

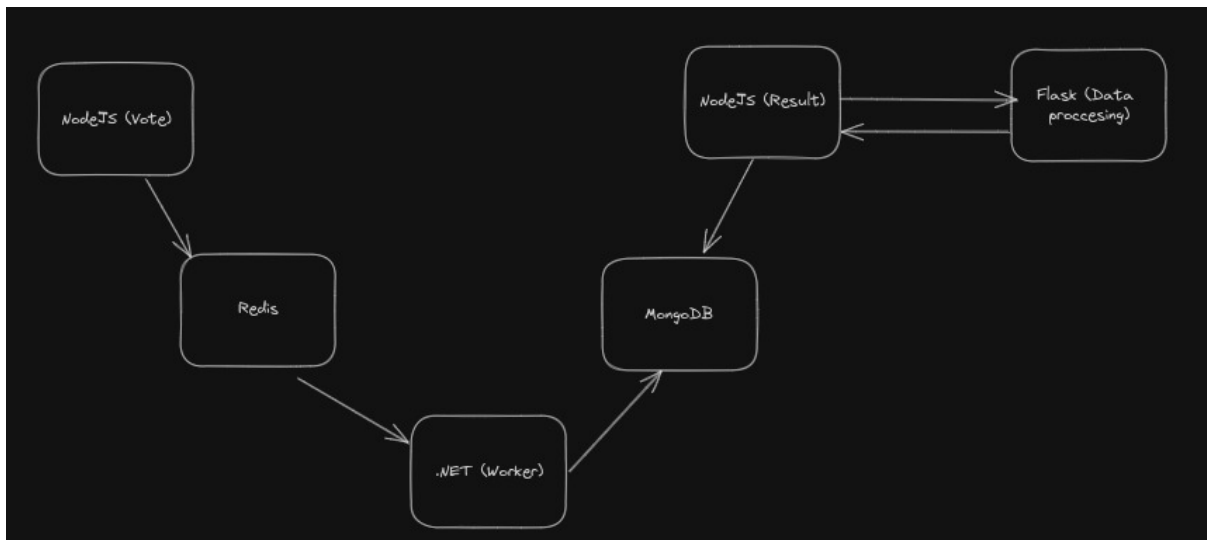
Desarrollo de Aplicaciones Empresariales Avanzado e Inteligencia de negocios

Examen

Alumno(s):	<i>-Escobar Mendoza Juan</i> <i>-Molina Quispe Junior Alessandro</i> <i>-Nina Choquehuanca Joaquín Gonzalo</i> <i>-Paucar Rodrigo Luis Franco</i> <i>-Valdivia Mamani Steven Brandon</i> <i>-Hinojosa Amudio Eduardo Johel</i> <i>-Navarro Sacramento Sebastian Estefano</i>				Nota	
Grupo:	<i>C24 A y B</i>		Ciclo: VI			
Criterio de Evaluación		Excelente (4pts)	Bueno (3pts)	Requiere mejora (2pts)	No acept. (0pts)	Puntaje Logrado
Identifica los conceptos de predicción de soporte vectorial						
Identifica las variables discretas						
Identifica las variables continuas						
Conoce el ámbito del algoritmo de máquinas de soporte vectorial						
Implementa métodos del algoritmo de máquinas de soporte vectorial						

Sistema de recomendación (50M en 7s)

Arquitectura:



Vote:

Aplicación hecha en NodeJS que permite a nuevos usuarios registrarse y brindar calificación sobre 5 películas distintas. Esta aplicación envía los datos a Redis además de enviar una señal abierta.

Redis:

Funciona como BD temporal recepcionando los datos de la aplicación Vote y emitiendo una señal abierta.

Worker:

Aplicación hecha en .NET la cual está constantemente encendida gracias a un bucle infinito. Esta app espera cualquier señal nueva abierta de Redis, y cuando pase jala los datos de Redis para guardarlo en la base de datos de MongoDB.

MongoDB:

Base de datos general de la aplicación, es el punto final de los usuarios nuevos registrados que después sus datos pueden ser manipulados por las otras 2 aplicaciones (Result y Flask).

Result:

Aplicación hecha en NodeJS la cual recoge los datos de MongoDB para mostrarlos en navegador y permitir a los usuarios calcular sus distancias frente a otros (euclidiana, manhattan, pearson y cosenos). Además posee una funcionalidad "POST" para enviar un usuario a la API Flask para calcular el vecino más cercano y película recomendada.

Flask:

API que recibe un usuario por POST y que contiene un CSV de 10 millones de datos (temporalmente). Cumple la función de comparar el usuario recibido por POST con cada

usuario del CSV para calcular su vecino más cercano y película recomendada para enviarlo en el código de respuesta de la petición POST.

Link del repositorio de GitHub: <https://github.com/EduardoHinojosa127/DAEA-Project.git>

Partes importantes:

Envío de datos a Flask:

```
app.post('/enviar-datos', async (req, res) => {
  try {
    // Obtén los datos del formulario enviado por el cliente
    const datosFormulario = req.body;
    // Buscar el usuario en la base de datos
    const db = mongoose.connection.db;
    const collection = db.collection('movie_scores');
    const movieScores = await collection.find({ _id: 0, __v: 0 }).toArray();
    const usuarioEncontrado = await collection.find({ usuario: datosFormulario.usuario }, { _id: 0, __v: 0 }).toArray();

    console.log(datosFormulario.usuario);
    console.log(usuarioEncontrado[0]);

    if (usuarioEncontrado) {

      const respuestaAPI = await axios.post('http://Flask-container:5000/procesar', usuarioEncontrado[0]);

      // Manejar la respuesta de la API según sea necesario
      console.log('Respuesta del servidor:', respuestaAPI.data);
      const respuesta = respuestaAPI.data;
      console.log(respuesta.usuario_recibido);
      res.render('index', { respuesta, movieScores });

    } else {
      console.log('Usuario no encontrado. Valor buscado:', datosFormulario.usuario);
      res.status(404).json({ error: 'Usuario no encontrado' });

      res.render('index', { respuesta: undefined });
    }
  } catch (error) {
    // Manejar los errores de la solicitud
    console.error('Error al enviar datos:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});
```

Procesamiento de datos:

```
@app.route('/procesar', methods=['POST'])
def procesar():
  try:
    user_data = request.json
    print('Usuario recibido:', user_data)

    # Eliminar la clave 'id' del usuario recibido
    user_data.pop('id', None)

    # Convertir datos a arrays NumPy para cálculos más eficientes
    user_ratings = np.array([int(user_data[f'pelicula{i}']) for i in range(1, 6)])
    data_ratings = data.iloc[:, 1:6].values.astype(int)

    # Calcular la distancia euclidiana entre el usuario enviado y todos los usuarios en el conjunto de datos
    distancias = np.linalg.norm(data_ratings - user_ratings, axis=1)

    # Encontrar el índice del usuario con la distancia euclidiana más baja
    vecino_mas_cercano_index = np.argmin(distancias)

    vecino_mas_cercano = data.iloc[vecino_mas_cercano_index].to_dict()

    # Obtener la película recomendada del vecino más cercano
    pelicula_recomendada = obtener_pelicula_recomendada(vecino_mas_cercano)

    # Preparar la respuesta
    respuesta = {
      'vecino_mas_cercano': vecino_mas_cercano,
      'pelicula_recomendada': pelicula_recomendada,
      'usuario_recibido': user_data['usuario'],
    }

    return jsonify(respuesta)

  except Exception as e:
    print('Error en la función procesar:', str(e))
    return jsonify({'error': 'Error interno'}), 500

def obtener_pelicula_recomendada(vecino):
  # Lógica para obtener la película recomendada del vecino más cercano
  # En este caso, se elige la película con la calificación más alta entre las películas 6 a 10

  # Crear una lista de tuplas (película, calificación) para las películas 6 a 10
  peliculas_calificaciones = [(f'pelicula{i}', vecino[f'pelicula{i}']) for i in range(6, 11)]

  # Encontrar la película con la calificación más alta
  pelicula_recomendada, _ = max(peliculas_calificaciones, key=lambda x: x[1])

  return pelicula_recomendada

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000, debug=True)
```

Recepción de datos de Redis y almacenamiento en MongoDB:

```
static void BuscarEnRedisYGuardarEnMongoDB(string usuario)
{
    Console.WriteLine("entro a funcion");
    // Configura la conexión a MongoDB local
    var mongoConnectionString = "mongodb://mongo:27017";
    var mongoClient = new MongoClient(mongoConnectionString);
    var database = mongoClient.GetDatabase("DAEA11");
    var collection = database.GetCollection<BsonDocument>("movie_scores");

    // Configura la conexión a Redis
    var redisConnectionString = "redis-server:6379"; // Reemplaza con la IP y el puerto correctos
    var redis = ConnectionMultiplexer.Connect(redisConnectionString);
    var redisDatabase = redis.GetDatabase();

    // Obtener valores para las tres películas
    var pelicula1 = redisDatabase.HashGet(usuario, "pelicula1");
    var pelicula2 = redisDatabase.HashGet(usuario, "pelicula2");
    var pelicula3 = redisDatabase.HashGet(usuario, "pelicula3");
    var pelicula4 = redisDatabase.HashGet(usuario, "pelicula4");
    var pelicula5 = redisDatabase.HashGet(usuario, "pelicula5");

    // Crear el filtro para buscar el documento existente por usuario
    var filter = Builders<BsonDocument>.Filter.Eq("usuario", usuario);

    // Crear el documento BSON
    var document = new BsonDocument
    {
        { "usuario", usuario },
        { "pelicula1", pelicula1.ToString() }, // Convertir a string
        { "pelicula2", pelicula2.ToString() }, // Convertir a string
        { "pelicula3", pelicula3.ToString() },
        { "pelicula4", pelicula4.ToString() },
        { "pelicula5", pelicula5.ToString() } // Convertir a string
    };

    // Reemplazar el documento existente o insertar uno nuevo
    var result = collection.ReplaceOne(filter, document, new ReplaceOptions { IsUpsert = true });

    Console.WriteLine($"Datos guardados en MongoDB para el usuario: {usuario}");

    if (result.IsAcknowledged && result.ModifiedCount > 0)
    {
        Console.WriteLine($"Registro actualizado en MongoDB para el usuario: {usuario}");
    }
}
```

Envío de datos a Redis:

```
// Ruta para manejar las valoraciones enviadas por el formulario
app.post('/valorar', (req, res) => {
    const { usuario, pelicula1, pelicula2, pelicula3, pelicula4, pelicula5 } = req.body;

    // Agrega registros de consola para imprimir el contenido del JSON
    console.log('JSON recibido:', req.body);

    // Guardar las valoraciones en Redis
    redisClient.hset(usuario, {
        'pelicula1': pelicula1,
        'pelicula2': pelicula2,
        'pelicula3': pelicula3,
        'pelicula4': pelicula4,
        'pelicula5': pelicula5
    }, (err) => {
        if (err) {
            console.error('Error al guardar las valoraciones en Redis: ${err}');
            res.status(500).send('Error interno del servidor');
        } else {
            redisClient.publish('nuevosDatos', usuario);
            res.status(200).send('Valoraciones guardadas correctamente en Redis');
        }
    });
});

// Iniciar el servidor
app.listen(port, () => {
    console.log(`Servidor iniciado en http://localhost:${port}`);
});
```

docker-compose.yml:

```
version: '3'

services:
  worker:
    build:
      context: ../worker
      dockerfile: Dockerfile
    networks:
      - my_network
    depends_on:
      - redis
      - mongo

  vote:
    build:
      context: ../
      dockerfile: Dockerfile
    networks:
      - my_network
    depends_on:
      - redis
    ports:
      - "3000:3000"

  flask:
    build:
      context: ../flask
      dockerfile: Dockerfile
      container_name: "flask-container"
    networks:
      - my_network
    ports:
      - "5000:5000"

  result:
    build:
      context: ../result
      dockerfile: Dockerfile
    networks:
      - my_network
    depends_on:
      - mongo
      - flask
    ports:
      - "3001:3001"

  redis:
    image: "redis:latest"
    container_name: "redis-server"
    networks:
      - my_network
    ports:
      - "6379:6379"

  mongo:
    image: "mongo:latest"
    container_name: "mongo"
    networks:
      - my_network
    ports:
      - "27017:27017"


networks:
  my_network:
    driver: bridge
```

Despliegue en Amazon EC2:

Instancia:

Resumen de instancia de i-0fbb528eefe00770f (server) [Información](#)


Se ha actualizado hace less than a minute

ID de la instancia
 i-0fbb528eefe00770f (server)


Dirección IPv6
-


Tipo de nombre de anfitrión
Nombre de IP: ip-172-31-58-60.ec2.internal

Responder al nombre DNS de recurso privado IPv4 (A)
-

Dirección IP asignada automáticamente
 100.26.238.150 [IP pública]


Rol de IAM
-


Dirección IPv4 pública
 100.26.238.150 [|dirección abierta](#)


Estado de la instancia
 En ejecución


Nombre DNS de IP privada (solo IPv4)
 ip-172-31-58-60.ec2.internal

Tipo de instancia
t2.large


ID de VPC
 vpc-01e0e7322b6c6ac43 [|](#)

ID de subred
 subnet-066b5e617865999dc [|](#)

Direcciones IPv4 privadas
 172.31.58.60

DNS de IPv4 pública
 ec2-100-26-238-150.compute-1.amazonaws.com [|dirección abierta](#)

Direcciones IP elásticas
-

Hallazgo de AWS Compute Optimizer
 [Suscribirse a AWS Compute Optimizer para recibir recomendaciones.](#)
[| Más información](#)

Nombre del grupo de Auto Scaling
-

Reglas de entrada de la instancia:

sgr-08448d0bc77dcffdf	SSH	TCP	22	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-088a2bd56711e79d	TCP personalizado	TCP	2377	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-05067b605d4e608f0	TCP personalizado	TCP	27017	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-000f0d4c66907c8b	TCP personalizado	TCP	3001	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-0c78a59f6cd536279	HTTP	TCP	80	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-0f6501d6c88223d9b	HTTPS	TCP	443	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-090a3f9fae984594e	TCP personalizado	TCP	6379	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-0240b809d7bb514b	TCP personalizado	TCP	5000	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-0f0dd67162e0ffac	Regla ICMP personalizada - IPv4	Repetir solicit...	N/D	Personalizada	Q		Eliminar
					0.0.0.0/0	X	
sgr-01a1e262aadda5605	TCP personalizado	TCP	3000	Personalizada	Q		Eliminar
					0.0.0.0/0	X	

Levantamiento en instancia con docker:

```
root@ip-172-31-58-60:/home/ubuntu/DAAE-Project# docker compose up -d
[+] Running 15/16
 ! mongo 9 layers [0.0.0.0/0.0.0.0] 147.9MB/224.4MB Pulling
   ✓ cbe3537751ce Download complete 0.3s
   ✓ a88d99d2ce19 Download complete 0.1s
   ✓ cdb44dc221f9 Download complete 0.1s
   ✓ 52cece2eeeb6 Download complete 0.3s
   ✓ 9484737e86c4 Download complete 0.2s
   ✓ 43ad935b79c6 Download complete 0.3s
   ✓ a3ac68e8ff6 Download complete 0.5s
   ✓ 98586617c783 Downloading [----->] 147.9MB/224.4MB 2.7s
   ✓ 3c932f959341 Download complete 0.6s
 ! redis 7 layers [0.0.0.0/0.0.0.0] 68/68 Pulling
   ✓ 1f7cc2fa4dab Download complete 2.9s
   ✓ 3c6368585bf1 Download complete 0.6s
   ✓ 3911d271d7d8 Download complete 0.7s
   ✓ ac68a9d4e21 Download complete 0.8s
   ✓ 127cd73a68a2 Download complete 1.3s
   ✓ 4f4fb788ef84 Download complete 1.6s
   ✓ f3993c1184fc Download complete 1.2s
   ✓ 3993c1184fc Download complete 1.3s

root@ip-172-31-58-60:/home/ubuntu/DAAE-Project# docker compose up -d
[+] Running 7/7
 ! Network daea-project_my_network Created
 ! Container mongo Started
 ! Container redis-server Started
 ! Container flask-container Started
 ! Container daea-project-vote-1 Started
 ! Container daea-project-worker-1 Started
 ! Container daea-project-result-1 Started
root@ip-172-31-58-60:/home/ubuntu/DAAE-Project# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED      STATUS      PORTS                               NAMES
ad9f2162e31b       daea-project-result "docker-entrypoint.s..." 4 seconds ago Up 1 second 0.0.0.0:3001->3001/tcp, :::3001->3001/tcp daea-project-result-1
c7fe9fb2b0a6       daea-project-worker "dotnet worker.dll &..." 4 seconds ago Up 1 second 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp daea-project-worker-1
39765caeca3a       daea-project-vote  "docker-entrypoint.s..." 4 seconds ago Up 2 seconds 0.0.0.0:27017->27017/tcp, :::27017->27017/tcp daea-project-vote-1
d578268a725b       mongo:latest       "docker-entrypoint.s..." 4 seconds ago Up 2 seconds 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp mongo
81a3f651a892       daea-project-flask "python app.py"          4 seconds ago Up 2 seconds 6379/tcp, 0.0.0.0:6739->6739/tcp, :::6739->6739/tcp flask-container
2bb9efab2f3f       redis:latest        "docker-entrypoint.s..." 4 seconds ago Up 3 seconds                               redis-server
root@ip-172-31-58-60:/home/ubuntu/DAAE-Project#
```

Ejecución:

Creación de nuevos usuarios:

No es seguro | 100.26.238.150:3000

sesión en Ca...

Valoración de Películas

Usuario:

Eduardo

Matrix:

87

Dodgeball:

43

WWII:

16

Pokemon:

10

Dr. House:

90

Enviar Valoraciones

No es seguro | 100.26.238.150:3000

en Ca...

Valoración de Películas

Usuario:

Luis

Matrix:

76

Dodgeball:

19

WWII:

10

Pokemon:

16

Dr. House:

77

Enviar Valoraciones

Cálculo de distancias internas:



Tabla de Resultados

Usuario	Matrix	Dodgeball	WWII	Pokemon	Dr. House
Eduardo	87	43	16	10	90
Luis	76	19	10	16	77

Calcular Distancia

Usuario 1:

Eduardo

Usuario 2:

Luis

Tipo de Distancia:

Euclidiana

Calcular Distancia

Calcular Distancia y Recomendar

Distancia euclidean entre Eduardo y Luis: 27.0740

Vecino más cercano y película recomendada para user1:



Tabla de Resultados

Usuario	Matrix	Dodgeball	WWII	Pokemon	Dr. House
Eduardo	87	43	16	10	90
Luis	76	19	10	16	77

Calcular Distancia

Usuario 1:

Eduardo

Usuario 2:

Eduardo

Tipo de Distancia:

Manhattan

Calcular Distancia

Calcular Distancia y Recomendar

Usuario: Eduardo

Vecino más cercano: User1582221

Película recomendada: película9

Verificación en los logs con el tiempo de respuesta:

```
Eduardo
{
  _id: new ObjectId('656d2e4536c1a8e8a6082c66'),
  usuario: 'Eduardo',
  pelicula1: '87',
  pelicula2: '43',
  pelicula3: '16',
  pelicula4: '10',
  pelicula5: '90'
}
Tiempos total: 1510 ms
Respuesta del servidor: {
  pelicula_recomendada: 'película9',
  usuario_recibido: 'Eduardo',
  vecino_mas_cercano: {
    pelicula1: 87,
    pelicula10: 86,
    pelicula2: 44,
    pelicula3: 15,
    pelicula4: 9,
    pelicula5: 92,
    pelicula6: 56,
    pelicula7: 55,
    pelicula8: 41,
    pelicula9: 87,
    usuario: 'User1582221'
  }
}
Eduardo
100.26.238.150:3001/eliminar-datos
```


Nuevas actualizaciones:

Cambios a flask: Se hicieron cambios a la nueva version de flask, utilizando hasta 5 instancias para el procesamiento de mas datos asi optimizar la carga de datos

```
55
56     # Obtener la película recomendada del vecino más cercano
57     pelicula_recomendada = obtener_pelicula_recomendada(vecino_mas_cercano)
58
59     # Preparar la respuesta
60     respuesta = {
61         'vecino_mas_cercano': vecino_mas_cercano,
62         'pelicula_recomendada': pelicula_recomendada,
63         'usuario_recibido': user_data['usuario'],
64     }
65
66     return jsonify(respuesta)
67
68     except Exception as e:
69         app.logger.error('Error en la función procesar: %s', str(e))
70         return jsonify({'error': 'Error interno'}), 500
71
72  def obtener_pelicula_recomendada(vecino):
73     # Lógica para obtener la película recomendada del vecino más cercano
74     # En este caso, se elige la película con la calificación más alta entre las películas 6 a 10
75
76     # Crear una lista de tuplas (pelicula, calificacion) para las películas 6 a 10
77     peliculas_calificaciones = [(f'pelicula{i}', vecino[f'pelicula{i}']) for i in range(6, 11)]
78
79     # Encontrar la película con la calificación más alta
80     pelicula_recomendada, _ = max(peliculas_calificaciones, key=lambda x: x[1])
81
82     return pelicula_recomendada
83
84  if __name__ == '__main__':
85     app.run(host='0.0.0.0', port=5000, debug=True)
```

Aqui se ve los usuarios y que estamos pidiendo que calcule



Tabla de Resultados

Usuario	Matrix	Dodgeball	WWII	Pokemon	Dr. House
Luis	86	62	27	91	11
Nuevo	1	1	1	1	1
Eduardo	87	63	84	81	10

Calcular Distancia

Usuario 1:

Luis

Usuario 2:

Luis

Tipo de Distancia:

Manhattan

Calcular Distancia

Calcular Distancia y Recomendar

Usuario: Luis

Vecino más cercano: User38891389

Película recomendada: pelicula6

Aqui tenemos el tiempo de respuesta del servidor, ademas del usuario y la calificacion que le dio a las peliculas y sus vecinos mas cercanos

```
Luis
{
  _id: new ObjectId('6573ff91f09e76b853e0c680'),
  usuario: 'Luis',
  pelicula1: '86',
  pelicula2: '62',
  pelicula3: '27',
  pelicula4: '91',
  pelicula5: '11'
}
Tiempo de ejecución del POST: 5349 ms
Respuesta del servidor: {
  pelicula_recomendada: 'pelicula6',
  usuario_recibido: 'Luis',
  vecino_mas_cercano: {
    pelicula1: 85,
    pelicula10: 51,
    pelicula2: 62,
    pelicula3: 27,
    pelicula4: 91,
    pelicula5: 11,
    pelicula6: 90,
    pelicula7: 81,
    pelicula8: 4,
    pelicula9: 73,
    usuario: 'User38891389'
  }
}
Luis
root@ip-172-31-94-45:/home/ubuntu/DAEA-Project# |
```

Las instancias conectadas

```

172.23.0.5 - - [11/Dec/2023 04:05:43] "POST /procesar HTTP/1.1" 200 -
[2023-12-11 04:06:30,826] INFO in app: Usuario recibido: {'_id': '6573ff91f09e76b853e0c680', 'usuario': 'Luis', 'pelicula1': '86', 'pelicula2': '62', 'pelicula3': '27', 'pelicula4': '91', 'pelicula5': '11'}
[2023-12-11 04:06:31,942] INFO in app: Distancia calculada por la instancia actual: 2.6457513110645907
[2023-12-11 04:06:32,910] INFO in app: Vecino mas cercano de la instancia en http://54.161.217.131:5000/procesar: {'pelicula1': 87, 'pelicula10': 5, 'pelicula2': 60, 'pelicula3': 26, 'pelicula4': 92, 'pelicula5': 11, 'pelicula6': 64, 'pelicula7': 19, 'pelicula8': 58, 'pelicula9': 94, 'usuario': 'User13412573'}
[2023-12-11 04:06:32,910] INFO in app: Distancia al vecino más cercano de la instancia en http://54.161.217.131:5000/procesar: 2.6457513110645907
[2023-12-11 04:06:34,127] INFO in app: Vecino mas cercano de la instancia en http://34.200.70.170:5000/procesar: {'pelicula1': 87, 'pelicula10': 5, 'pelicula2': 64, 'pelicula3': 28, 'pelicula4': 92, 'pelicula5': 11, 'pelicula6': 91, 'pelicula7': 77, 'pelicula8': 40, 'pelicula9': 74, 'usuario': 'User28925288'}
[2023-12-11 04:06:34,127] INFO in app: Distancia al vecino más cercano de la instancia en http://34.200.70.170:5000/procesar: 2.6457513110645907
[2023-12-11 04:06:35,234] INFO in app: Vecino mas cercano de la instancia en http://44.216.219.218:5000/procesar: {'pelicula1': 85, 'pelicula10': 51, 'pelicula2': 62, 'pelicula3': 27, 'pelicula4': 91, 'pelicula5': 11, 'pelicula6': 90, 'pelicula7': 81, 'pelicula8': 4, 'pelicula9': 73, 'usuario': 'User38891389'}
[2023-12-11 04:06:35,235] INFO in app: Distancia al vecino más cercano de la instancia en http://44.216.219.218:5000/procesar: 1.0
[2023-12-11 04:06:36,169] INFO in app: Vecino mas cercano de la instancia en http://3.215.220.204:5000/procesar: {'pelicula1': 85, 'pelicula10': 40, 'pelicula2': 62, 'pelicula3': 24, 'pelicula4': 91, 'pelicula5': 11, 'pelicula6': 40, 'pelicula7': 82, 'pelicula8': 11, 'pelicula9': 90, 'usuario': 'User42170866'}
[2023-12-11 04:06:36,169] INFO in app: Distancia al vecino más cercano de la instancia en http://3.215.220.204:5000/procesar: 3.1622776601683795
172.23.0.5 - - [11/Dec/2023 04:06:36] "POST /procesar HTTP/1.1" 200 -
root@ip-172-31-94-45:/home/ubuntu/DAAE-Project# |

```

```

* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 756-589-483
44.221.92.73 - - [11/Dec/2023 04:03:00] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:04:28] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:05:40] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:06:32] "POST /procesar HTTP/1.1" 200 -
root@ip-172-31-60-75:/home/ubuntu# |

```

```

44.221.92.73 - - [11/Dec/2023 04:06:01] "POST /procesar HTTP/1.1" 200 -
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 290-427-951
44.221.92.73 - - [11/Dec/2023 04:03:01] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:04:29] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:05:41] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:06:34] "POST /procesar HTTP/1.1" 200 -
root@ip-172-31-58-33:/home/ubuntu# |

```

```
[1 rows x 11 columns]

Última fila:
      usuario  pelicula1  pelicula2  ...  pelicula8  pelicula9  pelicula10
9999999  User40000001      94      49  ...      15      79      64

[1 rows x 11 columns]
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 161-210-672
44.221.92.73 - - [11/Dec/2023 04:04:05] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:04:30] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:05:42] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:06:35] "POST /procesar HTTP/1.1" 200 -
```

```
[1 rows x 11 columns]

Última fila:
      usuario  pelicula1  pelicula2  ...  pelicula8  pelicula9  pelicula10
9999998  User50000000      16      46  ...      99      71      83

[1 rows x 11 columns]
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 935-098-302
44.221.92.73 - - [11/Dec/2023 04:04:53] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:04:53] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:05:43] "POST /procesar HTTP/1.1" 200 -
44.221.92.73 - - [11/Dec/2023 04:06:36] "POST /procesar HTTP/1.1" 200 -
```