Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Sistemas de recomendación: Métodos de filtrado colaborativo

Óscar Navarro Mesa alu0101504094@ull.edu.es
Alejandro Javier Aguiar Pérez alu0101487168@ull.edu.es
Leandro Samuel Armas Morales alu0101464305@ull.edu.es

# Índice

# Introducción

Este proyecto tiene como objetivo desarrollar un sistema de recomendación utilizando técnicas de filtrado colaborativo, un enfoque ampliamente utilizado para ayudar a los usuarios a descubrir contenido que les pueda interesar. Con el volumen de contenido disponible, los sistemas de recomendación se han vuelto esenciales para mejorar la experiencia del usuario al personalizar sugerencias basadas en sus preferencias individuales y en las calificaciones de otros usuarios.

El sistema se basa en el análisis de una matriz de utilidad, que recopila las calificaciones que los usuarios asignan a diversos temas como películas y series. A través de esta matriz, se pueden identificar patrones de calificación y similitudes entre usuarios. Para calcular estas similitudes, se implementan varias métricas, como la correlación de Pearson, la distancia coseno y la distancia euclidiana.

- **Correlación de Pearson**: Evalúa la relación lineal entre dos usuarios, permitiendo identificar aquellos con patrones de calificación similares, sin tener en cuenta sus promedios individuales.
- **Distancia coseno**: Mide la orientación entre dos vectores de calificación, lo que es útil para encontrar usuarios con gustos comparables, sin importar la magnitud de sus calificaciones.
- **Distancia euclidiana**: Calcula la distancia real entre dos conjuntos de calificaciones, proporcionando una forma directa de ver cuán similares son los usuarios en sus preferencias.

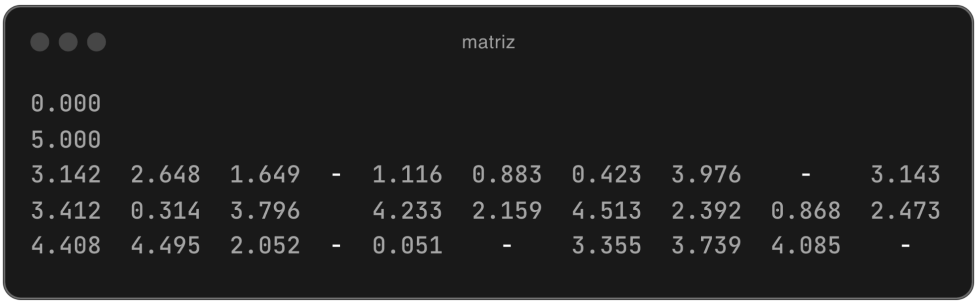
A través de estas métricas, el sistema no solo busca predecir calificaciones faltantes en la matriz, sino que también permite a los usuarios recibir recomendaciones personalizadas. Además, este proyecto representa una oportunidad para que los participantes apliquen conceptos de análisis de datos y aprendizaje automático en un contexto práctico, contribuyendo a su formación en el campo de la ingeniería informática y la gestión del conocimiento en las organizaciones.

## Formato de entrada

El sistema requiere un archivo de texto (.txt) que contenga la matriz de utilidad, con el siguiente formato:

1. Primera línea: Valor mínimo de calificación.
2. Segunda línea: Valor máximo de calificación.
3. Filas subsiguientes: Calificaciones de los usuarios, donde un guion (-) indica una calificación faltante.

Ejemplo de matriz de utilidad:



## Estructura del código

El código se estructura en varias funciones que manejan diferentes aspectos del sistema:

### Lectura de la matriz de utilidad

El sistema empieza leyendo la matriz de utilidad desde un archivo de texto. Esta matriz contiene las calificaciones dadas por los usuarios a los ítems, con NaN indicando valores faltantes. Los valores mínimo y máximo de calificación se leen de las dos primeras filas del archivo.

```
def leer_matriz_utilidad(fichero):
    with open(fichero, 'r') as f:
        lines = f.readlines()

    # Leer los valores mínimo y máximo de puntuación
    min_val = float(lines[0].strip())
    max_val = float(lines[1].strip())

    # Leer las filas de calificaciones, separadas por espacios
    matriz = []
    for line in lines[2:]:
        row = [float(x) if x != '-' else np.nan for x in line.strip().split()]
        matriz.append(row)

    return np.array(matriz), min_val, max_val
```

### Cálculo de métricas de similaridad

Se implementaron tres funciones para calcular la similaridad entre dos usuarios:

1. **Correlación de Pearson**: Considera la correlación entre los vectores de calificaciones.
2. **Similitud coseno**: Calcula el ángulo entre los vectores de calificaciones.
3. **Distancia euclídea**: Mide la diferencia directa entre calificaciones y devuelve la similitud inversa.

Cada función maneja los valores faltantes, ignorándose al calcular las similitudes.

```python
def pearson_correlation(usuario1, usuario2):
    mask = ~np.isnan(usuario1) & ~np.isnan(usuario2)

    if np.sum(mask) == 0:
        return 0  # No hay datos comunes, devolver 0

    usuario1_filtrado = usuario1[mask]
    usuario2_filtrado = usuario2[mask]

    mean1 = np.mean(usuario1_filtrado)
    mean2 = np.mean(usuario2_filtrado)

    numerador = np.sum((usuario1_filtrado - mean1) * (usuario2_filtrado - mean2))
    denominador = np.sqrt(np.sum((usuario1_filtrado - mean1)**2)) *
np.sqrt(np.sum((usuario2_filtrado - mean2)**2))

    return numerador / denominador if denominador != 0 else 0

def cosine_similarity(usuario1, usuario2):
    mask = ~np.isnan(usuario1) & ~np.isnan(usuario2)

    if np.sum(mask) == 0:
        return 0  # No hay datos comunes, devolver 0

    usuario1_filtrado = usuario1[mask]
    usuario2_filtrado = usuario2[mask]

    dot_product = np.dot(usuario1_filtrado, usuario2_filtrado)
    norm1 = np.linalg.norm(usuario1_filtrado)
    norm2 = np.linalg.norm(usuario2_filtrado)

    return dot_product / (norm1 * norm2) if norm1 != 0 and norm2 != 0 else 0

def euclidean_distance(usuario1, usuario2):
    mask = ~np.isnan(usuario1) & ~np.isnan(usuario2)

    if np.sum(mask) == 0:
        return 0  # No hay datos comunes, devolver 0

    usuario1_filtrado = usuario1[mask]
    usuario2_filtrado = usuario2[mask]

    dist_euclidea = np.sqrt(np.sum((usuario1_filtrado - usuario2_filtrado) ** 2))
    return 1 / (1 + dist_euclidea)
```

## Selección de vecinos

Para realizar las predicciones, el sistema selecciona un número de vecinos más cercanos a cada usuario. Esto se hace calculando las similitudes entre el usuario objetivo y todos los demás, ordenando a los vecinos por mayor similaridad.

```python
def obtener_vecinos(matriz, idx, metrica, num_vecinos):
    distancias = []
    entidad = matriz[idx]
    for i in range(matriz.shape[0]):
        if i != idx:
            similaridad = calcular_similaridad(matriz[idx], matriz[i], metrica)
            distancias.append((i, similaridad))

    distancias.sort(key=lambda x: x[1], reverse=True)
    return distancias[:num_vecinos]
```

## Predicciones

El sistema ofrece dos tipos de predicciones:

-**Predicción simple**: Se ponderan las calificaciones de los vecinos según la similitud.

-**Predicción con media**: Se ajusta la predicción considerando las diferencias respecto a la media de cada usuario.

```python
def predecir_simple(matriz, usuario_idx, item_idx, vecinos):
    num, denom = 0, 0
    for vecino_idx, similaridad in vecinos:
        calificacion_vecino = matriz[vecino_idx, item_idx]
        if not np.isnan(calificacion_vecino):
            num += similaridad * calificacion_vecino
            denom += abs(similaridad)
    return num / denom if denom != 0 else np.nan

def predecir_con_media(matriz, usuario_idx, item_idx, vecinos):
    media_usuario = np.nanmean(matriz[usuario_idx])
    num, denom = 0, 0
    for vecino_idx, similaridad in vecinos:
        media_vecino = np.nanmean(matriz[vecino_idx])
        calificacion_vecino = matriz[vecino_idx, item_idx]
        if not np.isnan(calificacion_vecino):
            num += similaridad * (calificacion_vecino - media_vecino)
            denom += abs(similaridad)
    return media_usuario + (num / denom) if denom != 0 else np.nan
```

## Predicción de la matriz completa

Finalmente, se recorre la matriz original y se predicen los valores faltantes en base a los vecinos más cercanos y la métrica seleccionada.

```
def predecir_matriz(matriz, metrica, num_vecinos, tipo_prediccion):
    matriz_predicha = np.copy(matriz)
    n = matriz.shape[0]
    predicciones_hechas = []  # Lista para almacenar las predicciones realizadas

    for idx in range(n):
        vecinos = obtener_vecinos(matriz, idx, metrica, num_vecinos)
        for item_idx in range(matriz.shape[1]):
            if np.isnan(matriz[idx, item_idx]):
                if tipo_prediccion == 'simple':
                    prediccion = predecir_simple(matriz, idx, item_idx, vecinos)
                elif tipo_prediccion == 'media':
                    prediccion = predecir_con_media(matriz, idx, item_idx, vecinos)

                matriz_predicha[idx, item_idx] = prediccion
                predicciones_hechas.append((idx + 1, item_idx + 1, prediccion))  # Almacena el
índice del usuario, el ítem y la predicción

    return matriz_predicha, predicciones_hechas
```

## Interfaz de línea de comandos

El código utiliza "`argparse`" para permitir la entrada de parámetros desde la línea de comandos, especificando el fichero de entrada, la métrica de similaridad, el número de vecinos y el tipo de predicción.

```python
# Función principal
def main():
    parser = argparse.ArgumentParser(description="Sistema de recomendación basado en filtrado colaborativo de películas y series.")
    parser.add_argument('fichero', type=str, help="Ruta al fichero con la matriz de utilidad (TXT)")
    parser.add_argument('--metrica', type=str, choices=['pearson', 'coseno', 'euclidean'], required=True, help="Métrica de similaridad")
    parser.add_argument('--vecinos', type=int, required=True, help="Número de vecinos a considerar")
    parser.add_argument('--prediccion', type=str, choices=['simple', 'media'], required=True, help="Tipo de predicción ('simple' o 'media')")

    args = parser.parse_args()

    # Mostrar opciones seleccionadas
    print(f"\nOpciones seleccionadas:\nMétrica: {args.metrica}\nNúmero de vecinos: {args.vecinos}\nTipo de predicción: {args.prediccion}")

    # Leer la matriz de utilidad
    matriz, min_val, max_val = leer_matriz_utilidad(args.fichero)

    # Mostrar la matriz original
    print("\nMatriz de utilidad original:")
    imprimir_matriz(matriz)

    # Mostrar métricas solo entre usuarios
    mostrar_metricas(matriz, args.metrica)

    # Predecir la matriz y mostrar resultados
    matriz_predicha = predecir_matriz(matriz, args.metrica, args.vecinos, args.prediccion)

    print("\nMatriz de utilidad predicha:")
    imprimir_matriz(matriz_predicha)

if __name__ == "__main__":
    main()
```

### Guardar matriz

Esta función se encarga de guardar la matriz en un fichero junto a las métricas para una mejor visualización de la salida del comando cuando se imprimen matrices grandes.

```
def guardar_matriz(fichero_salida, matriz, min_val, max_val, metricas, predicciones_hechas,
matriz_predicha):
    with open(fichero_salida, 'w') as f:

        # Escribir las métricas
        f.write("Métricas:\n")
        f.write("\n".join(metricas) + "\n\n")

        # Escribir las predicciones
        f.write("Predicciones:\n")
        for usuario, item, prediccion in predicciones_hechas:
            f.write(f"Usuario {usuario}, Ítem {item}: Predicción = {prediccion:.1f}\n")
        f.write("\n")

        f.write("Matriz Predicha:\n")
        f.write(f"{min_val:.1f}\n")
        f.write(f"{max_val:.1f}\n")

        for fila in matriz_predicha:
            fila_formateada = ['{:.1f}'.format(x) if not np.isnan(x) else '-' for x in fila]
            f.write(" ".join(fila_formateada) + "\n")
```

## Ejecución

Para ejecutar el programa, se debe utilizar la línea de comandos de la siguiente manera:

```bash
python sistema_recomendacion.py ../data/matrix/utility-matrix-test.txt --metrica pearson --
vecinos 2 --prediccion simple --salida resultado.txt
```

## Resultados

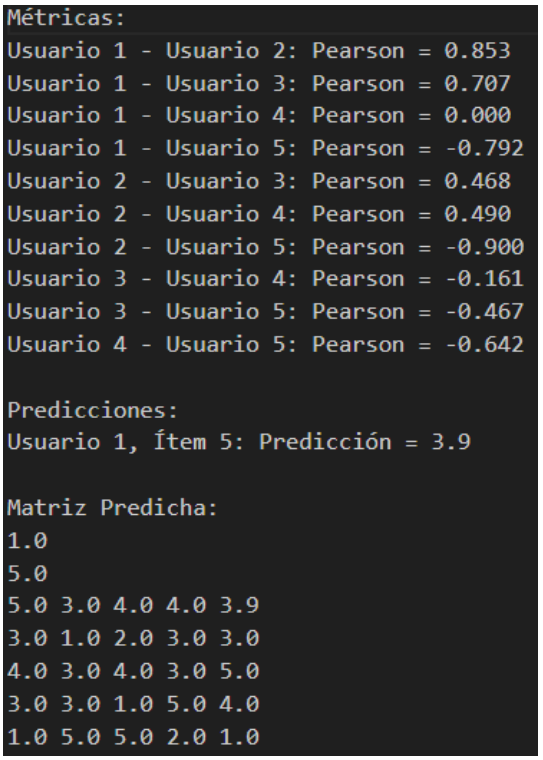
El programa genera un fichero de salida que contiene varias secciones. Primero, se incluyen las métricas calculadas entre los pares de usuarios, indicando las similitudes entre ellos. Luego, se presenta una lista de predicciones realizadas para los ítems faltantes, con el usuario, el ítem correspondiente y el valor de la predicción. Finalmente, se muestra la matriz de utilidad predicha, donde las calificaciones originales se han completado con las predicciones calculadas. El fichero también guarda los valores mínimos y máximos de la escala de puntuación utilizados en la matriz.

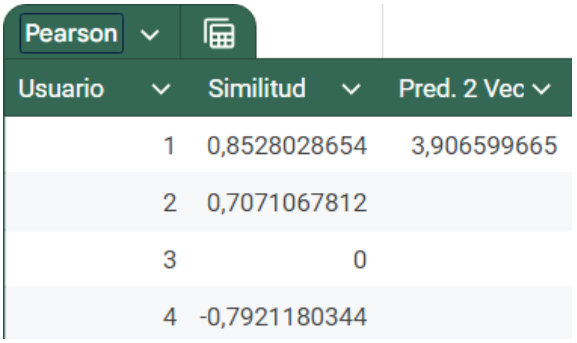
resultado.txt

```
Métricas:
Usuario 1 - Usuario 2: Pearson = 0.853
Usuario 1 - Usuario 3: Pearson = 0.707
Usuario 1 - Usuario 4: Pearson = 0.000
Usuario 1 - Usuario 5: Pearson = -0.792
Usuario 2 - Usuario 3: Pearson = 0.468
Usuario 2 - Usuario 4: Pearson = 0.490
Usuario 2 - Usuario 5: Pearson = -0.900
Usuario 3 - Usuario 4: Pearson = -0.161
Usuario 3 - Usuario 5: Pearson = -0.467
Usuario 4 - Usuario 5: Pearson = -0.642

Predicciones:
Usuario 1, Ítem 5: Predicción = 3.9

Matriz Predicha:
1.0
5.0
5.0 3.0 4.0 4.0 3.9
3.0 1.0 2.0 3.0 3.0
4.0 3.0 4.0 3.0 5.0
3.0 3.0 1.0 5.0 4.0
1.0 5.0 5.0 2.0 1.0
```

## Comprobación de la fiabilidad

El programa funciona correctamente con cualquier tamaño de matriz de utilidad siempre que se use el formato correcto, además hemos podido demostrar haciendo los cálculos en las hojas de cálculo de Google que la respuesta del programa es la esperada usando cualquiera de los tres tipos de métricas. En las siguientes imágenes se muestran los resultados del programa con una matriz de utilidad que se encuentra en el repositorio del trabajo comparado con la salida esperada.




Se ha de mencionar que en “data/comparisons” en el repositorio se encuentran todas las salidas comprobadas en la hoja de cálculo.

## Conclusiones

Este sistema de recomendación demuestra la efectividad del filtrado colaborativo basado en similitudes entre usuarios. La implementación de métricas como la correlación de Pearson, la similitud coseno y la distancia euclidiana permite captar diferentes aspectos de las relaciones en las calificaciones. Por ejemplo, la correlación de Pearson identifica patrones comunes, la similitud coseno evalúa la orientación de los vectores de calificación, y la distancia euclidiana se centra en las diferencias absolutas.

El desarrollo manual de estas métricas favorece un mejor entendimiento de los algoritmos del filtrado colaborativo, lo cual es valioso para estudiantes y desarrolladores. Sin embargo, en aplicaciones a gran escala, se sugiere usar bibliotecas optimizadas que pueden manejar grandes volúmenes de datos y ofrecer técnicas avanzadas.

Además, la selección del número de vecinos y el tipo de predicción afecta directamente la precisión de las recomendaciones. Por último, es crucial considerar la ética en el manejo de datos de los usuarios y garantizar la privacidad y la transparencia en la generación de recomendaciones. En resumen, este trabajo no solo refuerza el conocimiento técnico sobre sistemas de recomendación, sino que también plantea importantes consideraciones éticas en su implementación.