Recomendador

Pseudocódigo de los Algoritmos

Gerard Madrid Miró
Guillem Gràcia Andreu
Ismael Quiñones Gama
Pol Ken Galceran Kimura

En este documento se encuentra el pseudocódigo de los dos algoritmos principales del programa y la unión entre los dos (Hybrid).

COLLABORATIVE FILTERING

En este algoritmo, trataremos tres funciones principales:

1. Cálculo de la distancia entre un centroide y un usuario de KMeans
2. Cálculo del KMeans
3. Cálculo del SlopeOne

1. Función de cálculo de distancia

1: $Suma \ = \ 0$

2: Por cada ítem valorado tanto por el centroide como por el otro usuario:

2.1: $Suma \ += \ (Nota\ centroide\ a\ ese\ ítem \ - \ Nota\ del\ usuario\ a\ ese\ ítem)^2$

3: Retorna $\sqrt{Suma}$

2. Función KMeans

1: Cojo la información de las valoraciones de todos los usuarios.

2: Inicializo los K centroides haciendo que sean iguales a K usuarios aleatorios.

3: Para cada usuario: (Inicialización de los K grupos)

3.1: Miro la distancia a cada uno de los centroides.

3.2: Lo meto en el grupo del centroide al que tenía la mínima distancia.

4: Mientras no haya convergido:

4.1: Ha convergido = true.

4.2: Elimino los datos de los antiguos centroides.

4.3: Los nuevos centroides serán el centro de cada uno de los grupos anteriores:

- Cada centroide será un usuario “falso” con ítems = todos los ítems de los usuarios de su mismo cluster y valoraciones = la media de las valoraciones a esos ítems de los usuarios de ese mismo cluster.

4.4: Para cada usuario:

4.4.1: Miro la distancia a cada uno de los centroides.

4.4.2: Si tiene una distancia a otro centroide menor a la del centroide del grupo en el que se encontraba:

4.4.2.1: Lo borro de su grupo

4.4.2.2: Lo asigno en el nuevo grupo

4.4.2.3: Ha convergido = false

4.5: Si ha convergido == false salta a '4'.

4.6: Sino salta a '5'.

5: Fin. Kgrupos contiene los K diferentes clusters de usuarios.

3. Función slope one

1: Cojo la información de los ítems valorados por el usuario activo.

2: Cojo la información de los usuarios similares al usuario activo (los usuarios del cluster con el que tenga más afinidad).

3: Creo una matriz con filas = usuarios similares y columnas = los ítems que ha valorado el usuario activo. El contenido son las notas a esos ítems (-1 si no valorado).

4: Creo una matriz con filas = usuarios similares y columnas = los ítems NO valorados por el usuario activo. El contenido son las notas a esos ítems (-1 si no valorado)

5: Inicializo vector de predicciones. Items = todos los items NO valorados por el usuario activo y Valoración = -1.

6: Para cada uno de los ítems a predecir (a cada uno lo llamaremos itemI):

6.1: Inicializo vacío el vector ratingJplusDeviationIandJ (RJDIJ)

6.2: Para cada uno de los items que había valorado el usuario activo (itemJ):

6.2.1: SumOfDifferences = 0 (SOD) y NumOfUsersRatingItemsIandJ = 0 ($NURIJ$).

6.2.2: Para cada uno de los usuarios similares al usuario activo (simUser):

6.2.2.1: Cojo el rating de simUser sobre itemI (RII)

6.2.2.2: Cojo el rating de simUser sobre itemJ (RIJ)

6.2.2.3: Los resto

6.2.2.4: Si ambos ratings eran válidos:

$$SOD = RII + RIJ$$
$$NURIJ \mathrel{+}= 1$$

6.2.3: Si numOfUsersRatingItemsIandJ es diferente de 0:

$$MediaDesviacionesParaIyJ\ (MDIJ) = SOD\ /\ NURIJ$$

6.2.4: Si es igual a 0:

MDIJ = Inválido

6.2.5: Si MDIJ no es Inválido:

Sumo la nota de mainUser sobre el itemJ y la desviación entre I y J y se lo añado al vector RJDIJ.

6.3: PredictedRatingI = 0

6.4: Si RJDIJ está vacío:

predictedRatingI = Inválido

6.5: Si no está vacío:

6.5.1: sumaDePrediccionesDeIenBaseAUnJ (SPIJ) = 0

6.5.2: Para cada elemento de RJDIJ (a cada uno lo llamaremos predIenBaseJ):

$$SPIJ \mathrel{+}= predIenBaseJ$$

6.5.3: $PredictedRatingI = SPIJ\ /\ RJDIJ.size()$

7: Ordeno de forma decreciente esas predicciones (Las inválidas tienen nota = MIN_FLOAT así que no me preocupan)

8: Si la predicción es válida pero es mayor al máximo valor que puede tener una nota (o menor al mínimo), la igualo al máximo (o mínimo)

9: Retorno las N primeras posiciones (siendo N el número de predicciones que me han solicitado)

CONTENT BASED FILTERING

De este algoritmo, trataremos tres funciones principales:

1. Cálculo de las ponderaciones
2. Cálculo del algoritmo de Jaccard
3. Cálculo de similitud entre dos ítems

1. Cálculo de ponderaciones (ComputePonderations)

La intención de este algoritmo es calcular qué importancia tiene un atributo sobre el resto a la hora de comparar dos ítems. Para comprobarlo, hago lo siguiente:

1: Creo dos ArrayList nuevas. La primera tendrá los valores del atributo, mientras que la segunda tendrá el número de veces que se ha repetido ese valor para ese atributo.

2: Corto el máximo de ítems a tratar en 1500 (explicado en Decisiones tomadas), y asigno el valor de dos variables al 90 y 20 por ciento del número de ítems.

3: Cojo el número de atributos que tienen los items a comprobar, e itero sobre cada atributo (es decir, las columnas de la matriz).

4: Limpio el valor de los ArrayList definidos puesto que vamos a tratar con un nuevo atributo que no hemos ponderado aún y necesitamos usar estas estructuras de datos.

5: Itero sobre cada valor tomado por el atributo entre todos los ítems (es decir, cada valor de un atributo para **un** ítem).

6: Itero sobre cada valor dentro del bucle anterior para tratar casos de atributos con valores múltiples. (P.ej: {en, es, fr, it})

7: Ahora para cada uno de los anteriores, instancio una variable temporal de tipo String vacía, cojo el atributo concreto (P.ej: {en}), busco si mi ArrayList de valores lo contiene y:

7.1: Si contiene el valor, sumo uno al ArrayList de conteo.

7.2: Si no contiene el valor, lo añado al ArrayList de valores y añado un 1 al ArrayList de conteo.

8: Una vez hecho eso para todos los valores de un atributo (P. ej: {en, es, fr, it}), entran en juego los descartes para tener en cuenta o no ese atributo a la hora de comparar:

8.1: Si el tamaño del ArrayList de valores es mayor al noventa por ciento del total de ítems (es decir, si más del 90% de los ítems son totalmente distintos unos de otros), no los tengo en cuenta y asigno un 0 a su posición en el ArrayList de ponderaciones.

8.2: En caso contrario, miro la dispersión de los datos:

8.2.1: Creo una variable temporal dispersionCount encargada de tener la cuenta de la distribución del atributo analizado.

8.2.2: Itero sobre el ArrayList de valores, y compruebo para cada valor si la cuenta (el número de veces que ha aparecido) es mayor al veinte por ciento del número de items. En caso de serlo, añado uno a la variable dispersionCount y compruebo si la cuenta es mayor a su vez que el noventa por ciento del tamaño de los ítems, en cuyo caso se asigna directamente un 0 a la variable de dispersión y se sale del bucle.

8.2.3: Por último, si la cuenta de dispersión tras el bucle es mayor al veinte por ciento del número total de ítems o vale 0, se asigna un 0 a su posición en el ArrayList de ponderaciones. De cualquier otro modo, se considera que el atributo ha pasado las pruebas y le asigno un 1 y lo tendré en cuenta para comparar items.

2. Cálculo del algoritmo de Jaccard (ComputeJaccard)

La intención de este algoritmo es comparar dos strings que entran por parámetro y obtener su parecido. El funcionamiento es el siguiente:

1: Creo dos Sets de String llamados intersección y unión para llenarlos más adelante.

2: Guardo el tamaño de los strings recibidos por parámetro y, en caso de valer 0 alguno de los 2, retorno un 0 como parecido puesto que no se pueden comparar.

3: Inicio en falso un atributo booleano que usaremos más adelante, y empiezo a iterar sobre el tamaño del primer string recibido por parámetro. Para cada iteración hago lo siguiente:

> **3.1:** Añado a la unión el carácter i-ésimo del primer string.
>
> **3.2:** Itero sobre el segundo string recibido por parámetro y hago lo siguiente:
>
> > **3.2.1:** Si los dos caracteres iterando (i-ésimo del primer string y j-ésimo del segundo) son iguales, los añado a la intersección.
> >
> > **3.2.2:** Siempre que el atributo finished no sea cierto, añado a la unión también el carácter j-ésimo.
>
> **3.3:** Una vez terminado el segundo bucle, pongo a cierto el atributo finished dado que ya se ha terminado el segundo string pasado por parámetro, y ahora solo se añadirán a la unión los caracteres restantes del string primero.

4: Retorno como resultado un float que contiene la división de la intersección y la unión de los dos strings siguiendo así el algoritmo principal de Jaccard.

3. Cálculo de similitud entre dos items (CompareItems)

La intención de este algoritmo es comparar dos ítems que entran por parámetro y evaluar su similitud usando los dos algoritmos anteriores. El funcionamiento es el siguiente:

1: Inicio en 0 dos variables: countIguales y comparisonCount.

2: Compruebo si los dos ítems comparten ID, en cuyo caso retorno -1 para que se trate de forma distinta a los demás, dado que se estarían intentando comparar dos ítems iguales.

3: Itero sobre la cantidad de atributos de los ítems (al ser ambos del mismo dataset deberían compartir número de atributos). Para cada atributo hago lo siguiente:

> **3.1:** Compruebo si hay que tener en cuenta ese atributo mirando su valor en la lista de ponderaciones. Solo sigo en caso de ser mayor que 0.
>
> **3.2:** Sumo uno a comparisonCount dado que vamos a hacer una comparación y esa variable sirve de contador, y obtengo el tamaño del atributo para ambos ítems. (P.ej: SizeOf({en, es, it}) = 3)
>
> **3.3:** Itero sobre ambos valores recién obtenidos, y diferencio su tratamiento si son String o no:
>
> > **3.3.1:** En caso de no ser un atributo de la clase String, compruebo si ambos valores son iguales, en cuyo caso sumaré un valor proporcional a su peso al countIguales.
> >
> > (<Ejemplo>)
> > AttrA = {10, 25, 50}
> > AttrB = {10, 11, 11}
> > countIguales += 2/6
> > (</Ejemplo)
> >
> > **3.3.2:** En caso de ser un atributo de la clase String, utilizo Jaccard para comparar ambos strings y guardo su valor en una variable auxiliar que luego se divide entre la suma de ambos tamaños y se añade al contador countIguales.

4: Finalmente, se devuelve el porcentaje de cercanía de ambos ítems mediante la división de countIguales entre comparisonCount multiplicado por 100.

HYBRID APPROACH

De este algoritmo, trataremos la recomendación de Ítems a usuarios mediante la unión de los dos algoritmos explicados anteriormente. Los pasos son los siguientes:

1: Declaro thresholds:

> **1.1:** Threshold de la K del content based (la K del collaborative se obtiene en la constructora).
>
> **1.2:** Threshold del número de recomendaciones que le pediré al collaborative.
>
> **1.3:** Threshold del porcentaje a partir del cual considero un ítem "similar".
>
> **1.4:** Threshold a partir del cual considero una buena valoración (P.ej: $0.8 \cdot maxRatingValue$).

2: Obtengo las valoraciones del user al que quiero recomendar.

3: De las valoraciones, me quedo solamente con aquellas que tengan una nota superior o igual a *POSITIVE_RATING_THRESHOLD*.

4: Obtengo todos los items similares a los del paso 3 utilizando el content based filtering.

5: Le pido un cierto número de recomendaciones al collaborative filtering.

6: Inicializo vacío el vector de recomendaciones

7: Para cada una de las recomendaciones retornadas por el collaborative, en el mismo orden en el que me las retornó:

> **7.1:** Miro si ese ítem fue alguno de los retornados por el content based en el paso 4.
>
> **7.2:** Si ese es el caso:
>
> > Añado en el vector de recomendaciones ese ítem (con nota predicha igual a la que me dijo el collaborative)

8: Si el usuario quería más recomendaciones de las que hay hasta este punto, voy añadiendo a la cola del vector de recomendaciones los primeros elementos del collaborative que no había añadido ya.

9: Retorno las recomendaciones.

Apunte: El punto favorable del algoritmo híbrido es que retorno como recomendaciones aquellos ítems muy similares a los que le habían gustado al usuario, pero en el mismo orden que el collaborative me predijo en sus primeras X recomendaciones. (En nuestro caso usando X = 50).

Cargar CSV

1: Abrir fichero en modo lectura

2: Leer primera línea (nombre de las columnas)

3: Mientras (haya más filas)

3.1: Leer siguiente fila

3.2: Dividir por columnas

3.3: Crear un Objeto nuevo

3.4: Encontrar el tipo de la celda (int, float, boolean, String) y hacer el cast

3.5: Añadirlo al conjunto de ese tipo de Objeto

4: Cerrar fichero

Guardar CSV

1: Abrir fichero en modo escritura (borrando todo su contenido)

2: Escribir la línea de los nombres de las columnas

3: Por todo Objeto dentro del conjunto

3.1: Crear un string con cada atributo en el mismo orden que esté el CSV y con ese formato (separación por comas)

3.2: Escribir fila de datos

4: Cerrar fichero