3. Descripción de las Estructuras de Datos y algoritmos utilizados para implementar las funcionalidades principales de:

Preprocesado del conjunto de ítems:

Las funcionalidades relativas al preprocesado del dataset del conjunto de ítems se encuentran en la clase *Cjt_items*. Las funcionalidades principales son la lectura y el preprocesado del dataset de ítems y el conjunto de atributos uniforme asociado y el posterior tratamiento de cada atributo.
Los métodos y algoritmos principales de esta clase son los asociados a la inferencia del tipo de dato de cada atributo del dataset y los asociados a tratar los atributos categóricos: la decisión de mantener o descartar la información que ofrecen, etc. Se ha automatizado cada una de estas funcionalidades, aplicando algoritmos que deciden el tipo de atributo con el que estamos tratando, si un atributo de tipo string debe o no considerarse categórico,y posteriormente, determinan si la decisión de tomar el atributo por categórico es correcta en base a la cantidad de instancias únicas que toma el atributo para los distintos ítems del conjunto. Para los juegos de datos de los que disponemos, los métodos infieren correctamente todos los tipos de datos para los distintos atributos y descartan correctamente los atributos no categóricos como título de película, links, etc...

La lectura del dataset se realiza con la librería scanner, que nos permite escoger un delimitador de línea y se aplica posteriormente el tratado de la línea, en nuestro caso para separar en columnas de atributos, utilizamos la coma como delimitador.

En adición a los métodos que la librería ofrece, ha sido necesario principalmente construir un algoritmo que identifica la columna que nos indica el identificador de ítem (utilizando ***regular expressions***) y un algoritmo que trate los casos en los que un atributo en cierta línea del dataset contiene un carácter *newline*. Este caso lo detectamos cuando al separar la línea en sus columnas, el número de atributos es menor al detectado en la lectura inicial de los identificadores de columna, en este caso se hará append de tantas posteriores líneas como haga falta para igualar el número de atributos.

Para la inferencia de tipo, para cada tipo de datos hay una función que decide si el parámetro string pertenece al tipo - y el orden en qué se llaman es importante para identificar correctamente el tipo de dato del atributo. Para implementar estas funciones se han utilizado principalmente *regular expressions,* que han permitido incluso identificar los valores enteros que se atribuyen a *doubles* debido a que se salen del rango de representación del tipo *int* de java.

El tratamiento de las variables categóricas ha requerido una complejidad significativa, debido a la información que se debe mantener durante la lectura del dataset para determinar finalmente si se descarta o mantiene el atributo categórico.

Los métodos y algoritmos principales asociados al tratamiento de los atributos categóricos utilizan contenedores TreeMap. Esta decisión la he tomado en base a las columnas, que se utilizan como identificador para los atributos categóricos asociados, se tratan en orden, y por lo tanto, además de aprovechar la eficiencia en términos de espacio del TreeMap, se aprovecha que la mayoría de inserciones serán *best-case*.

A lo largo de la lectura, se decide para cada categoría instanciada en los atributos categóricos del conjunto, un valor que la identifica únicamente en la columna/atributo correspondiente. Esto nos permitirá implementar el One-Hot Encoding de los atributos que finalmente se deciden tomar por categóricos.

Para decidir, en una primera lectura de la totalidad del dataset, si un atributo es potencialmente categórico, tomamos un límite de palabras que puede tener (si es una única categoría), y si el string tratado se identifica como lista (contiene ‘;’ o ‘,’), se descartan los atributos que contengan algún elemento de la lista cuya cantidad de palabras se desvía significativamente de la media de palabras entre todos los elementos de la lista, pues esto nos indica que muy probablemente se trata de un texto descriptivo, donde no existe ninguna relación entre las strings obtenidas de identificar los separadores. En esta fórmula para descartar, se tiene en cuenta que cuanto menor es la media, mayor desviación se permite: así si la media de palabras de los elementos de la potencial lista de categorías es 1, se permiten elementos con máximo 2 palabras.

Posteriormente a esta decisión, y una vez se dispone de todos los conjuntos de atributos categóricos del dataset, se decide, en base al número de categorías únicas con respecto a la unión de todas las categorías instanciadas en el conjunto, que si este ratio supera un determinado threshold (0.9 ofrece resultados ideales para los datasets ofrecidos en la asignatura), se descarta ese atributo categórico ya que se trataría de un atributo categórico absurdo: títulos, sinopsis, links...

Content based recommender:

La funcionalidad principal de un *content based recommender* es, en base a los ratings que hace un usuario de ciertos ítems, ofrecer un subconjunto de sugerencias o recomendaciones. Los métodos asociados a este algoritmo recomendador se encuentran en la clase ***ContentBased***.

Una de las estructuras de datos principales utilizada es el HashMap, donde se guardan para un usuario determinado los ítems que ha valorado. Se consideró que esta era la mejor estructura ya que no se requiere de ningún tipo de ordenación y en cuanto a eficiencia tiene una complejidad *O(1)* para los accesos e inserciones.

El algoritmo principal que se usa es el "k-nearest-neighbour". Este algoritmo se aplica entre ítems, de forma que dado un cierto ítem se calculan los k vecinos más cercanos. Para implementarlo se precisa de un algoritmo que calcule la distancia o similitud entre ítems. Esto es lo que hace la clase ***DistanceItems***.

La distancia entre ítems se calcula mediante la distancia euclidiana. El procedimiento es el siguiente, dados un par de ítems a i b que pertenecen al mismo dataset, se calcula la distancia euclidiana entre los subconjuntos de atributos del mismo tipo. Es decir se cogen los atributos de tipo integer de a y de b, y mediante la distancia euclidiana se obtiene su similitud, luego lo mismo con los de tipo double, y así con todos los demás. Para poder calcular la distancia euclidiana primero hay que obtener la similitud para cada par de atributos de la misma columna del dataset. Esto se hace de forma diferente según su tipo. Para integers y doubles se hace la diferencia dividida por la máxima diferencia que puede haber en dicho atributo, para los de tipo date los mismo pero con solo el año (dos datas con el mismo año se consideran iguales), para los booleanos solo se obtiene 0 o 1 en función de si son iguales o diferentes respectivamente y para los de tipo categórico se hace el cardinal de la intersección entre el cardinal de la unión. Una vez se obtiene la similitud de cada par de atributos de un mismo tipo se hace la raíz cuadrada de suma de sus cuadrados dividido por el número de atributos de este mismo tipo, es decir normalizando para así obtener un número entre 0 y 1. En el cálculo de la distancia total se hace una suma ponderada de las distancias euclidianas entre cada tipo de atributo, dándole más peso a los de tipo categórico, seguido de los de tipo entero, double y date que reciben el mismo peso y por último los de tipo booleano.

Para el algoritmo que computa los ***k vecinos más próximos*** de un ítem a, se itera por todos los demás ítems del dataset calculando la distancia. A partir de esta distancia y con la valoración media del ítem del dataset así como la valoración hecha por el usuario que pide la recomendación del ítem se hace una predicción de la valoración que daría el usuario a el ítem del dataset. Una vez procesados todos los ítems del dataset nos quedamos solo con los k que tienen una mejor predicción de valoración y los devolvemos en un diccionario con los ítems y sus valoraciones predecidas.

En la implementación se usa un SortedMap de tamaño k que guarda siempre las k mejores valoraciones predecidas hasta el momento con una lista de todos los ítems que tienen esta

misma predicción. Aunque el coste de la inserción es logarítmico, el coste de obtener el elemento mínimo es constante por lo que en caso peor tendríamos O(nlogk) donde n es el tamaño del dataset. Al terminar de procesar el dataset se hace una conversión de este SortedMap a un HashMap para lidiar lidiar con los items que tiene misma predicción y asegurar que se devuelven solo k. Se devuelve este HashMap.

Usando esto, el método *recommended_items* de la clase ***ContentBased*** es el que se encarga de por cada ítem valorado por el usuario, buscar sus k-NN y al final quedarse con los k con mejor predicción. Estos serán los ítems recomendados que nos datá el Content Based.

En este caso se usa un HashMap cuyo tiempo en caso peor es un poco más elevado, O(nk), pero la decisión viene justificada por el hecho de que en este caso las *keys* son los ítems y una ordenación por ítem no nos beneficiaría en nada. Si se usase un SortedMap con la predicción como *key* aumentaría el coste en la función que nos calcula el k-NN.

K-means:

K-means es un algoritmo de clusterización que agrupa objetos (en nuestro caso usuarios) en *k* grupos (clusters) basándose en sus características. Cada cluster tiene un centroide, que viene a ser la posición promedio de los objetos pertenecientes a dicho grupo. Cada usuario valora uno o varios ítems y en base a eso realizamos el agrupamiento calculando la distancia entre el usuario y el centroide. La distancia entre el usuario y el centroide se calcula mirando las valoraciones de los ítems que tienen en común el usuario y el centroide.

El algoritmo consta de tres pasos:

1. **Inicialización:** una vez escogido el número de clusters, *k*, se establecen *k* centroides (que pueden ser escogidos al azar o de cualquier otra forma).
2. **Asignación de los usuarios a los centroides:** cada usuario es asignado a su centroide más cercano.
3. **Actualización de los centroides:** se actualiza la posición del centroide de cada cluster tomando como nuevo centroide la posición promedio de los usuarios pertenecientes a dicho cluster.

Se repiten los pasos 2 y 3 hasta que los centroides no se mueven.

Slope one:

Slope one es un algoritmo de predicción numérica basada en la relación que la valoración de un usuario tiene con las demás, o lo que es lo mismo: es un algoritmo que, para un conjunto de valoraciones de unos usuarios, nos permite predecir cuáles podrían ser las valoraciones de nuestro usuario actual. SlopeOne es una parte constitutiva del algoritmo de ***Collaborative Filtering*** y se aplica tras la computación del KMeans.

El algoritmo aquí tratado plantea lo siguiente: dado que las predicciones se realizan en base a los datos ya existentes de valoraciones a objetos (sean estos objetos cualesquiera, homogéneos entre ellos), cabe tener en consideración todo aquello que nos pueda resultar de utilidad para determinar cual de estos puede ser más provechoso para nuestro usuario actual. De esta manera, las medidas claramente más útiles para medir esta relación son la frecuencia con que dos objetos se valoran, y la diferencia de valoraciones que estas mismas reciben.

Mediante las métricas planteadas, hay que computar, para todos los items existentes, la relación que estos mantienen con todos los demás, y a partir de esta casuística podremos encontrar qué objetos pueden gustar más a nuestro usuario actual. La manera de proceder seria tal que:

-Debemos conocer los datos de valoraciones de objetos (tanto el valor de la valoración como la cantidad total que tenemos, a nivel global, de valoraciones)

-Debemos computar la relación que los ítems mantienen entre ellos (mediante las métricas ya descritas)

-Debemos considerar, para todos los ítems que valore el usuario actual, cuáles se adecuan más a sus gustos. Este paso implica la búsqueda de aquellos ítems que se adecuen a las valoraciones del usuario y, además, el cálculo promedio de cuál sería el valor promedio que nuestro usuario les daría a dichos objetos.

El algoritmo, tal y como ha quedado implementado, requiere de un paso previo al cómputo descrito: la transmutación de los datos en una estructura usable para el algoritmo. Esto nos lleva a hablar de las estructuras usadas: principalmente se ha tratado con HashMap, estructura generalmente eficiente para acceder a un conjunto dado de llaves y sus respectivos valores (acelera el acceso a aquellos datos y valores de interés durante la aplicación del algoritmo), además de usar también Lists (más concretamente ArrayLists), por el simple hecho de tener almacenado en alguna estructura (con relativa facilidad de acceso) toda aquella información sobre los ítems del conjunto, los usuarios con los que se trata y las valoraciones que estos hayan podido realizar.

La gracia de este algoritmo, para el problema que tratamos, es la siguiente: la aplicación del KMeans presupone la creación de clusters, o lo que es lo mismo, de una clasificación de un conjunto de usuarios en subconjuntos, con lo cual reducimos el margen de error a la hora de realizar recomendaciones; una vez creados los clusters, sabiendo que estamos buscando una recomendación para un usuario determinado, elegimos el cluster en el que nuestro usuario actual ha sido clasificado y dicho espacio de búsqueda es enviado al SlopeOne. Aquí entra en juego el cómputo de la relación entre ítems y sus valoraciones: es

desde este punto en que se realizan las predicciones de las valoraciones que nuestro usuario actual podría realizar.

De manera breve, cabe destacar lo siguiente: la clase que implementa dicho algoritmo hace uso de 4 estructuras de datos (HashMaps) para almacenar información de la predicción, además de 4 métodos de cálculo. Por un lado, tenemos 2 matrices (de dimensiones #items**x**#items), encargadas de almacenar la frecuencia y diferencia de valoración entre ítems, y 2 matrices de almacenamiento de valoraciones de los usuarios para un conjunto determinado de ítems (una de estas, inputData, son los datos de entrada del problema; la otra, outputData, es la que almacena las predicciones generadas para un usuario o, en su defecto, para un conjunto de usuarios determinado). Por otro lado, tenemos 4 métodos para realizar el cómputo de las predicciones: la preparación de los datos de entrada en un HashMap; la preparación de las matrices de diferencia y frecuencia para ser usadas en el cálculo de las predicciones; finalmente el cálculo propiamente dicho, esto es, la generación de las predicciones para un usuario dado.