TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CULIACÁN

SISTEMA DE RECOMENDACIÓN

ALUMNO: CHAPARRO CASTILLO CHRISTOPHER

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA: INTELIGENCIA ARTIFICIAL

MAESTRA: ZURIEL DATHAN MORA FELIX

HORARIO: 9:00 – 10:00

FECHA Y LUGAR: CULIACÁN, SIN., 16 DE FEBRERO DEL 2025

1. Tecnología y Frameworks

Tecnologías Clave:

- **Lenguajes de Programación**: Python, Java, Scala, R.
- **Bases de Datos**: SQL (MySQL, PostgreSQL), NoSQL (MongoDB, Cassandra).
- **Procesamiento de Datos**: Apache Hadoop, Apache Spark.
- **Machine Learning**: TensorFlow, PyTorch, Scikit-learn.
- **Cloud Computing**: AWS, Google Cloud Platform (GCP), Microsoft Azure.

Frameworks Populares:

- **Apache Mahout**: Para algoritmos de recomendación escalables.
- **LensKit**: Un framework de código abierto para sistemas de recomendación.
- **Surprise**: Una biblioteca Python para construir y analizar sistemas de recomendación.
- **TensorFlow Recommenders (TFRS)**: Un framework específico para recomendaciones basado en TensorFlow.

2. Herramientas para la Recomendación en Amazon y Google (GCP)

Amazon Web Services (AWS):

- **Amazon Personalize**: Un servicio que utiliza machine learning para crear recomendaciones personalizadas.
- **Amazon SageMaker**: Para construir, entrenar y desplegar modelos de machine learning, incluyendo sistemas de recomendación.
- **AWS Lambda**: Para ejecutar código sin gestionar servidores, útil para la implementación de recomendaciones en tiempo real.

Google Cloud Platform (GCP):

- **AI Platform**: Para entrenar y desplegar modelos de machine learning.
- **BigQuery**: Para el análisis de grandes volúmenes de datos.
- **Cloud Dataflow**: Para el procesamiento de datos en tiempo real y por lotes.
- **Recommendations AI**: Un servicio específico de Google para construir sistemas de recomendación.

3. Algoritmos y Frameworks para la Optimización de Recursos

Algoritmos de Recomendación:

- **Filtrado Colaborativo**: Basado en la similitud entre usuarios o ítems.
  - **User-User Collaborative Filtering**: Recomienda ítems que usuarios similares han gustado.
  - **Item-Item Collaborative Filtering**: Recomienda ítems similares a los que el usuario ha gustado.
- **Filtrado Basado en Contenido**: Recomienda ítems similares en contenido a los que el usuario ha gustado.
- **Modelos Híbridos**: Combina filtrado colaborativo y basado en contenido.
- **Modelos de Factorización de Matrices**: Como Singular Value Decomposition (SVD) y Alternating Least Squares (ALS).
- **Deep Learning**: Utiliza redes neuronales para capturar patrones complejos en los datos.

Algoritmos de Optimización:

- **Dijkstra**: Para encontrar el camino más corto en grafos, útil en sistemas de recomendación basados en grafos.
- **Algoritmos Genéticos**: Para optimizar hiperparámetros en modelos de recomendación.
- **Gradient Descent**: Para optimizar modelos de machine learning.

4. Implementación

Pasos para Implementar un Sistema de Recomendación:

1. **Recopilación de Datos**: Recoger datos de usuarios, ítems y interacciones.
2. **Preprocesamiento de Datos**: Limpiar y transformar los datos para su análisis.
3. **Selección del Modelo**: Elegir el algoritmo de recomendación adecuado.
4. **Entrenamiento del Modelo**: Utilizar datos históricos para entrenar el modelo.
5. **Evaluación del Modelo**: Medir la precisión y efectividad del modelo.
6. **Despliegue**: Implementar el modelo en un entorno de producción.
7. **Monitoreo y Mantenimiento**: Asegurar que el sistema funcione correctamente y actualizar el modelo según sea necesario.

```python
from surprise import Dataset, Reader, SVD
from surprise.model_selection import cross_validate

# Cargar datos
data = Dataset.load_builtin('ml-100k')

# Definir el modelo
algo = SVD()

# Validación cruzada
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

```

Dijkstra

```python
import heapq

def dijkstra(graph, start):
    # Diccionario para almacenar las distancias más cortas desde el nodo inicial
    distances = {node: float('inf') for node in graph}
    distances[start] = 0  # La distancia al nodo inicial es 0

    # Cola de prioridad (heap) para almacenar nodos a explorar
    priority_queue = [(0, start)]

    while priority_queue:
        # Obtener el nodo con la distancia más corta
        current_distance, current_node = heapq.heappop(priority_queue)

        # Si ya encontramos un camino más corto, lo ignoramos
        if current_distance > distances[current_node]:
            continue

        # Explorar los vecinos del nodo actual
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            # Si encontramos un camino más corto, actualizamos la distancia
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances

# Ejemplo de uso
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start_node = 'A'
shortest_distances = dijkstra(graph, start_node)

print(f"Distancias más cortas desde el nodo {start_node}:")
for node, distance in shortest_distances.items():
    print(f"{node}: {distance}")
```

5. Optimización de Recursos

Estrategias de Optimización:

- **Escalabilidad**: Utilizar tecnologías como Apache Spark para manejar grandes volúmenes de datos.
- **Paralelización**: Distribuir el procesamiento en múltiples nodos.
- **Almacenamiento Eficiente**: Utilizar bases de datos optimizadas para consultas rápidas.
- **Caching**: Almacenar resultados de consultas frecuentes para reducir el tiempo de respuesta.