Universidad de La Laguna

Sistemas de recomendación: Filtrado Colaborativo

Gestión del conocimiento de las organizaciones

Déniz Isael Ipekel Castro
Maya Santa Amador
José Miguel Díaz González

ÍNDICE

Introducción

En este documento se trata el procedimiento y desarrollo de nuestro sistema de recomendación mediante métodos de filtrados colaborativos, con un enfoque basado en usuarios. Con el objetivo de mostrar un análisis y conclusión detallada de diversos casos a estudiar.

Método de filtrado Colaborativo: Enfoque en usuarios

Dado un usuario y un ítem que no ha sido visto por él. Buscamos encontrar un conjunto de usuarios, a los que referimos como vecinos cercanos, que tengan gustos parecidos al usuario objetivo y así predecir si le gustará o no el ítem i.

Para ello, dispondremos de una matriz de utilidad, con diversas calificaciones de diversos usuarios y con ciertas casillas sin puntuar, a las que llamaremos incógnitas. Para despejar estas incógnitas usaremos las siguientes métricas:

Correlación de Pearson

Mide el grado de relación lineal entre las valoraciones de dos usuarios, siempre y cuando ambas sean cuantitativas y continuas. Su fórmula y traslación a código son las siguientes:

$$sim(u,v) = \frac{\sum_{i \in S_{uv}} (r(u,i) - \overline{r}(u)) \cdot (r(v,i) - \overline{r}(v))}{\sqrt{\sum_{i \in S_{uv}} (r(u,i) - \overline{r}(u))^2} \sqrt{\sum_{i \in S_{uv}} (r(v,i) - \overline{r}(v))^2}}$$

```java
private double pearson(double[] a, double[] b) {
    double mediaA = media(a);
    double mediaB = media(b);
    double num = 0.0, denA = 0.0, denB = 0.0;

    for (int i = 0; i < a.length; i++) {
        if (!Double.isNaN(a[i]) && !Double.isNaN(b[i])) {
            double da = a[i] - mediaA;
            double db = b[i] - mediaB;
            num += da * db;
            denA += da * da;
            denB += db * db;
        }
    }
    if (denA == 0 || denB == 0) return 0.0;
    return num / Math.sqrt(denA * denB);
}
```

Distancia de Coseno

Se mide el ángulo entre los vectores de valoración. Cuanto más paralelo el vector, más similares son los usuarios. Cuando más cerca esté el valor de 1, más similaridades tienen su preferencia, y cuando más a 0 es, más distintos son. Su fórmula y traducción a código son las siguientes:

$$sim(u,v) = \frac{\sum_{i \in S_{uv}} r(u,i) \cdot r(v,i)}{\sqrt{\sum_{i \in S_{uv}} (r(u,i))^2} \sqrt{\sum_{i \in S_{uv}} (r(v,i))^2}}$$

```java
/** Similitud del coseno */
private double coseno(double[] a, double[] b) {
    double num = 0.0, denA = 0.0, denB = 0.0;

    for (int i = 0; i < a.length; i++) {
        if (!Double.isNaN(a[i]) && !Double.isNaN(b[i])) {
            num += a[i] * b[i];
            denA += a[i] * a[i];
            denB += b[i] * b[i];
        }
    }
    if (denA == 0 || denB == 0) return 0.0;
    return num / (Math.sqrt(denA) * Math.sqrt(denB));
}
```

Distancia Euclídea

Es una medida de diferencia absoluta entre las valoraciones de dos usuarios. La similitud entre usuarios es inversamente proporcional a la distancia entre ambos. La fórmula y traducción a código java de este es la siguiente:

$$d(u,v)_{euc} = \sqrt{\sum_{p \in P} (r(u,i) - r(v,i))^2}$$

```java
private double euclidea(double[] a, double[] b) {
    double suma = 0.0;

    for (int i = 0; i < a.length; i++) {
        if (!Double.isNaN(a[i]) && !Double.isNaN(b[i])) {
            double diff = a[i] - b[i];
            suma += diff * diff;
        }
    }

    double dist = Math.sqrt(suma);
    return 1.0 / (1.0 + dist);
}
```

Tipos de predicción

Para la realización de la práctica nos apoyaremos en dos tipos de predicciones

Predicción simple

La predicción simple se utiliza para estimar la valoración que un usuario podría dar a un ítem basándose directamente en las valoraciones de otros usuarios o ítems similares, utilizando medidas de similitud como referencia. A continuación se muestran la fórmula y su transformación a código.

$$\hat{r}(u,i) = \frac{\sum_{v \in N_u^k} sim(u,v) \cdot r(v,i)}{\sum_{v \in N_u^k} |sim(u,v)|}$$

```java
public double predecirSimple(int u, int i) {
    double[][] simMatrix = sim.getMatrizSimilitud();
    double[][] ratings = um.getMatrix();

    // Obtener los vecinos más similares
    int[] vecinos = topNVecinos(simMatrix[u], ratings, u, i, numVecinos);

    System.out.println("Predicción de " + u + " sobre " + i + ":");

    double num = 0.0;
    double den = 0.0;

    String vecs = "";

    for (int v : vecinos) {
        vecs += v + ",";

        double simUV = simMatrix[u][v];
        System.out.println("Similitud de " + u + " con " + v + ": " + simUV);

        double ratingVI = ratings[v][i];
        System.out.println("Valoración de " + v + " sobre " + i  + ": " + ratingVI);

        if (!Double.isNaN(ratingVI)) {
            num += simUV * ratingVI;
            den += Math.abs(simUV);
        }
    }

    double valorPred;
    if (den == 0) {
        valorPred = Double.NaN;
    } else {
        valorPred = num / den;
    }
```

Predicción con media

La predicción con media se utiliza para estimar la valoración que un usuario podría dar a un ítem considerando su promedio de valoraciones y las diferencias con respecto a usuarios similares. Este método ajusta la predicción según la similitud entre usuarios, ofreciendo resultados más personalizados. A continuación se muestran la fórmula y su transformación a código.

$$\hat{r}(u,i) = \overline{r}(u) + \frac{\sum_{v \in N_u^k} sim(u,v) \cdot (r(v,i) - \overline{r}(v))}{\sum_{v \in N_u^k} |sim(u,v)|}$$

```java
public double predecirConMedia(int u, int i) {
    double[][] simMatrix = sim.getMatrizSimilitud();
    double[][] ratings = um.getMatrix();

    double mediaU = um.getUserMean(u);
    int[] vecinos = topNVecinos(simMatrix[u], ratings, u, i, numVecinos);

    System.out.println("Predicción de " + u + " sobre " + i + ":");
    System.out.println("Media de " + u + ": " + mediaU);

    double num = 0.0;
    double den = 0.0;

    String vecs = "";

    for (int v : vecinos) {
        vecs += v + ",";

        double simUV = simMatrix[u][v];
        System.out.println("Similitud de " + u + " con " + v + ": " + simUV);

        double ratingVI = ratings[v][i];
        System.out.println("Valoración de " + v + " sobre " + i + ": " + ratingVI);

        if (!Double.isNaN(ratingVI)) {
            double mediaV = um.getUserMean(v);
            System.out.println("Media del vecino " + v + ": " + mediaV);
            num += simUV * (ratingVI - mediaV);
            den += Math.abs(simUV);
        }
    }

    double valorPred;
    if (den == 0) {
        valorPred = Double.NaN;
    } else {
        valorPred = mediaU + (num / den);
    }
```

Recomendaciones

Hemos elegido tomar como criterios para hacer las recomendaciones a los usuarios los tres ítems con mayor puntuación y los que superen el valor medio entre la valoración mínima y máxima posible, ya que no vemos sentido recomendar algo que sabemos que no va a gustar al usuario.

Caso a estudiar 1: Matriz 5×10 (Pearson, k = 2, predicción simple)

Análisis

En este caso se usó una matriz pequeña (5 usuarios × 10 ítems) con 9 huecos sin valorar en la matriz.

La métrica elegida fue Pearson, usando 2 vecinos más parecidos por usuario y una predicción simple, sin tener en cuenta la media de cada uno.

- Valor mínimo: 0,0 | Valor máximo: 5,0
- 9 predicciones.
- Algunas similitudes entre usuarios dieron negativas, lo que hace que ciertas predicciones también salgan por debajo de cero.
- Solo se obtuvo una recomendación útil (usuario 2 → ítem 9), ya que la mayoría de los valores predichos no superan la media.

Al haber pocos usuarios y pocos datos compartidos entre ellos, el sistema tiene dificultades para encontrar patrones fiables.

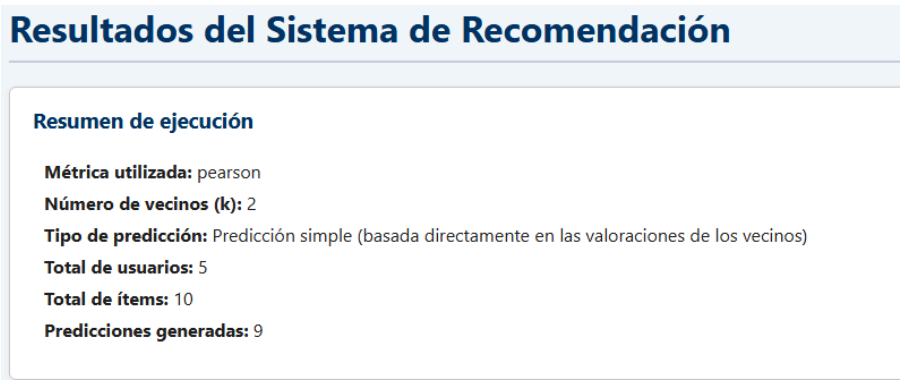
Con Pearson se detectan bien las relaciones entre usuarios, pero si es poca información puede exagerar mucho las similitudes y las diferencias.

Por eso aparecen valores muy negativos. Usando la versión con diferencia de la media, los resultados son más estables.

Conclusión

Con matrices pequeñas, el sistema funciona, pero los resultados son poco fiables.

Es mejor aumentar el número de vecinos o usar otra métrica como por ejemplo el coseno para evitar las correlaciones negativas.



Caso a estudiar 2: Matriz 10×25 (Pearson, k = 3, predicción con diferencia de la media)

Análisis

En este segundo caso se usó una matriz de (10 usuarios × 25 ítems), lo que permite tener más valoraciones cruzadas entre usuarios y un patrón más estable.

La métrica utilizada fue Pearson, con 3 vecinos por usuario y aplicando la predicción con diferencia de la media, que tiene en cuenta el sesgo individual de cada usuario.

Los valores se mueven entre 0,0 y 5,0

El sistema devolvió bastante predicciones, y los valores ahora esi estan en un rango más realista, sin los extremos negativos que aparecían antes.

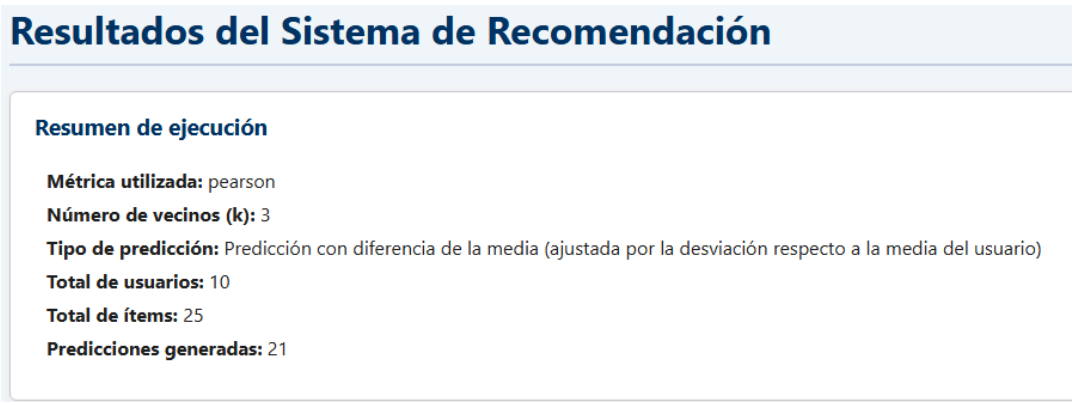
Se ven correlaciones positivas moderadas entre varios usuarios (por ejemplo, usuario 0 con el 6, con una similitud de 0,59).

Se ven recomendaciones como la de los usuarios 0, 1, 2, 3, 4, 6 y 7, que reciben ítems con valores predichos por encima de la media.

Conclusión

Con un mayor número de usuarios e ítems, el sistema se vuelve más fiable y las recomendaciones tienen más sentido.

Las predicciones se distribuyen mejor y la media ajustada ayuda a corregir diferencias entre usuarios muy generosos o muy estrictos con sus puntuaciones.



Caso a estudiar 3: Matriz 25×100 (Pearson, k = 3, predicción con diferencia de la media)

Análisis

En este caso se usó una matriz grande (25×100). Se utilizó la métrica de Pearson, considerando los 3 vecinos más similares por usuario y aplicando la predicción con diferencia de la media, para corregir sesgos individuales y se generaron 65 predicciones.

Las similitudes entre usuarios fueron en su mayoría bajas (entre –0.2 y 0.2) y el sistema logró recomendaciones estables y razonables para usuarios con más datos en común.

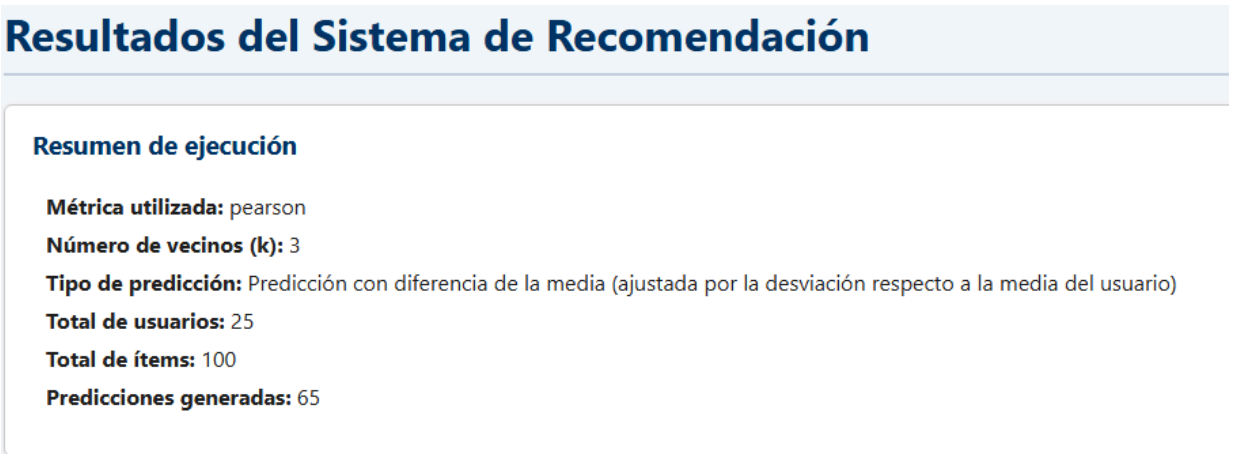
Los resultados muestran una mayor estabilidad que en los casos anteriores porque las predicciones se ajustan mejor al comportamiento medio de cada usuario y no aparecen valores fuera de lo esperado.

Que la matriz sea de mayor tamaño permite que haya más datos en común entre usuarios, generando menos ruido y mejorando la precisión de las recomendaciones.

Conclusión

Con una matriz más grande que antes, el sistema mejora las predicciones ahora son más fiables y equilibradas.

La métrica de Pearson funciona bien en este contexto y, al corregir las medias, se evitan sesgos fuertes.



Caso a estudiar 4: Matriz 50×250 (Pearson, k = 3, predicción con media)

Análisis

En este caso se uso con una matriz mucho más grande (50 usuarios × 250 ítems), lo que permitió que el sistema dispusiera de mucha más información compartida entre usuarios.

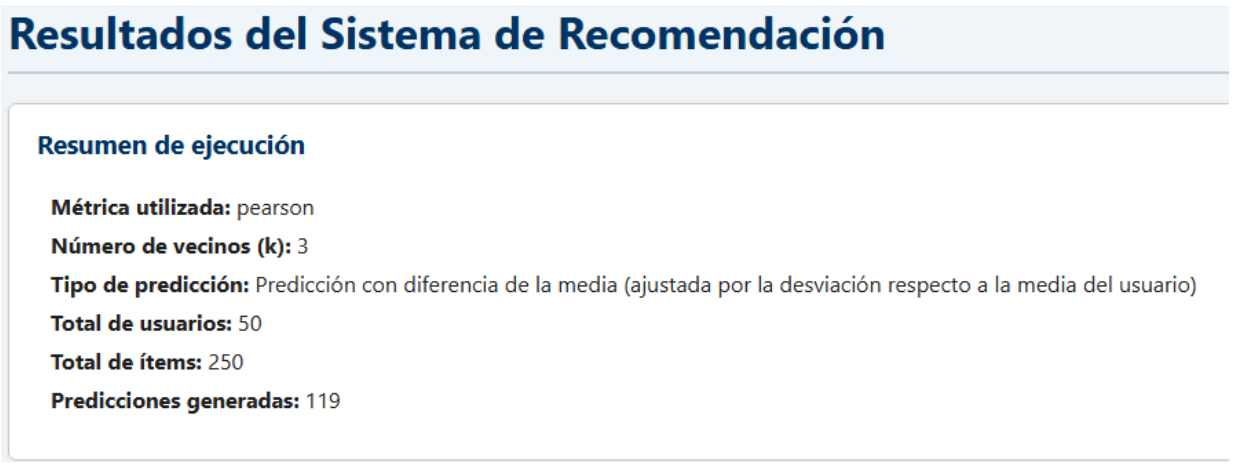
El método Pearson con diferencia de la media devuelve las predicciones muy estables, evitando los valores extremos o incoherentes que se observaban en matrices pequeñas.

El tiempo de procesamiento aumentó muchísimo por el tamaño de los datos, pero los resultados obtenidos son más consistentes y equilibrados.

Las similitudes entre usuarios se distribuyen de forma más homogénea y se aprecian recomendaciones más variadas, con menos casos de sobreajuste.

Conclusión

A medida que crece el tamaño de la matriz, el sistema gana precisión y estabilidad, aunque requiere más tiempo de cálculo y se dificulta ver gráficamente los resultados por el gran tamaño.



Caso a estudiar 5: Matriz 100×1000 (Pearson, k = 3, predicción con media)

Análisis

Este caso representa la prueba con la matriz más grande (100 usuarios × 1000 ítems) El tiempo de cálculo aumenta significativamente, pero el resultado final refleja una red de similitudes bien definida y una distribución de recomendaciones más diversa. La métrica de Pearson, combinada con la corrección por media, muestra un comportamiento muy estable, con valores coherentes y sin predicciones fuera de rango.

Es complicado observar los resultados obtenidos ya que la matriz es demasiado grande, la manera que hemos tenido de poderla visualizar entera es gracias a volcar los datos obtenidos en un html, ya que en la terminal superan los caracteres máximos.

Conclusión

Con un conjunto así de grande de datos, el sistema consigue el mejor rendimiento, con predicciones equilibradas y recomendaciones mas relevantes.

El coste computacional es muy grande y tarda mucho tiempo en procesar toda la información, pero la calidad del filtrado colaborativo mejora muchísimo.