponden al tamaño del índice.

Aprende más en la RAG time serie: https://aka.ms/rag-time/journey3

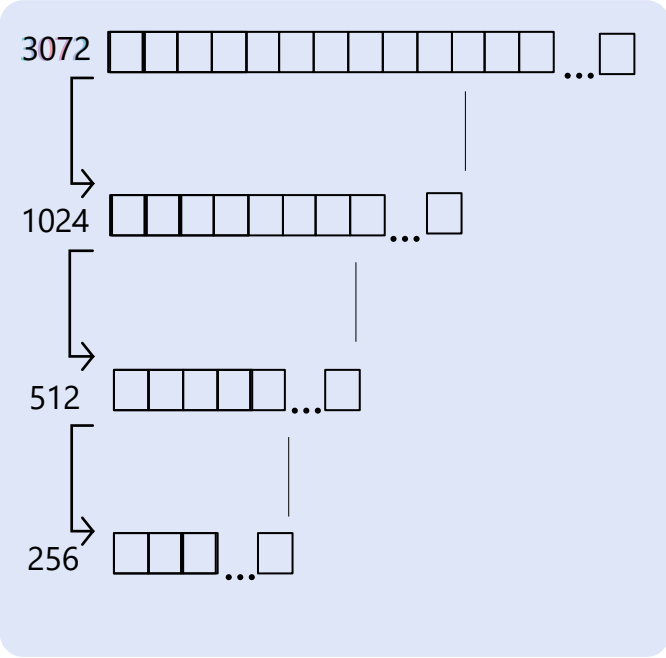
Reducción de dimensiones MRL

MRL: Matryoshka Representation Learning

MRL es una técnica que te permite reducir las dimensiones de un vector, mientras se conserva gran parte de la representación semántica original.

El modelo OpenAI text-embedding-3-large tiene dimensiones predeterminadas de **3072**,
pero puede truncarse hasta **256**.

⚠️ ¡Solo algunos modelos soportan MRL!



Puedes truncar:

- al generar los embeddings
- o al almacenarlos en la base de datos (si se admite)

Reducción de dimensiones con OpenAI SDK

Especifica las dimensiones al generar un embedding:

```
embeddings_response = openai_client.embeddings.create(
  model ="text-embedding-3-small",
  input="hello world",
  dimensions=256
)

print(embeddings_response.data[0].embedding)
```

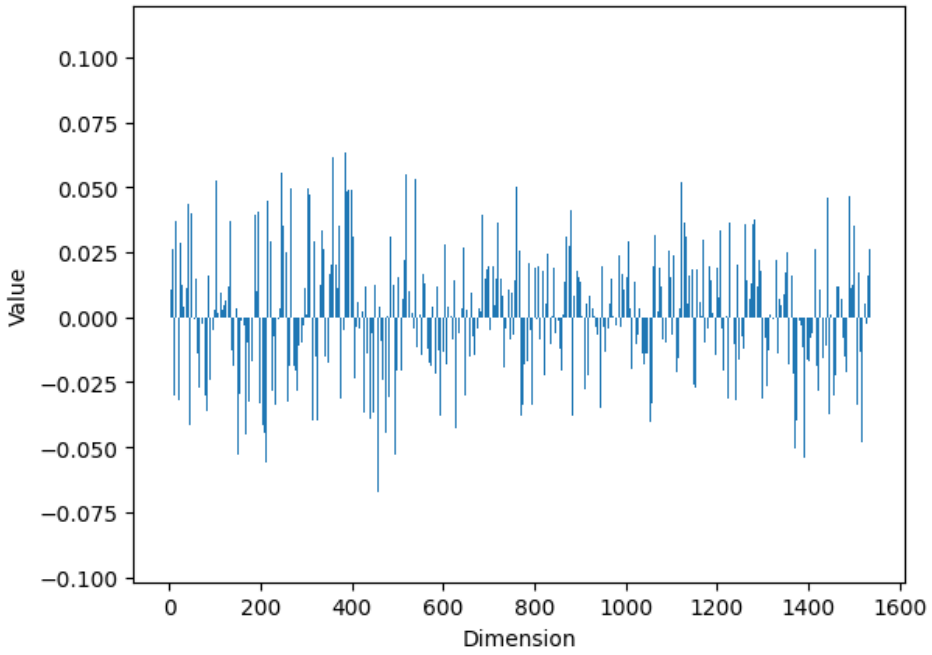
Notebook: dimension_reduction.ipynb

Reducción de dimensiones: Antes y después

"Moana"

dimensions=1536

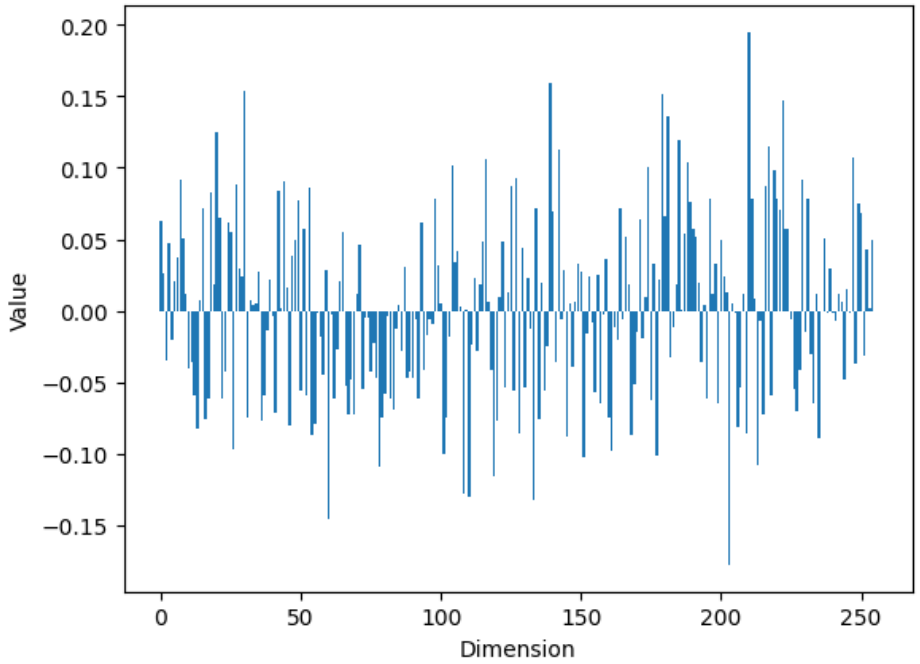
```
[0.032651733607053 76,
0.013703714124858 38,
-0.017748944461345673,...]
```



"Moana"

dimensions=256

```
[0.06316128373146057,
0.026508368551731 11,
-0.034333433955907 82,...]
```



Reducción de dimensiones: Efectos sobre la similitud

dimensions=1536

```
[0.0326517336070537 6,
0.0137037141248583 8,
-0.0177489444613456 73,...]
```

dimensions=256

```
[0.0326517336070537 6,
0.0137037141248583 8,
-0.0177489444613456 73,...]
```

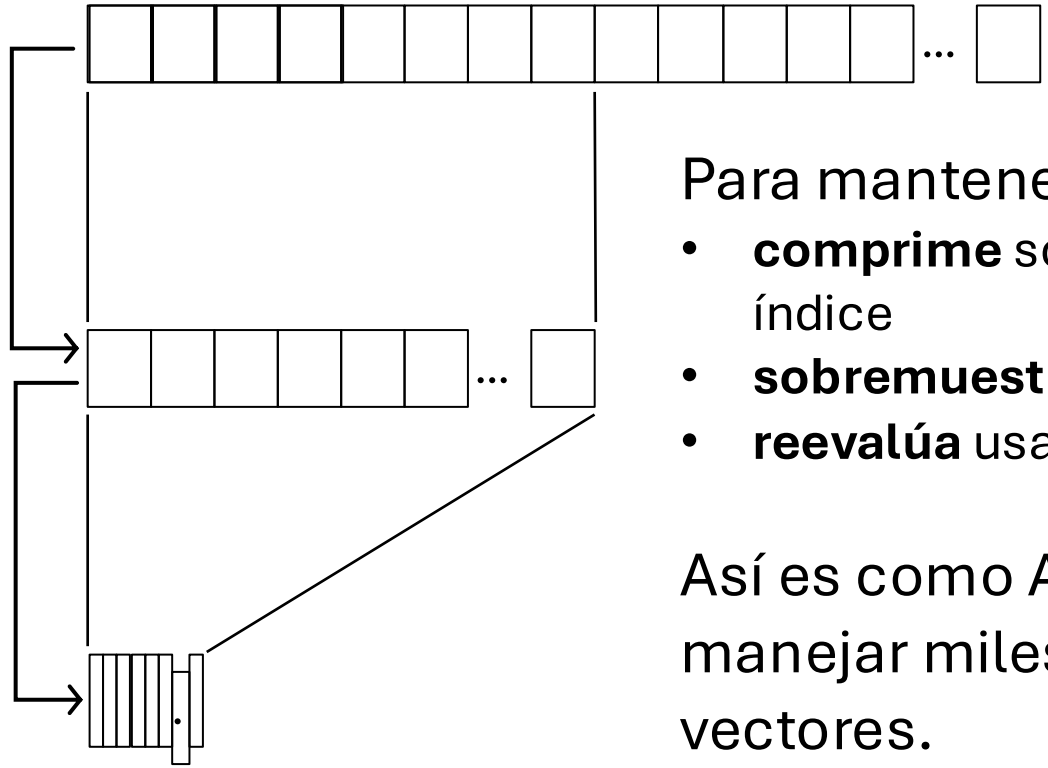
movie	similarity
Moana	1.000000
Mulan	0.546800
Lilo & Stitch	0.502114
The Little Mermaid	0.498209
Big Hero 6	0.491800
Monsters University	0.484857
The Princess and the Frog	0.471984
Finding Dory	0.471386
Maleficent	0.461029
Ice Princess	0.457817

movie	similarity
Moana	1.000000
The Little Mermaid	0.587367
Mulan	0.583428
Lilo & Stitch	0.575990
Big Hero 6	0.574590
The Princess and the Frog	0.568726
Finding Dory	0.549391
The Lion King	0.521125
Tangled	0.513131
Maleficent	0.511412

Reducción de dimensiones más cuantización

¡Para lograr una compresión máxima de vectores, combina ambas técnicas!

1. MRL Dimension Reduction
2. Scalar or Binary Quantization



Para mantener alta precisión:

- **comprime** solo los vectores en el índice
- **sobremuestrea** al recuperar
- **reevalúa** usando los originales.

Así es como Azure AI Search puede manejar miles de millones de vectores.

Learn more in RAG time series: https://aka.ms/rag-time/journey3

Aprende mas acerca de vector embeddings

Vector embeddings 101

- Embedding projector
- Why are Cosine Similarities of Text embeddings almost always positive?
- Expected Angular Differences in Embedding Random Text?
- Embeddings: What they are and why they matter

ANN algorithms

- HNSW tutorial
- Video: HNSW for Vector Search Explained

Distance metrics:

- Two Forms of the Dot Product
- Is Cosine-Similarity of Embeddings Really About Similarity?

Quantization:

- Scalar quantization 101
- Product quantization 101
- Binary and scalar quantization

MRL dimension reduction:

- Unboxing Nomic Embed v1.5: Resizable Production Embeddings with MRL
- MRL from the Ground Up

Próximos pasos

¡Únete a los próximos streams! →

Aka.ms/agenthack

Ven a las horas de oficina los Lunes en Discord:
aka.ms/pythonia/ho

Obtén más recursos de Python AI
aka.ms/thesource/Python_AI

🧠 3/11: LLMs

↖ 3/13: Vector embeddings

🔍 3/18: RAG

🖼️ 3/19: Models de Vision

🛠️ 3/25: Salidas estructuradas

🛡️ 3/27: Calidad y Seguirdad

Regístrate @ aka.ms/PythonIA/series

Gracias

aka.ms/pythonia/embeddings/diapositivas

mis redes: aka.ms/madebygps