# SISTEMA RECOMANADOR

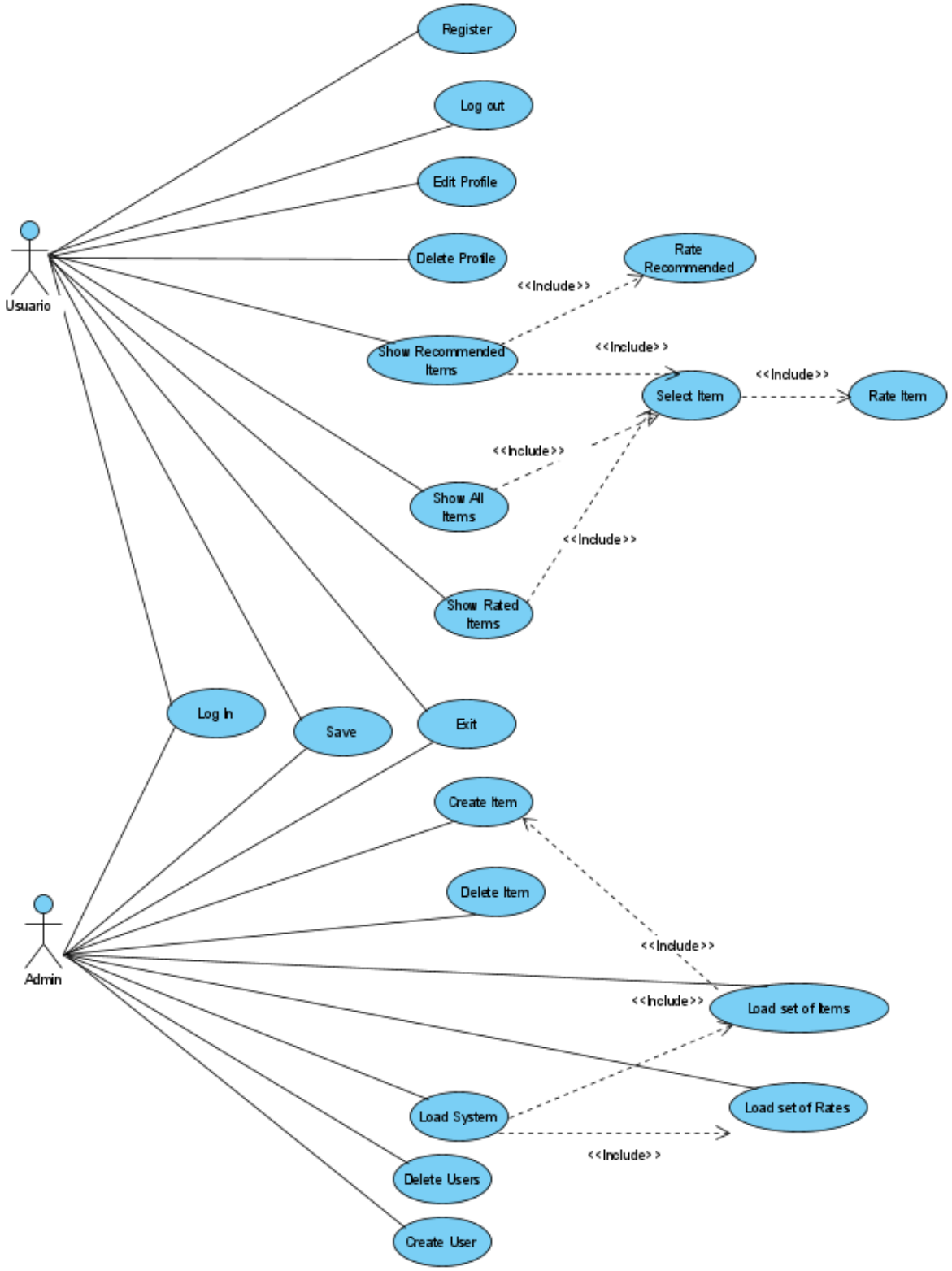
## PROP Curs 2021/22
## SubGrup 6.3

Roberto Amat Alins
Marc Camarillas Parés
Oriol Cuéllar Barrionuevo
Jordi Olmo Ricis

# Índice

# 1.DEFINICIÓN DE CASOS DE USO

## Diagrama de Casos de Uso



# Definición

## Register

**Actores:** Usuario

**Descripción:**

El usuario introduce su UserId que lo identifica y la contraseña. Se crea el usuario y se le asigna el rol de usuario. El usuario activo es el creado.

**Errores posibles:**

UserId ya existe, alguien ya ha iniciado sesión.

## Log in

**Actores:** Usuario, Administrador

**Descripción:**

El usuario o administrador introduce su UserId que lo identifica, la contraseña. Se actualiza el usuario activo en el sistema.

**Errores posibles:**

UserId no existe, contraseña incorrecta.

## Log out

**Actores:** Usuario, administrador.

**Descripción:**

El usuario activo pasa a ser null.

**Errores posibles:**

El usuario activo ya era null.

## Edit Profile

**Actores:** Usuario

**Descripción:**

El usuario activo indica que atributo quiere modificar. Se introducen los nuevos datos y se actualiza el usuario.

**Errores posibles:**

UserId nuevo ya existe, usuario activo es un administrador.

## Delete Profile

**Actores:** Usuario

**Descripción:**

El sistema elimina el usuario activo y todas sus valoraciones.

**Errores posibles:**

Usuario activo es administrador.

## Show recommended Items

**Actores:** Usuario

**Descripción:**

El sistema muestra al usuario activo un conjunto de ítems recomendados

**Errores posibles:**

Usuario activo es un administrador, no hay ítems en el sistema.

## Rate recomendation

**Actores:** Usuario

**Descripción:**

El sistema utiliza un algoritmo para determinar la precisión de una recomendación.

**Errores posibles:**

Ninguno

## Select Item

**Actores:** Usuario

**Descripción:**

El sistema actualiza el ítem activo.

**Errores posibles:**

No hay ítems en el sistema, usuario activo es administrador.

## Rate Item

**Actores:** Usuario

**Descripción:**

El usuario valora el ítem seleccionado.

**Errores posibles:**

Ítem seleccionado es null, la valoración no está entre 0 y 5, el usuario activo es administrador.

## Show all Items

**Actores:** Usuario

**Descripción:**

El sistema muestra todos los ítems guardados por el sistema.

**Errores posibles:**

No hay ítems en el sistema, usuario activo es administrador.

## Show Rated Items

**Actores:** Usuario

**Descripción:**

El sistema muestra todos los ítems valorados por el usuario activo en el sistema.

**Errores posibles:**

El usuario activo es null, no hay ítems valorados por el usuario activo, usuario activo es administrador.

## Save

**Actores:** Usuario, administrador

**Descripción:**

El sistema guarda toda la información en ficheros CSV.

**Errores posibles:**

Ninguno

## Exit

**Actores:** Usuario, administrador

**Descripción:**

El sistema se cierra

**Errores posibles:**

Ninguno

## Create Item

**Actores: A**dministrador

**Descripción:**

Se introducen los datos de un ítem con el nombre de los atributos. Si no existía ese tipos de ítem se crea.

**Errores posibles:**

Ítem ya existe, usuario activo no es administrador.

## Delete Item

**Actores: A**dministrador

**Descripción:**

Se introducen los datos de un ítem y se borra del sistema.

**Errores posibles:**

Datos no corresponden con ningún ítem en el sistema, usuario activo no es administrador.

## Load set of Items

**Actores: A**dministrador

**Descripción:**

Se introduce el path a un documento con ítems i se cargan en el sistema.

**Errores posibles:**

Path incorrecto, documento incorrecto, usuario activo no es administrador, ítem ya existe.

## Load set of Rates

**Actores: A**dministrador

**Descripción:**

Se introduce el path a un documento con valoraciones y se cargan en el sistema.

**Errores posibles:**

Path incorrecto, documento incorrecto, usuario activo no es administrador.

## Delete User

**Actores: A**dministrador

**Descripción:**

Se introduce el userId y se borra del sistema el usuario correspondiente.

**Errores posibles:**

UserId no corresponde a ningún usuario, UserId es un administrador, usuario activo no es un administrador.

## Create User

**Actores: A**dministrador

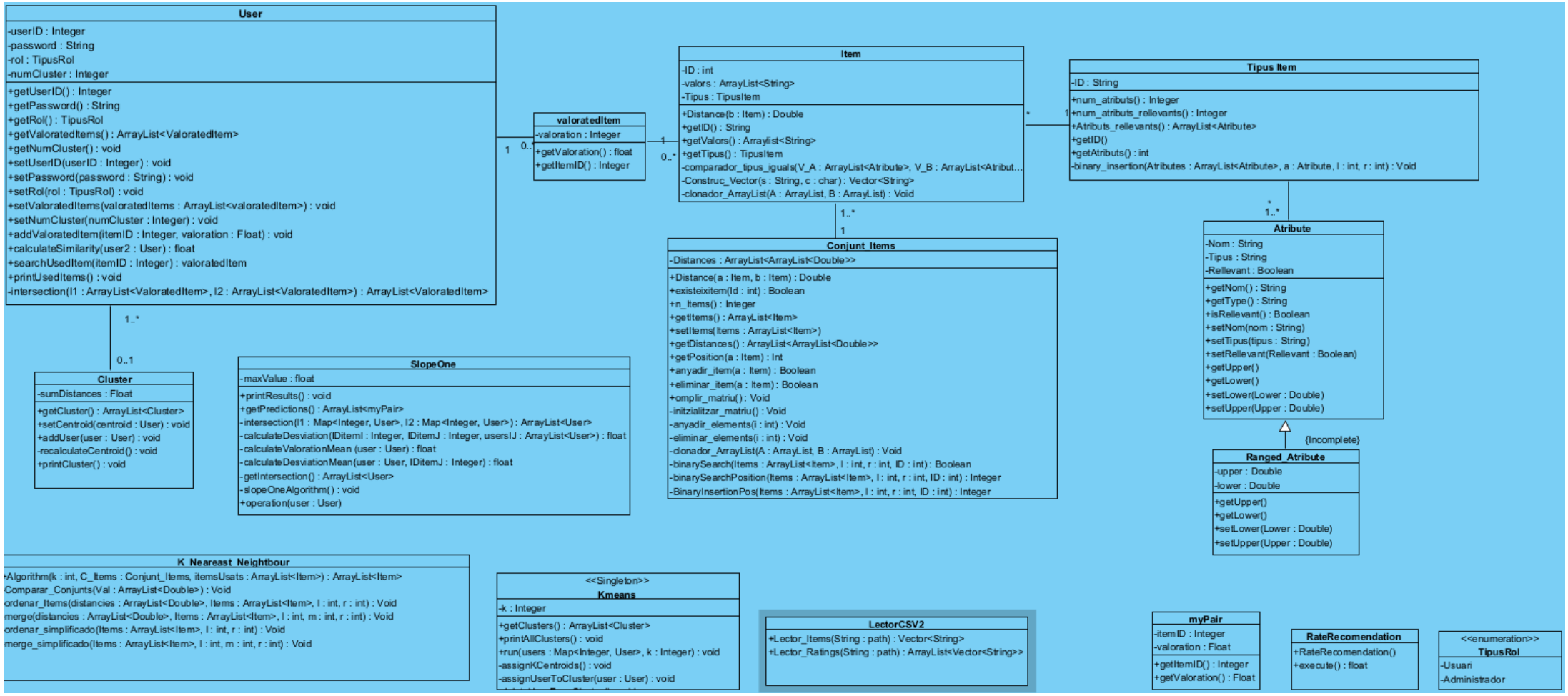
**Descripción:**

Se introduce el userId, contraseña y se crea en el sistema el usuario.

**Errores posibles:**

UserId no corresponde a ningún usuario, UserId es un administrador, usuario activo no es un administrador, usuario ya existe.

# 2. MODELO CONCEPTUAL



# 3.DESCRIPCIÓN DE LA ESTRUCTURA DE DATOS Y ALGORITMOS

SlopeOne

Para esta clase hemos visto conveniente usar un map que representa para cada ítem que usuarios lo han valorado. Con esto lo que conseguimos es poder encontrar rápidamente la intersección de los usuarios que han valorado ambos ítems, necesario para luego calcular las desviaciones entre el ítem que se quiere predecir y los otros ítems.

User

Para esta clase usamos un ArrayList de ítems valorados para representar los items que ha valorado cada usuario.
Para calcular las distancias entre usuarios usamos un coeficiente de similaridad que va entre 0 y 1, donde 0 significa que los usuarios no tienen nada en común y 1 quiere decir que los usuarios son iguales. Este coeficiente lo calculamos con el cosine-based approach, el cual se basa en el producto escalar de dos vectores. Se calcula con la siguiente fórmula:

$$\text{simil}(x,y) = \cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{||\vec{x}|| \times ||\vec{y}||} = \frac{\sum_{i\in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i\in I_x} r_{x,i}^2}\sqrt{\sum_{i\in I_y} r_{y,i}^2}}$$

valoratedItem

Un ítem valorado está formado por un objeto de la clase Ítem y la valoración que se le ha dado. Como funciones principales solo tiene getters.

Cluster

Para esta clase hemos optado por utilizar un ArrayList de *User* que contiene todos los usuarios que forman parte del cluster, y otro para las distancias. Utilizamos ArrayList dado que no necesitamos ni tenerlos ordenados ni cualquier otra propiedad que nos anime a usar otra estructura.
No almacenamos todos los pares de distancias entre cada par de usuarios del cluster, sino únicamente la suma de estas, ya que un par de distancias entre dos usuarios es irrelevante, aquello relevante es la suma de todas las distancias para designar el nuevo centroide.

Kmeans

En esta clase utilizamos un ArrayList de *Clusters* para guardar todos los clusters que forman parte del algoritmo. Usamos esta estructura de datos ya que no necesitamos realizar ningún tipo de búsqueda ni ordenación sobre ésta, y una con una mayor complejidad no merecería la pena.
Para almacenar los usuarios que el algoritmo repartirá en los *k clusters* utilizamos un Mapa, donde el key de cada usuario es su ID. Utilizamos este Mapa ya que se puede adecuar correctamente a esta clase y nos sirve para muchas otras, aportando una gran flexibilidad.

El algoritmo se basa en generar inicialmente *k clusters* con centroide aleatorio e ir añadiendo a estos los usuarios en función de su proximidad. La proximidad se calcula con la función explicada previamente de la clase *User*.

myPair

En esta clase únicamente almacenamos un entero *itemID* y un float *valoration*. Como funciones tiene únicamente getters. No es necesario utilizar ninguna estructura de datos.

K_Nearest_Neightbour

Esta clase es la que implementa el algoritmo del mismo nombre. El algoritmo en sí se encuentra en la función Algorithm. Este recibe como parámetros: el número de k de Ítems a recomendar, un conjunto de ítems guardado en ConjuntItems (una estructura de datos propiamente implementada, que se detalla más adelante), una

Arraylist con ítems valorados por el usuario y otra ArrayList con las valoraciones donde las posiciones de los Ítems y sus valoraciones deben coincidir.

Primero adecua el tamaño de las valoraciones para que coincida con el de ConjuntItems, mientras añade los Ítems valorados si estos no estaban en el conjunto. Como se ha añadido Ítems se calcula la matriz Distance(se verá más adelante)y se guarda esta.

Seguidamente procede a hacer una copia de la Array del ConjuntItems creando una matriz de n x n ítems, a la vez que copia también la de Distance(que tendrá el mismo tamaño). Mientras crea estas matrices cada fila de las dos matrices las ordena de forma decreciente según la distancia(como las ordena a la vez cada ítem y su distancia siguen correspondiendo) utilizando una modificación del merge_sort.

Finalmente llama a la función comparar conjuntos y está rellena una Arraylist de ítems de tamaño k con la solución al algoritmo.

comparar_conjunts

Comparar_conjunts mira las dos matrices que le pasan como parámetros (de ítems y de distancias creadas en Algorithm) y va guardando en una Arraylist de Doble que corresponde cada posición con la posición de un ítem el resultado de la siguiente fórmula:
Siendo las matrices en orden de los parámetros X,Y y el vector V -> por unos Ítem i, j cualquiera. Valors_i = Sumatorio(Xij * Vi).

Seguidamente ordena el ArrayList I_Finals decrecientemente según valors usando merge_sort y devuelve los k primeros.

ConjuntItems

ConjuntItems és la estructura de datos implementada que se usa básicamente para almacenar los Ítem y guardar las distancias calculadas entre ellos.

Por eso la estructura se basa en una ArrayList de Item ordenada crecientemente por los ID de los Ítems. Para eso utiliza algoritmos derivados de merge_sort y búsqueda binaria para ordenar, buscar añadir y eliminar Items. Se utilizó este método porque a lo largo del algoritmo se requerirían hacer muchas consultas sobre el Conjunt_Items y así cada una bajaría a coste logarítmico.

También cuenta con una matriz de ArrayList de Double con la distancia de todos los ítems entre sí. Cada vez que se añade un item este se pone con distancia -1.0 con todos los ítem para así después poder rellenarlo. Cada vez que se requieran distancias de 2 items y en la matriz sea -1, esta se calculará y se guardará en la matriz para facilitar futuros cálculos. Como si un ítem se elimina o se modifica también se modificará la matriz se mantendrá la consistencia.

Item

Cada Ítem está formado por su Id (que lo identifica), su Tipus y su ArrayList de String Valores. A su vez el tipus está formado por un ArrayList de Atributes cada uno con su nombre tipo y Booleana que indica si es relevante y un ID que será [] con detro todos los nombres de los atributos que lo forman separados por comas, haciendo asi que sea único . La posición de la ArrayList de atributos del tipo del item tiene que coincidir con la del valor de ese atributo en el vector de String.

Atribute

El atributo como se ha dicho está formado por el nombre el tipo y si es Relevante. El nombre puede ser cualquier String, el Tipus tiene que debería ser uno de los siguientes para usarse en el algoritmo {Boolean, String, Vector de String, Data, Rang}. Y si es relevante o no, para si se tiene que tener en cuenta a la hora de calcular las distancias.

Los tipos son autoexplicativos menos Rang que además tiene una subclase propia Ranged_Atribute para los que son de ese tipo. Este representa los atributos que tienen un máximo y un mínimo necesarios para compararlos, pueden ser Int , float o Double. Por eso la existencia de la subclase ya que tiene dos atributos extras para guardar este minimo y maximo Lower y Upper.

Distance

Con todo esto se puede entender el algoritmo para calcular la distancia entre dos ítems llamado Distance que se encuentra en la clase Item.

Este recorre las dos ArrayList de valores y los va comparando según el tipo de atributo en la misma posición y va sumando el resultado en un double que será el resultado de la función (teniendo en cuenta que se usa para calcular la distancia una variación de la distancia euclidiana donde cada solución local se eleva al cuadrado y la distancia total es la suma de estas soluciones locales haciendo la raíz cuadrada.

Si és Booleano o String da 1 si son iguales 0 en caso contrario.Siés un vector de String drara el número de String iguales dividido entre el tamaño del mayor vector, si es una Data dará la diferencia en días de las dos fechas dividido entre el número de días de 50 años, haciendo así que las distancias mayores a 50 años de 0. Finalmente si es Rang da la resta el valor más grande menos el más pequeño, dividido entre el max menos el mínimo. Esto es a menos que el máximo y el mínimo sean iguales en cuyo caso devolverá uno ya que los dos valores deberán valer lo mismo.

Rate Recomendation

En esta clase se evalúa una recomendación. Mediante un algoritmo se obtiene un número normalizado a 1 que representa la precisión de la valoración. Siendo 1 una valoración perfecta y 0 una valoración nefasta.

Teniendo un vector de valoraciones de longitud p.
i = posicion actual del vector.
reli = valoración en la posición i del vector.

Valoración obtenida.

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

Valoración ideal.

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

Valoración normalizada a 1.

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p},$$

LectorCSV2

Esta clase tiene métodos para leer CSV. Concretamente CSV que contienen ítems o valoraciones de ítems por parte de usuarios. Se utilizan vectores de Strings para devolver los ítems leídos. Todo el item se almacena en un único string separado por comas. Se utilizan ArrayList de vectores de Strings para devolver las valoraciones leídas. Teniendo la primera posición del vector el UserID, la segunda el ItemID, y en la tercera la valoración.

# 4. REPARTICIÓN DEL TRABAJO

- **Roberto Amat**
  - Kmeans.java
  - DriverKmeans.java
  - myPair.java
  - DriverMyPair.java
  - Cluster.java
  - Estructuración Proyecto
  - DriverCluster.java
  - Doxygen

- **Marc Camarillas**
  - SlopeOne.java
  - DriverSlopeOne.java
  - User.java
  - DriverUser.java
  - valoratedItem.java
  - DriverValoratedItem.java
  - Makefile

- **Oriol Cuellar**
  - RateRecomendation.java
  - LectorCSV2.java
  - DriverRateRecomendation.java
  - DriverLectorCSV2.java
  - Main.java
  - Descripción casos de uso
  - Diagrama casos de uso
  - Documentación

- **Jordi Olmo**
  - Atribute.java
  - DriverAtribute.java
  - Ranged_Atribute.java
  - DriverRanged_Atribute.java
  - ConjuntItems.java
  - DriverConjuntItems.java
  - K_Nearest_Neightbour.java
  - DriverKND.java

- Tipus_Item.java
- DriverTipusItem.java
- Item.java
- DriverItem.java
- Suite_Resto_Item.java
- Suite_Distance_Item.java

# 5. MANUAL DE USO

- La documentación de cada clase se encuentra en el directorio /Docs/DoxyGen . Para consultarla hay que abrir el archivo index.html que se encuentra en el mismo directorio.

- Hacer # make desde el directorio donde se encuentra el Makefile para compilar todas las clases.

- Si se desea ejecutar los drivers hay que hacer # make <nombreDriver> o # java -cp ./EXE FONTS.src.domini.drivers.nombreDriver .

- Los test por defecto de los drivers se encuentran en el directorio ./EXE/Entradas_CSV . Se pueden usar estos ficheros o el propio driver te dará la opción de cargar los ficheros que se deseen usar.

- Si lo que se desea es ejecutar el Main hay que hacer # make Main o # java -cp ./EXE FONTS.src.domini.model.Main.

- Cada ejecutable al ejecutarse imprime sus propias instrucciones de uso, además en el directorio ./EXE hay una pequeña explicación de cada ejecutable.

Nos ha dado error al ejecutar junit por terminal, pero hemos conseguido poderlo ejecutar desde un IDE y hemos comprobado que pasa todos los test tal y como sale en las siguientes imágenes.


Run:
DriverItem
Tests passed: 17 of 17 tests – 15 ms
DriverItem (FONTS.src.domini.drivers) 15 ms
Suite_Distance_Item 15 ms
Suite_Resto_Item 0 ms
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -ea -Didea.test.cyclic.buffer.size=1048576 -javaagent:/snap/intellij-idea-com
Process finished with exit code 0

# 6.BIBLIOGRAFIA


- Wikipedia(2021). Cosine-Similarity. Recuperat el 10 de novembre de 2021, des de https://en.wikipedia.org/wiki/Cosine_similarity
- Wikipedia(2021). Collaborative Filtering. Recuperat el 10 de novembre de 2021, des de https://en.wikipedia.org/wiki/Collaborative_filtering
- Wikipedia(2021). Discounted Comulative Gain, des de https://en.wikipedia.org/wiki/Discounted_cumulative_gain