SISTEMA RECOMANADOR

PROP Curs 2021/22
Segunda Entrega
SubGrup 6.3

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

Índice

1. Diagramas de Diseño

Adjuntamos en la carpeta de la entrega un archivo en formato PNG que contiene el diagrama completo (de las tres capas), y uno por separado de cada una de ellas.2. Mejoras Algoritmos.

2. Mejoras Algoritmos

2.1 K-means

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*.

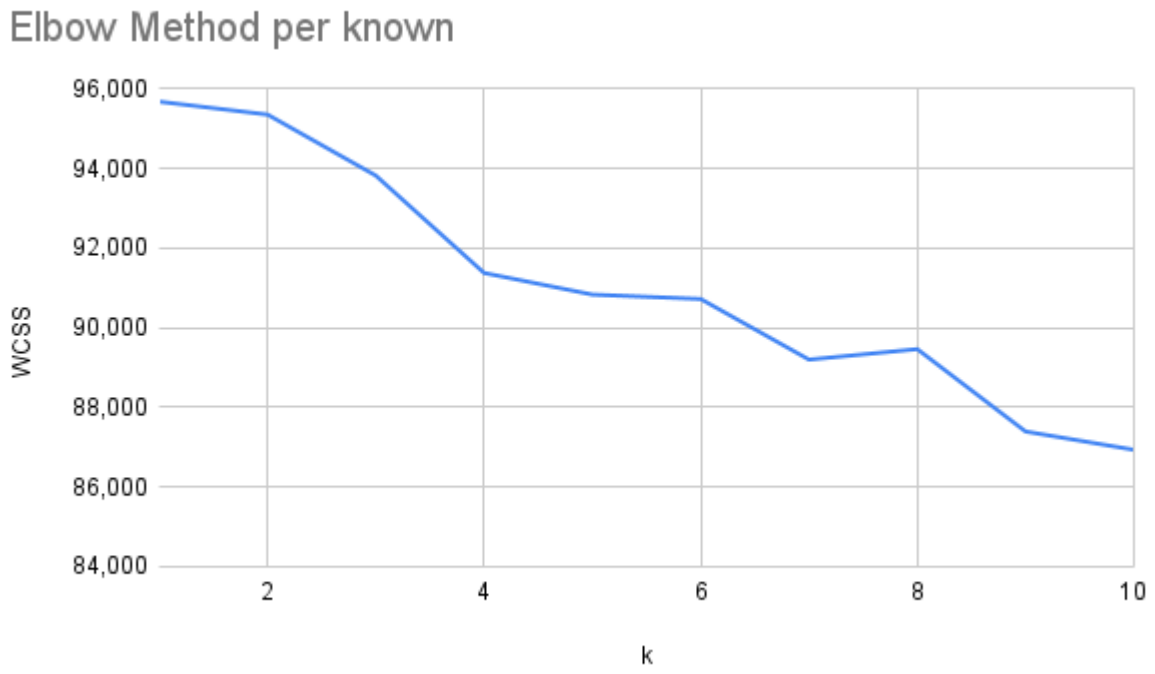
Mejoras realizadas en el algoritmo kmeans:

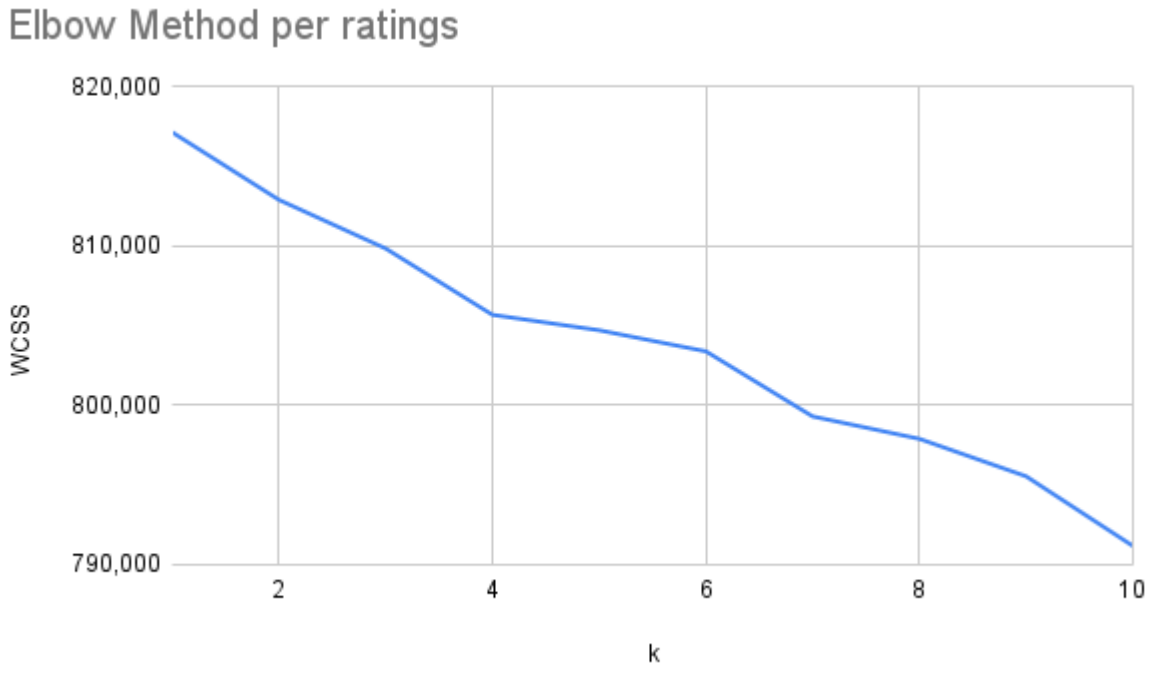
- ***Criterio de convergencia:*** Ejecutamos el algoritmo iterativamente con los centroides recalculados a partir de los usuarios que forman el cluster, siendo los nuevos centroides aquellos con menos distancia hacia los demás. Las iteraciones posteriores permiten que aquellos que ahora son más cercanos a otro centroide se cambien de clúster.
    - ***Criterio de parada:*** Cuando la diferencia de la suma de las distancias de los usuarios respecto de su centroide es igual o muy similar a la anterior (no hay mejora evidente) se para. También comprobamos los tres últimos resultados para evitar que haya ciclos de la forma a, b, a, siendo a y b la suma de las distancias.

- ***Criterio de evaluación de la k:*** Hemos creado una funcionalidad en el driver del Kmeans que nos permite experimentar la k óptima. Para ello usamos el

Elbow method el cual para cada valor de k calcula el WCSS (la raíz de la suma de los cuadrados de las distancias de cada usuario y el centroide * 100) hemos decidido multiplicar por cien debido a que nuestros números van entre 0 y 1 y si multiplicamos por cien el gráfico se veía más acentuado.

$$WCSS^{2} = \sum[(dist(u, centroide)* 100)^{2}]$$





Hemos considerado más eficiente fijar una k en vez de ir calculando cada vez que se ejecuta el algoritmo kmeans, esto lo hemos implementado así debido a que hemos observado con los dos gráficos anteriores, con distintos datasets (con 100 usuarios y con 8164 usuarios respectivamente) los cuales son de tamaños bastante dispares y hemos llegado a la conclusión que el valor óptimo de la k no parece variar. Además también hemos observado que al generar los centroides de manera aleatoria, frecuentemente dan resultados que no son los esperados y dan k óptimas que no deberían ser.
Para poder analizar los gráficos con el *Elbow Method* hay que ver la k en la que la función empieza a decrecer linealmente, entonces esa es la k óptima.

2.2 Slope One

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.

Mejoras:

- En caso que un ítem no haya sido valorado por ningún usuario del clúster, o que la intersección de los ítems valorados con el usuario sea vacia, le ponemos una valoración negativa debido a que no se puede predecir la valoración de dicho ítem. En la anterior entrega no poniamos la valoración negativa, poniamos que el mínimo fuera 0.
- En caso de que la predicción sea mayor que el máximo, hemos considerado conveniente no dejar el valor que hemos obtenido de la predicción en vez de poner el valor máximo que puede haber en las valoraciones (que es como lo teníamos en la anterior entrega). Esto lo hemos aplicado debido a que anteriormente sí había dos predicciones por encima de la puntuación

máxima, se recomendaban al mismo nivel y hemos creído conveniente que la que hemos predecido con más puntuación se recomienden antes.

También, gracias a las mejoras del Kmeans, al tener usuarios más similares entre sí en un cluster las predicciones que proporciona este algoritmo son mejores.

2.3 KNN

-Mejoras de rendimiento:

El principal motivo de estas mejoras es bajar el tiempo de ejecución del algoritmo. Para eso el camino principal que se ha recorrido ha sido intentar bajar al máximo el número de posiciones recorridas de matrices y ArrayLists. Para eso se han eliminado todos los recorridos que no eran imprescindibles y a su vez aprovechar los ineludibles para hacer el máximo de tareas a la vez.

La principal mejora en este aspecto se encuentra en la clase de K Nearest Neighbours, más concretamente en la función que implementa el algoritmo Algorithm().

Por ejemplo las matrices M de Items i Dis, que contenían todos los ítems en las filas y en la columnas otra vez los ítems en M de Ítems y las distancias entre los respectivos ítems en Dis. Ahora solo se guardan en las filas los ítems que tienen valoración en el algoritmo, es decir con los que se quieren comparar (itemsUsats). Esto se puede hacer ya que anteriormente se multiplicaban por la valoración para que no se tuvieran en cuenta los ítems que no pertenecían a estos (ya que previamente se había añadido posiciones a la ArrayList de valoraciones con valoración 0), por lo que eran prescindibles si se implementan unos pequeños cambios que se detallarán más adelante.

Esta mejora no disminuya el tiempo en el caso peor (que se quisieran comparar respecto todos los items) pero en una situación real donde el usuario habrá valorado un subconjunto mucho más pequeño de ítems respecto al total, permitirá ahorrar mucho tiempo al reducir drásticamente el número de filas (y por lo tanto aún más de columnas) en las operaciones de clonar la matriz, ordenarla, compararla…

Algunos de los pequeños cambios necesarios para este cambio son la inclusión de la función getDistanceofItem en la clase Item, que devuelve una ArrayList con las distancias respectivas al ítem solicitado, o cambiar el número de veces que se tiene que recorrer el bucle for de la función Algorithm() al tamaño de itemsUsats (lo que es en sí otra mejora más).

Gracias al cambio anterior se puede implementar otra mejora más a la función. La ArrayList de Valors en realidad tampoco era necesaria, ya que con unos cambios se podía usar la de las propias valoraciones de los itemsUsats. Esto es debido a que en la función comparar_conjunts el bucle externo que recorre las filas, ahora será del tamaño de itemsUsats, por lo que se puede usar para recorrer las valoraciones de esta ArrayList, ya que serán del mismo tamaño. Como al multiplicar por la valoración se multiplica por la valoración del ítem de la fila esto no supone un problema.

Esto supone una mejora segura, ya que se quita un recorrido de n (siendo n el tamaño del conjunto de ítems) a 0, ya que se elimina totalmente esta ArrayList y por lo tanto no hay que ir rellenando con 0 para los otros ítems.

Otra mejora que se relaciona con la anterior es que en la función comparar_conjunt, podemos aprovechar que las columnas de la matriz están ordenadas decrecientemente por la distancia (mayor distancia, items mas parecidos) para detenernos en el recorrido si la distancia es 0 .

Esta mejora nos ahorra realizar multiplicaciones por 0 en caso de que fuera el mismo ítem (solo un caso por fila) o que hayan ítems de tipos distintos, lo que es más probable en conjuntos muy grandes de ítems, lo que permitirá recorrer solo las columnas de los ítems del mismo tipo que ItemUsats.

Como es solo un pequeño cambio en la condición del for y el programa ya estaba preparado para esto, no se requiere ningún cambio adicional para su funcionamiento.

Otra gran mejora que se ha implementado es calcular las distancias de los ítems cada vez que se añadían . Así podemos ahorrarnos el ir recorriendo la matriz Distances de Conjunt de Ítems, para ver que todas las distancias estaban calculadas.

Así por un lado se consigue la matriz de Distance sea completamente independiente a la clase del algoritmo (ya que no hace falta llamar a omplir_matriu). Ademas nos ahorramos uno o varios recorridos de toda la matriz que supone cada uno un costo de n·n.

De esta mejora, se deriva una muy evidente que era eliminar del todo la función omplir_matriu y añadir ya las distancias en la función initzialitzar_matriu. Esta se usa en la creadora de conjunt_Items (hace una matriz cuadrada, de tamaño la ArrayList del parámetro de la creadora, inicializada con valores de -1) .

Así en vez de añadir todas las distancias en un principio con -1, y luego volver a recorrer la matriz para sustituirlos con la distancia posteriormente, se calculan ya en un primer momento. Además en esta nueva funcion se aprovecha la reflexión por la diagonal para no calcular las distancias dos veces (distancia de i -> j == j -> i).

Se podría ver esta matriz (la matriz Distance de conjut_Items) como una matriz de adyacencias de un grafo no dirigido con peso (la distancia entre dos ítems es la arista y su peso y los ítems los vértices). Por lo tanto, es evidente la demostración de que la diagonal hace de espejo en la matriz y de cómo la diagonal siempre será 0 (ya que no se quiere que se recomiendo un ítem a el mismo ítem).

Para añadir estas dos últimas mejoras se ha requerido un cambio en la función Distance de la clase conjunt_Items que dados dos items devuelve la distancia entre estos dos. Ahora debe comprobar que los dos ítems están en el conjunto (ya que se usa para añadir items) .

También, ya que tiene que calcular las distancias, si los dos ítems ya tienen su posición creada en la matriz (es decir no se está llamando a la función desde inicializar_matriu) se guardan sus distancias en la matriz. Esto sirve para no tener que hacerlo posteriormente o para actualizar las distancias si fuera necesario.

Gracias a este cambio de Distances, eat ya no solo sirve para consultar la distancia entre dos ítems sino que también se puede usar para actualizar la matriz si se sabe que está inicializada. El coste de este cambio es que debe consultar dos veces si existen ítems y otras dos su posición.

Como estas operaciones se hacen sobre una ArrayList ordenada y se usan operaciones dicotómicas para implementarlas solo supone un coste de $4 \cdot \log n$ (un precio justo por lo que se ahorra, aun solo teniendo en cuenta el n·n de omplir_matriu).

Por último una pequeña mejora de rendimiento, ha sido el eliminar la ordenación de los atributos de tipus_items. Ya que las operaciones que hacían uso de esta mejora no eran llamadas nunca y al pedir un atributo estos siempre se pedían en conjunto y luego se recorrían secuencialmente, esta ordenación no suponía ninguna mejora de rendimiento . Por este motivo era innecesaria y se ha podido suprimir sin problema

Además esta eliminación facilita la comprensión y programación orientadas a esta clase, como puede ser en el caso de controladores o otras clases que la hagan servir.

Esta pequeña mejora supone una reducción de coste de $m \cdot n \cdot \log n$, siendo m el número de tipus_Item, i n el número de atributos. Esto es así ya que la ordenación se realizaba con cada Ítem al insertar los atributos respectivos al tipo.

-Mejoras de las recomendaciones:

Los atributo de tipo ranged ya engloban todos los valores numéricas que se pudieran operar con ellos (enteros, naturales y decimales, todos positivos), y ya se tenían en cuenta los booleanos, fechas, cadenas de string y todo lo que no encajaba en alguno de estos se guardaba como string.

Debido a esto, no se nos ha ocurrido ningún tipo a tener en cuenta sin que este supusiera una dificultad muy grande. Algunos de estos con dificultad demasiado grande como para abarcar en los plazos de la asignatura son: reconocer url y ponerlas como no relevantes, notaciones científicas como -9.654E16, etc..

Por eso las mejoras en este aspecto han sido mínimas, ya que ya se disponía de un algoritmo bastante sólido en los otros aspectos.

La única que se nos ha ocurrido ha sido, utilizar el máximo de las valoraciones del parámetro de la función comparar_conjunts de la clase del algoritmo. Esto nos permite normalizar las valoraciones, para que a la hora de multiplicarlas por las distancias estas no cobren demasiado peso.

Esto se daba en casos en los que había distancias grandes, en estos la diferencia de multiplicar por ejemplo por 10 o 9 era bastante elevada, lo que hacía que otros ítems que alomejor tienen una distancia mayor, pero una valoración solo un poco inferior no se recomendarán.

Para arreglarlo, como hemos dicho antes hemos normalizado el valor normalizado (dividirlo entre el máximo para que de un valor entre 0 y 1) y en vez de multiplicarlo por este , la distancia se multiplica por (1 + valoración normalizada). Por lo tanto la nueva fórmula de la función comparar_conjunts queda así:

Siendo la matriz X la que contiene las distancias entre ítems y el vector V las valoraciones -> por unos Ítem i, j cualquiera :

Valors_i = Sumatorio(Xij * (1 + Vi/Max(V))

-Prueba de rendimiento

Con todas estas mejoras implementadas se ha intentado realizar el experimento de que mejora real suponen en el rendimiento. Se tiene que tener en cuenta que este experimento no se ha realizado en condiciones ideales (cambio de contexto por el sistema operativo, diferencias en el hardware en el momento como caches, paginacion virtual …,y la utilización del método System.get CurrentTimeMillis() y su posible fiabilidad, etc..).

Por los motivos explicados se tiene que considerar el experimento como una aproximación y no como una realidad absoluta. Para obtener los tiempos, se han ejecutado varias veces el DriverKND y se ha hecho la media para luego redondearlos. Los resultados son los siguientes:

Antes de las mejoras el algoritmo se ejecutaba en una media de 11 segundos y las pruebas con la versión después de las mejoras es de 3 segundos.

2.4 Hybrid Recommendation

Para hacer la hybrid recommendation hemos optado por elegir la mejor recomendación de cada método e ir alternando de uno a otro.

2.5 Evaluar Recomendación

En la versión antigua no se calculaba bien la valoración. Ahora se calcula correctamente y se ha optimizado para coger solo los valores útiles de la recomendación creada.

El algoritmo recomendador crea una lista de items ordenada de más interesante para el usuario a menos interesante. En esta lista L1 inicial se encuentran todos los ítems del sistema. Se lee una segunda lista K1 que son las valoraciones reales que daría el usuario a unos ítems concretos. K1 también está ordenada crecientemente por la valoración. L1 contiene todos los ítems de K1, pero no viceversa. Se crea una lista L2 donde meteremos todos los ítems ordenados decrecientemente de L1 que se encuentren en K1. Por lo tanto nos quedaría una lista L2 del mismo tamaño que K1 i con los mismos ítems. Ahora solo falta aplicar el algoritmo mirando posición por posición y hacer el sumatorio. De esta manera calculamos la valoración de la recomendación de una manera más precisa, ya que solo contamos con los ítems que sabemos cuál sería su valor real y no todos.

3. Documentación

3.1 Descripción de atributos y métodos de las clases del dominio.

La documentación referente al dominio se encuentra en la carpeta doxygen, en el fichero index.html. La hemos realizado en este formato para que sea más sencillo de entender la explicación de los métodos consultando su código.

3.2 Descripción de las clases de la capa de presentación

- ***AdminMainPage*** Clase que implementa las funciones que puede hacer un administrador, siendo estas cargar items, cargar usuarios y recomendaciones y eliminar usuarios o ítems.
- ***LoginView*** Clase que permite iniciar sesión o registrarte. En caso de iniciar sesión te llevará a la pantalla inicial de usuario o de administrador dependiendo del rol del usuario que ha realizado el log in.
- ***MainMenu*** Clase que muestra el menú principal para un usuario normal (no admin). Muestra 5 opciones: 3 para ver las recomendaciones (collaborative, content y hybrid), una para ver todos los ítems y otra para ver los items que el usuario ha puntuado.
- ***ProfileView*** Clase que muestra información del usuario (su id) y permite hacer log out, el cual guarda automáticamente el sistema. También tiene un botón que lleva a la vista StatsView.
- ***StatsView*** Clase que muestra datos curiosos sobre el usuario como el número de ítems que ha valorado, su valoración media y su item favorito.
- ***ShowAllItems*** Clase que contiene una lista con todos los ítems presentes en el sistema. Al clicar dos veces un Ítem este se selecciona y lleva a la vista ShowAtributtes del Item seleccionado.
- ***ShowRatedItems*** Clase que contiene una lista con todos los ítems valorados por el usuario. Al clicar dos veces un Ítem este se selecciona y lleva a la vista ShowAtributtes del Item seleccionado.

- ***ShowRecommendedItems*** Clase que contiene una lista con todos los ítems recomendados al usuario. Al clicar dos veces un Ítem este se selecciona y lleva a la vista ShowAtributtes del Item seleccionado.
- ***ShowAtributtes*** Clase que muestra todos los atributos y los valores asociados a cada atributo del Ítem seleccionado en la clase de la que se proviene. También permite valorar el ítem seleccionado y muestra la valoración si este la tenía.
- ***SignUp*** Clase que permite registrarse como usuario ingresando usuario, contraseña y repitiendo contraseña. Devuelve una pantalla de error si el usuario no está disponible o si las contraseñas no son iguales.

Además de todo lo mencionado todas las clases menos SingUp y LoginView incluyen un botón para ir a la clase ProfileView. Además las anteriores incluyendo a SignUp también tienen un botón que permite ir a la clase de la que provienen.

3.3 Descripción de las clases de la capa de persistencia

- ControladorPersistenciaItem Es la clase que se comunica con los ficheros de datos de Ítem. La primera fila es la cabecera donde están los nombres de los atributos. A partir de ahí todas las líneas son los valores de los items ordenados por los atributos previamente definidos.

```
public Vector <String> Lector_Items(String csvFile)
```

- Es la función capaz de leer de un fichero con path = csvFile. Lee los datos de los ítems y cada línea leída la mete en un String y posteriormente en un vector. Estos Strings seràn procesados posteriormente en la capa de dominio para crear los objetos correspondientes. Esta función es capaz de detectar si un fichero no existe con la excepción `FileNotExistsException(csvFile);`
  También puede detectar si el fichero que se está leyendo está vacío. `EmptyFileException("Lector_Items");`

```
public void Escritor_Items(String csvFile, Conjunt_Items list_items)
```

- Es la función capaz de escribir en un fichero con path = csvFile. Lee los datos de los items pasados por parámetros de la estructura de datos Conjunt_Items. Cada ítem obtiene sus valores y se escribe en el fichero del tipo de ítem que le toca, para clasificarlo por tipos.

- ControladorPersistencialRatings Es la clase que se comunica con los ficheros de datos de Ratings. Son ficheros que contienen valoraciones. La primera línea es la cabecera donde estará el id de usuario, el id de item y valoración. A continuación cada línea tendrá una valoración con sus datos correspondientes.

```
public ArrayList<Vector<String>> Lector_Ratings(String csvFile )
```

- Es la función capaz de leer de un fichero con path = csvFile. Lee los datos de los ratings y cada linea leida la trocea en tres Strings. UserId, ItemId, Valoración. Esto lo mete en un vector y este vector en otro para guardar todas las lineas leidas. Estos vectores serán procesados posteriormente en la capa de dominio para crear los objetos correspondientes. Esta función es capaz de detectar si un fichero no existe con la excepción `FileNotExistsException(csvFile);`
  Tambien puede detectar si el fichero que se esta leyendo esta vacío. `EmptyFileException("Lector_Ratings");`

```
public void Escritor_Ratings(String csvFile, Map<Integer, User> list_users)
```

- Es la función capaz de escribir en un fichero con path = csvFile. Lee los datos de los items pasados por parámetros de la estructura de datos Map <Integer, User>. De cada Usuario se obtienen sus valores y se escriben en el fichero.

- ControladorPersistencialRecomendation Es la clase que se comunica con los ficheros de datos de Recomendation. Son ficheros que contienen una recomendación, cada linea un id del item.

```
public ArrayList <String> Lector_Recomendation(String csvFile)
```

- Es la función capaz de leer de un fichero con path = csvFile. Lee los datos de las Recomendaciones y cada línea leída la mete en un string y posteriormente en el vector que se devolverá. Estos vectores serán procesados posteriormente en la capa de dominio para crear los objetos correspondientes. Esta función es capaz de detectar si un fichero no existe con la excepción `FileNotExistsException(csvFile);`
  También puede detectar si el fichero que se está leyendo está vacío. `EmptyFileException("Lector_Recomendation");`

```
public void Escritor_Recomendation(String csvFile, ArrayList <myPair> lista)
```

- Es la función capaz de escribir en un fichero con path = csvFile. Lee los datos de los items pasados por parámetros de la estructura de datos ArrayList <myPair>. Guardas los id de los ítems ordenadamente en el fichero.