UNIVERSITAT POLITÈCNICA DE CATALUNYA

SISTEMA RECOMANADOR

DOCUMENTACIÓ - PRIMERA ENTREGA - v1.0

PROJECTES DE PROGRAMACIÓ

Autors

HECTOR Pueyo Casas
(Hector.Pueyo@estudiantat.upc.edu)

AGNÈS FELIP I DÍAZ
(agnes.felip@estudiantat.upc.edu)

MIQUEL FLORENSA
(miquel.florensa@estudiantat.upc.edu)

DAVID LATORRE
(david.latorre.romero@estudiantat.upc.edu)

Supervisor

SERGIO ÁLVAREZ NAPAGAO

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

1. CASOS D'ÚS 3
1.1 ESQUEMA 3
1.2 EXPLICACIÓ 4
2. DIAGRAMA UML 12
2.1 ESQUEMA 12
2.2.1 Usuari 13
2.2.1.1 UActiu 13
2.2.2 Item 13
2.2.3 Valoració 14
2.2.4 Atribut 14
2.2.4.1 AtrBool 14
2.2.4.2 AtrString 15
2.2.4.3 AtrlInt 15
2.2.4.4 AtrDouble 15
2.2.5 TipusAtribut 15
2.2.6 Distancia 16
2.2.6.1 DistItem 16
2.2.6.2 DistUsuari 16
2.2.7 Algorisme 17
2.2.7.1 CBFiltering 17
2.2.7.2 CFiltering (kMeans, SlopeOne) 17
3. DESCRIPCIÓ DELS ALGORISMES 18
3.1 COLLABORATIVE FILTERING 18
3.1.1 DISTÀNCIES ENTRE USUARIS 18
3.1.2 K-MEANS 19
3.1.3 SLOPE ONE 20
3.2 CONTENT-BASED FILTERING 21
3.2.1 DISTÀNCIES ENTRE ITEMS 21
3.2.2 K-NEAREST NEIGHBOURS 23
4. JUSTIFICACIÓ DE LES ESTRUCTURES DE DADES 25
4.1 COLLABORATIVE FILTERING 25
4.2 CONTENT-BASED FILTERING 27
5. RELACIÓ DE LES CLASSES IMPLEMENTADES PER CADA MEMBRE DE L'EQUIP 29

1. CASOS D'ÚS

1.1 ESQUEMA

1.2 EXPLICACIÓ

Per aquesta primera entrega, és important destacar que, pel fet que totes les classes que formen el programa només s'han avaluat de forma independent, ja que encara no s'han implementat controladors (només s'han implementats drivers que comproven i avaluen el funcionament dels algorismes junt amb la classes de distàncies conjuntament, la resta són controladors independents de les classes), que la majoria d'aquests casos d'ús no estaran disponibles fins que el programa no disposi d'almenys una interfície visual.

A més, també se'ns presenten alguns casos d'ús que queden per determinar, i que seran determinats durant el procés de creació de controladors i interfície visual

Descripció del cas d'ús Sortir

Nom: Sortir

Actor: Usuari

Comportament:

- L'usuari pot sortir del programa des de qualsevol punt.
- El Sistema guarda l'últim conjunt de recomanacions
- Sistema fa Logout. [Logout]

Errors possibles i cursos alternatius:

Descripció del cas d'ús Login

Nom: Login

Actor: Usuari

Comportament:

- L'usuari proporciona un nom d'usuari i una contrasenya.
- El sistema valida l'existència d'un usuari amb el nom indicat.
- El sistema valida que la contrasenya sigui vàlida pel nom d'usuari indicat.
- El sistema Carrega les recomanacions. [Carrega recomanacions]

Errors possibles i cursos alternatius:

- L'usuari no existeix. El sistema informa de l'error i suggereix la possibilitat de registrar-se.
- La contrasenya no coincideix amb el nom d'usuari. El sistema informa de l'error.

Descripció del cas d'ús Crear Perfil

Nom: Crear Perfil

Actor: Usuari

Comportament:

- L'usuari proporciona un nom d'usuari, una contrasenya i una confirmació de contrasenya.
- El sistema fa una valoració dels valors donats i crea un nou perfil amb el nom d'usuari indicat.

Errors possibles i cursos alternatius:

- En cas de diferència entre la contrasenya i la seva confirmació, el sistema informa de l'error i demana un altre cop la confirmació de contrasenya.

Descripció del cas d'ús Eliminar Perfil

Nom: Eliminar Perfil

Actor: Usuari

Comportament:

- L'usuari pot eliminar el seu perfil.
- El sistema espera una confirmació de l'usuari.
- L'usuari ha de confirmar l'eliminació del seu perfil.
- El Sistema dóna de baixa la instància.

Errors possibles i cursos alternatius:

Descripció del cas d'ús Modificar Perfil

Nom: Modificar Perfil

Actor: Usuari

Comportament:

- L'usuari pot modificar totes les seves dades.

Errors possibles i cursos alternatius:

Descripció del cas d'ús Logout

Nom: Logout

Actor: Usuari

Comportament:

- L'usuari pot tancar la sessió actual al programa.
- El sistema guarda l'últim conjunt de recomanacions. [Guardar Recomanacions]

Errors possibles i cursos alternatius:

Descripció del cas d'ús Consultar Usuari [PER DETERMINAR]

- **Nom:** Consultar Usuari
- **Actor:** Usuari
- **Comportament:**
  - Des de la busca d'usuaris, és possible consultar-ne un, és a dir, obtenir informació sobre el seu nom, així com items valorats i aquestes valoracions.

- **Errors possibles i cursos alternatius**

Descripció del cas d'ús Navegar [PER DETERMINAR]

- **Nom:** Navegar
- **Actor:** Usuari

- **Comportament:**
  - Acció que permet buscar ítems, i usuaris, introduint el nom d'aquests.

- **Errors possibles i cursos alternatius**

Descripció del cas d'ús Buscar Usuari per Nom [PER DETERMINAR]

- **Nom:** Buscar Usuari per nom
- **Actor:** Usuari

- **Comportament:**
  - L'usuari informa del nom d'usuari
  - El sistema retorna una llista d'ids d'usuaris que siguin o s'assemblin al nom.
  - El sistema dóna la possibilitat de consultar algun d'aquests usuaris.
- **Errors possibles i cursos alternatius**
  - No s'obté cap resultat perquè no existeixen usuaris similars i per tant es retorna una llista buida, el sistema avisa a l'usuari de l'error.

Descripció del cas d'ús Buscar Ítem per Nom [PER DETERMINAR]

- **Nom:** Buscar Ítem per Nom
- **Actor:** Usuari

- **Comportament:**
  - L'usuari informa del nom del ítem
  - El sistema retorna una llista d'ids d'item que tinguin el nom o s'assemblin al nom.
  - El sistema dóna la possibilitat de consultar algun d'aquests usuaris.
- **Errors possibles i cursos alternatius**
  - No s'obté cap resultat perquè no existeixen ítems similars i per tant es retorna una llista buida, el sistema avisa a l'usuari de l'error.

Descripció del cas d'ús Consulta Ítem

- **Nom:** Consulta Ítem
- **Actor:** Usuari, Sistema

- **Comportament:**
  - El sistema comprova que l'ítem que es vol consultar està present al conjunt de dades.
  - El sistema proporciona la informació (ja sigui al propi sistema, o a l'usuari actiu (per mitjà de la interfície gràfica)).

- **Errors possibles i cursos alternatius:**
  - En cas que l'Ítem el qual s'ha clicat per consultar ja no existeixi al conjunt de dades (ja sigui perque la recomanació estigui desactualitzada, etc.) el sistema informa d'error

Descripció del cas d'ús Gestió Ítem

- **Nom:** Gestió Ítem
- **Actor:** Usuari, Sistema

- **Comportament:**
  - Cas d'ús que engloba la Creació, Edició, i Eliminació d'un Ítem

- **Errors possibles i cursos alternatius:** Cap

Descripció del cas d'ús Elimina Ítem

• **Nom:** Elimina Ítem

• **Actor:** Usuari

• **Comportament:**

- L'usuari decideix eliminar l'Ítem el qual està consultant.
- El sistema l'elimina del conjunt de dades amb el qual s'està treballant.

• **Errors possibles i cursos alternatius:**

- En aquell cas en que l'usuari vulgui eliminar un ítem el qual no ha creat ell, es retornarà error.

Descripció del cas d'ús Crea Ítem

• **Nom:** Crea Ítem

• **Actor:** Usuari

• **Comportament:**

- L'usuari, proporciona les dades del nou ítem (id, títol, atributs, altre info).
- El sistema introdueix aquest al conjunt de dades amb el qual s'està treballant.

• **Errors possibles i cursos alternatius:**

- En cas que l'usuari vulgui crear un ítem amb un id ja existent, es retornarà un error.

Descripció del cas d'ús Edita Ítem

• **Nom:** Edita Ítem

• **Actor:** Usuari

• **Comportament:**

- L'usuari, que vol editar l'ítem el qual està consultant, és capaç de modificar les dades de l'item (títol, atributs, altre info) mitjançant la interfície gràfica.
- El sistema actualitzarà aquest ítem al present conjunt de dades.

• **Errors possibles i cursos alternatius:**

- En cas que l'usuari modifiqui el títol de l'ítem, i aquest nou títol coincideixi amb el d'un altre ítem ja present al conjunt de dades, es retornarà error.

Descripció del cas d'ús Consulta Valoracions

- **Nom:** Consulta Valoracions
- **Actor:** Usuari, Sistema

- **Comportament:**
  - L'usuari, que està consultant l'ítem del qual en vol obtenir les valoracions, o el sistema (que de cara a la creació de recomanacions vol consultar les valoracions), reben les valoracions (nota + arguments) d'un ítem.

- **Errors possibles i cursos alternatius:** Cap

Descripció del cas d'ús Gestió Valoracions

- **Nom:** Gestió Valoracions
- **Actor:** Usuari, Sistema

- **Comportament:**
  - Cas d'ús que engloba la creacio, modificació, eliminació i valoració de les valoracions pels ítems del conjunt de dades present.

- **Errors possibles i cursos alternatius:** Cap.

Descripció del cas d'ús Crear Valoració

- **Nom:** Crear Valoració
- **Actor:** Usuari

- **Comportament:**
  - L'usuari proporciona les dades de la nova valoració (nota + arguments).
  - El sistema introdueix aquest al conjunt de dades de valoracions amb el que s'està treballant.

- **Errors possibles i cursos alternatius:**
  - En el cas que l'usuari vulgui crear una valoració per un ítem del qual ja n'ha fet una, es retornarà error.

Descripció del cas d'ús Elimina Valoració

- **Nom:** Elimina Valoració
- **Actor:** Usuari

- **Comportament:**
  - L'usuari decideix eliminar la valoració la qual està consultant.
  - El sistema la elimina del conjunt de dades de valoracions amb el qual s'està treballant.

- **Errors possibles i cursos alternatius:**
  - En el cas que l'usuari vulgui eliminar una valoració que no ha fet, es retornarà error.

Descripció del cas d'ús Consulta Recomanacions

- **Nom:** Consulta Recomanacions
- **Actor:** Usuari

- **Comportament:**
  - L'usuari, que està a la pàgina principal, consulta les recomanacions fetes pel sistema.
  - El sistema mostra certes recomanacions per un usuari concret.

- **Errors possibles i cursos alternatius:** Cap

Descripció del cas d'ús Gestió Recomanacions

- **Nom:** Gestió Recomanació
- **Precondició:**
  - L'usuari està consultant les recomanacions
- **Actor:** Usuari i Sistema

- **Comportament:**
  - L'usuari recarrega les recomanacions.
  - El sistema guarda i crea les recomanacions.

- **Errors possibles i cursos alternatius:** Cap

Descripció del cas d'ús Guardar Recomanacions

- **Nom:** Guardar Recomanacions
- **Actor:** Sistema
- **Precondició:**
  - L'usuari ha LogOut o ha tancat l'aplicació.
- **Comportament:**
  - El sistema guarda les recomanacions fetes a l'usuari actiu.

- **Errors possibles i cursos alternatius:** Es tanca l'ordinador sense sortir de l'aplicació, que pasa?

Descripció del cas d'ús Carregar Recomanacions

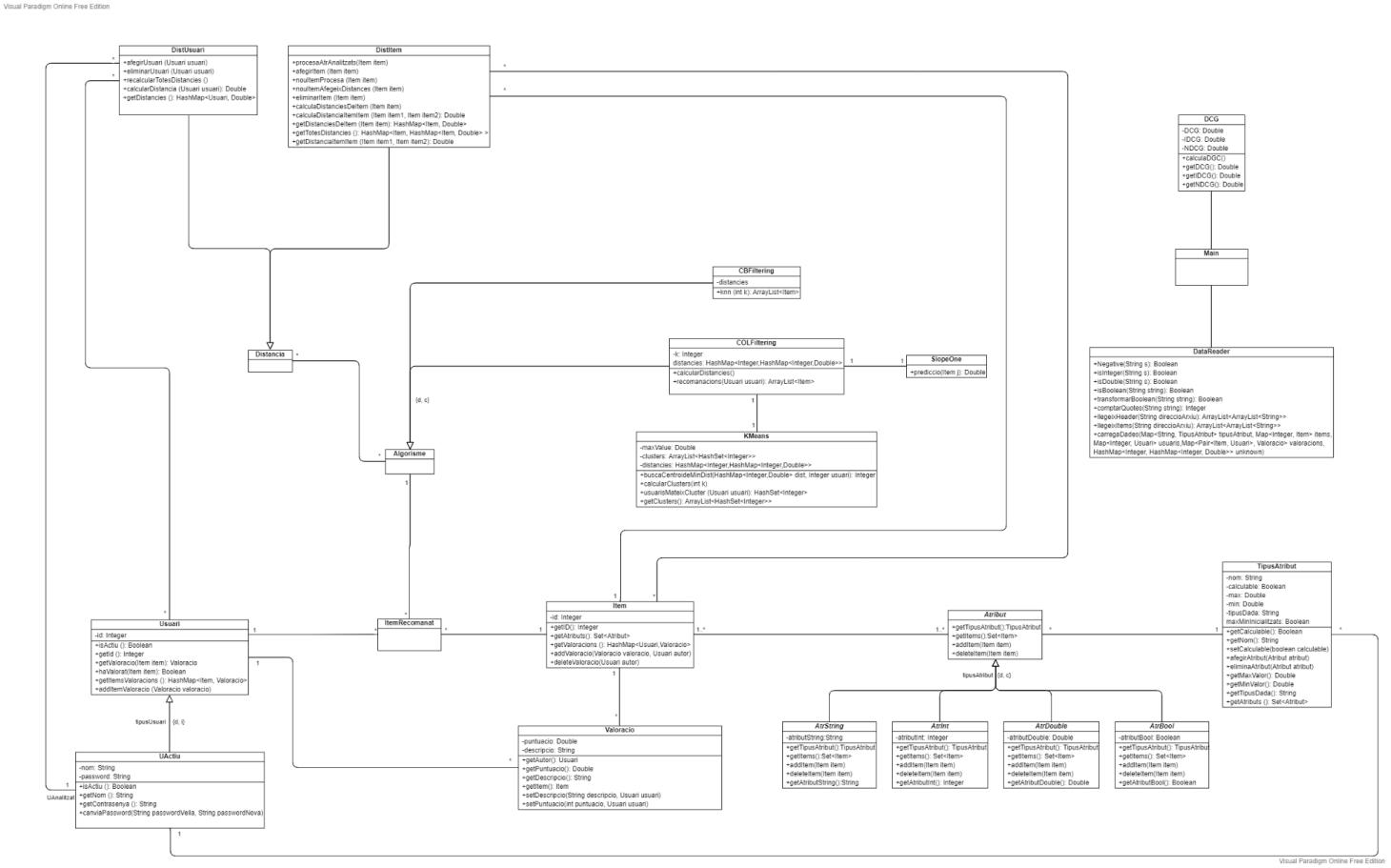
- **Nom:** Carregar Recomanacions
- **Actor:** Usuari, Sistema

- **Precondició:**
  - L'usuari fa login
  - L'usuari refresca les recomanacions.
  - El sistema acaba de crear una recomanació.
- **Comportament:**
  - El sistema mostra la primera recomanació en cas que s'acabi de crear l'usuari.
  - El sistema mostra les mateixes recomanacions que ja hi havia ja que no hi ha hagut canvis que afectin al sistema recomanador.
  - El sistema mostra unes noves recomanacions degut a canvis de informació necessària pel sistema recomanador.

- **Errors possibles i cursos alternatius:** Cap

2. DIAGRAMA UML

2.1 ESQUEMA



***Restricció:** Donat un TipusAtribut en que tipusDada == (“Double” || “Bool” || “Int”), un ítem només podrà estar associat amb un atribut d’aquest TipusAtribut. Això, no té perquè complir-se en cas que el TipusAtribut tingui tipusDada == “String”.

2.2 EXPLICACIÓ DE LES CLASSES

2.2.1 Usuari

Un usuari està identificat pel seu identificador “id”, que és únic. Un usuari pot ser actiu (veure la classe UActiu) o no. En el cas que no ho sigui, com que llavors serà un usuari que no pot iniciar sessió, ja que d’ell només en sabem les valoracions que ha fet a partir de fitxers de dades externs, no necessitem crear ni emmagatzemar un nom per aquests.

Un usuari està associat a totes les seves valoracions, és a dir valoracions que ha creat aquest. Per tant, podem saber quins items a valorat i quins no. **Un usuari només pot fer una valoració sobre un determinat ítem.**

2.2.1.1 UActiu

Subclasse de la classe Usuari. Es tracta d’un usuari que podriem dir que “té vida pròpia”. Es tracta d’un usuari que no ha estat automàticament per cap fitxer de dades, i per tant es pot iniciar sessió amb ell. Aquest, per tant, a part de tenir un identificador que el diferència de la resta d’usuaris, també té un nom i una contrasenya.

2.2.2 Item

Es tracta d’un ítem, el qual principalment és genèric, ja que segons amb les dades en que treballem podrà representar una cosa o l’altre (pel·lícula, sèrie, etc).

Per tant, un ítem només estarà identificat per un identificador “id”. No obstant, aquest ítem estarà relacionat amb una sèrie d’atributs (classe Atribut) que faran per tant que es conegui informació sobre aquest, i que per tant es pugui comparar amb els altres items.

Un ítem també està relacionat amb totes aquelles valoracions que s'han fet d'ell, i per tant també amb els usuaris que han creat aquestes valoracions.

2.2.3 Valoració

Una valoració s'ha fet de cara a criticar/avaluar un ítem, i per tant depèn d'aquest, i de l'usuari que l'ha creat.

A més, una valoració té una puntuació (un número del 0 al 5), i una descripció opcional, en cas que l'usuari que la crea vulgui explicitar més informació sobre la seva crítica.

2.2.4 Atribut

Un atribut és una classe abstracta que ens aporta informació de cara a analitzar/conèixer sobre un ítem. Un atribut, per tant, està associat a aquells ítems dels quals el disposen, i també està associat al tipus d'atribut (de la classe TipusAtribut) del qual en forma part.

Hi ha 4 subclasses d'atribut: AtrBool, AtrString, AtrInt, AtrDouble.

2.2.4.1 AtrBool

Subclasse atribut, que representa un atribut booleà. Per tant, està formada per un booleà. D'aquesta manera, per tant, si l'atribut és per exemple del TipusAtribut "Adult", i el seu valor (el del atribut) és fals, llavors voldrà dir que tots aquells items que estiguin relacionats amb aquest atribut seran items que no són de la categoria adult.

2.2.4.2 AtrString

Subclasse atribut, que representa un atribut String. Per tant, està formada per un String. D'aquesta manera, per tant, si l'atribut és per exemple del TipusAtribut "Gènere", i el seu valor (el de l'atribut) és "Ciència Ficció", llavors voldrà dir que tots aquells items que estiguin relacionats amb aquest atribut formaran part del gènere ciència ficció.

2.2.4.3 AtrlInt

Subclasse atribut, que representa un atribut Integer. Per tant, està formada per un Integer. D'aquesta manera, per tant, si l'atribut és per exemple del TipusAtribut "Any", i el seu valor (el de l'atribut) és "2021", llavors voldrà dir que tots aquells items que estiguin relacionats amb aquest atribut seran de l'any 2021.

2.2.4.4 AtrDouble

Subclasse atribut, que representa un atribut Double. Per tant, està formada per un atribut Double.

D'aquesta manera, per tant, si l'atribut és per exemple del TipusAtribut "vote_average", i el seu valor (el de l'atribut) és "5.3", llavors voldrà dir que tots aquells items que estiguin relacionats amb aquest atribut hauran obtingut una votació mitjana de 5.3.

2.2.5 TipusAtribut

TipusAtribut ens dóna a conèixer quin tipus d'informació podem interpretar a partir dels atributs que estan relacionat a aquest. És per això que la classe TipusAtribut conté un String "nom".

Per tant, si un TipusAtribut té en "nom": Any; llavors sabrem que tots els atributs associats a aquest TipusAtribut representen un any.

Un TipusAtribut, a més, està format per un String "tipusDada", que ens informa de quina subclasse Atribut son els atributs amb els que està associat, ja que un TipusAtribut no pot estar associat amb atributs de subclasses diferents d'Atribut (un TipusAtribut que té "nom": gènere; tindrà "tipusDada": String. Mentre que un TipusAtribut que té "nom": Any; tindrà "tipusDada": Int).

A més, la classe TipusAtribut també està formada per un booleà "calculable", el qual ens farà saber si podrem utilitzar els atributs associats amb aquest TipusAtribut per comparar i calcular distàncies entre items. Per exemple, aquells atributs que estiguin associats amb el TipusAtribut "imdb_id" no poden ser calculats per comparar distancies (similitud) entre items, mentre que aquells que estiguin associats amb el TipusAtribut "Gènere" si.

Cal destacar que, amb la intenció del càlcul de distàncies entre items, per aquelles instàncies de TipusAtribut en que tipusDada == Int; o tipusDada == Double; la classe també haurà de guardar el valor mínim entre tots aquells atributs amb els que està associat, i també el valor màxim.

2.2.6 Distancia

Classe que calcula les distàncies entre usuaris (un usuari respecte un altre), i distàncies entre ítems (un item respecte un altre). Quan parlem de distàncies, ens referim a similitud, ja sigui entre usuaris o ítems.

2.2.6.1 DistItem

Subclasse de la classe Distància, que calcula la distància entre diversos items. Més informació sobre aquesta classe a l'apartat 4.2.1.

2.2.6.2 DistUsuari

Subclasse de la classe Distància, que calcula la distància entre usuaris. Més informació sobre aquesta classe a l'apartat 4.1.1.

2.2.7 Algorisme

Classe que engloba els algorismes que s'utilitzen per crear les recomanacions d'items per un usuari.

2.2.7.1 CBFiltering

Subclasse de la classe algorisme que s'encarrega de crear recomanacions d'items a un usuari amb el mètode "Content-based Filtering". Més informació sobre aquesta classe a l'apartat 4.2.

2.2.7.2 CFiltering (kMeans, SlopeOne)

Subclasse de la classe algorisme que s'encarrega de crear recomanacions d'items a un usuari amb el mètode "Collaborative Filtering". Més informació d'aquest algorisme (kMeans i SlopeOne) a l'apartat 4.1.

3. DESCRIPCIÓ DELS ALGORISMES

3.1 COLLABORATIVE FILTERING

3.1.1 DISTÀNCIES ENTRE USUARIS

Per tal de calcular la similitud entre els usuaris existents (ja siguin tant actius, com no actius) ho fem amb la classe DistUsuari.

Primer de tot, cal destacar que aquesta classe no està pensada per calcular la distància (similitud) entre tots els usuaris d'entre un conjunt, sinó de la distància des d'un usuari determinat, cap a tots els altres (és a dir, cap aquells d'un conjunt), però de forma independent (és a dir una distància seria la que va de l'usuari determinat fins a un del conjunt; una altre seria la que va de l'usuari determinat fins a un altre del conjunt...).

Per tant, per una banda, aquesta classe està associada a un determinat usuari: Usuari "UAnalitzat". Primerament, la classe no hi posa cap restriccio sobre si aquest usuari hagi de ser actiu o no, tot i que el més lògic és que aquest ho sigui, ja que quan el programa estigui completament implementat, i es calculi una recomanació, aquesta es calcularà per a l'usuari que ha iniciat la sessió, i per tant, per a un usuari actiu (UActiu).

I, per l'altre banda, la classe també estarà associada a un conjunt d'usuaris, aquells amb els quals es calcularà la distància respecte UAnalitzat.

La distància entre un usuari i un altre té un valor que és un número, més exactament un Double. És per això que la classe enmagatzema en un HashMap la distància que hi ha entre l'usuari UAnalitzat i un del conjunt donat.

Ara bé, com calculem aquesta distància? Ho fem calculant la **distancia euclídea** entre els dos usuaris, segons les seves valoracions. És a dir, imaginem que volem calcular la distància entre UAnalitzat, i un altre usuari que anomenarem $Usuari_2$, llavors:

- Primerament tindrem un double anomenat **suma**, on hi guardarem la diferència entre els dos usuaris. Primerament, aquest double l'inicialitzarem a 0.

- LLavors, per cada ítem que hagi valorat UAnalitzat, mirarem si aquest també l'ha valorat $Usuari_2$. En cas que, per a un ítem, això no sigui així, no farem res. Però en el cas que l'ítem també l'hagi valorat $Usuari_2$, llavors calcularem la diferència que hi ha entre la puntuació de la valoració per a aquest ítem de UAnalitzat, respecte la puntuació de la valoració de $Usuari_2$. Això ho podem fer ja que un usuari només pot fer una valoració sobre un determinat item.

Un cop haguem calculat aquesta diferència per a un item, calcularem el quadrat d'aquesta i l'afegirem al double suma.

- Un cop ja haguem fet el pas anterior per tots els items de UAnalitzat, fariem **l'arrel quadrada del double suma**, calculant així la distància euclídea entre els dos usuaris.

  En la majoria de casos, el resultat de la distància entre dos usuaris serà aquest, però es poden produir dos casos en que això no sigui així:

  - Si aquesta suma supera el valor **20,** llavors la distància entre aquests dos usuaris la considerarem màxima, i per tant farem que la distància sigui de **20**.

  - En el cas que no s'hagi trobat cap ítem comú entre els dos usuaris, retornarem també la distància màxima, **20**, ja que això ens impossibilitat fer una comparació entre aquests dos usuaris.

Em triat que la distància màxima entre dos usuaris sigui **20** experimentalment, ja que si triem un valor molt alt, llavors això dificultarà la creació de clusters correctament a l'algorisme k-Means, ja que hi haura una diferència molt gran entre aquells usuaris que no s'han pogut comparar (ja que no coincideixen en termes d'items), i aquells que sí.

3.1.2 K-MEANS

L'algorisme k-Means és un mètode d'agrupament que té com a objectiu la partició d'un conjunt de n elements en k clusters en el qual cada observació pertany al grup més proper a la mitjana del clúster.
En aquest cas volem fer clústers d'usuaris (u_1, u_2, ..., u_n). Volem fer k particions de tal forma que cada usuari estigui amb un grup d'usuaris semblants. Per tal de determinar la proximitat entre dos usuaris diferents posseïm la distància entre cada parell d'usuaris. La distància és la distància **euclídea** i es calcula a partir de les valoracions de cada usuari a un ítem.

Primer de tot, assignarem un usuari a cada cluster, cada cluster contindrà un usuari diferent, i aquest usuari representarà el centroide. Per defecte hem posat un total de 3 clusters, però aquest número pot ser canviat desde la consola de comandes. Hem escollit una k de 3 ja que és un valor que bastant neutre que sol donar resultats en la mitjana, aquest valor l'hem trobat a partir de l'experimentació.

Tot seguit l'algorisme fa el següent procediment:

- S'itera per tots els usuaris que es disposen i es busca la distància de l'usuari a cada centroide.
  Al no disposar de punts en el espai, ni 1 ni 2 ni n dimensions, hem recorregut a només fer servir les distàncies per calcular distàncies a centroides. Aprofitem que un centroide és la mitjana de tots els punts del cluster per calcular la distància. D'aquesta forma tenim que la distància d'un usuari a un centroide és la mitjana de totes les distàncies entre l'usuari i tots els altres usuaris del cluster.
- Un cop hem calculat totes les distàncies a tots els clusters, ens quedem amb aquell cluster a menor distància ja que ens interessa ajuntar usuaris semblants.
- Col·loquem l'usuari al cluster a mínima distància, si no és que ja estava dins del cluster a menor distancia.
- Repetim aquest procés unes 20 vegades per tal d'assegurar-se que els clusters es formen d'una equitativa i correcte sense haver d'iterar moltes vegades.

3.1.3 SLOPE ONE

Slope One és un algorisme utilitzat per fer collaborative filtering i consisteix en predir la valoració que donaria l'usuari actiu a un item donat, a partir de les valoracions fetes per altres usuaris. Els altres usuaris solen ser usuaris pròxims, és a dir, amb gustos similars i es treuen del k-means.

La tècnica de Slope One es basa en que, depenent de com els usuaris propers, a l'usuari actiu, valoren uns certs items, l'usuari actiu valorarà de forma similar aquest items, ja que acostumen a tenir les mateixes preferències.

El càlcul del Slope One tracta d'agafar tots els items valorats per els usuaris del mateix cluster i mirar les desviacions entre el producte del que es vol predïr la valoració i un altre item que l'usuari veí ha valorat. D'aquestes desviacions entre parells d'ítems d'un mateix usuari, se'n treu una mitjana la qual és sumada a la valoració que ha fet l'usuari actiu sobre aquell item.

Així doncs amb el Slope One aconseguim saber com valoraria un cert ítem un cert usuari per posteriorment recomanar-li o no.

3.2 CONTENT-BASED FILTERING

Content-based filtering és l'estratègia que es basa en recomanar a l'usuari una sèrie d'ítems en funció a la seva activitat, és a dir els ítems amb els que ha interactuat i que li han agradat més o menys.

A diferència de Collaborative Filtering (apartat 4.1) aquest algorisme no necessita la informació d'altres usuaris, tan sols li calen les seves valoracions i el set d'ítems que s'utilitzaran.

Per aconseguir aquestes recomanacions s'utilitza l'algorisme k-nn (k-nearest neighbours, més informació sobre aquest algorisme a l'apartat 4.2.2), el qual retorna els k items més semblants al ítem sobre el que volem una recomanació.

3.2.1 DISTANCIES ENTRE ITEMS

Per tal de calcular la similitud entre els items existents (aquells que haguem carregat d'un fitxer de dades) ho fem amb la classe DistItem.

A diferència de la classe DistUsuari, en que es mesurava la distància d'un usuari a un conjunt d'usuaris, aqui no tenim cap ítem en concret, sinó que donat un conjunt d'items calculem la distància entre tots aquests, a tots aquests, i de forma independent, com a DistUsuari, és a dir, donat un item del conjunt calcularíem la distància entre aquest i un altre ítem del conjunt, fent això fins a trobar totes les combinacions possibles d'items.

Per tant, la classe estarà associada amb un conjunt d'ítems, que lògicament seran tots aquells dels quals disposem (que haguem creat a partir d'un fitxer de dades), ja que quan creem una recomanació, ho voldrem fer tenint en compte tots els ítems.

De forma igual que amb la classe DistUsuari, a la classe DistItem la distància entre un ítem i un altre s'expressa amb un número, un Double.

Ara bé, com calculem la distància en aquesta classe? Ho farem també calculant la **distancia euclídea**, tot i que entre dos items, segons els atributs de cada TipusAtribut que aquests dos tenen. Però aquí és una mica més complicat que respecte DistUsuari, ja que aquí segons el valor "tipusDada" de cada TipusAtribut, ho haurem de fer d'una manera o altre.

Comencem, doncs, pel principi. Imaginem que volem calcular la distància entre dos items, un l'anomenarem item1, i l'altre item2.

- Primerament, tindrem un double anomenat **suma,** on hi guardarem la diferència entre els dos items. Primerament, aquest double l'inicialitzarem a 0.

- A més, abans de calcular res, per a cada ítem ordenarem els seus atributs (els atributs (de la classe Atribut) als que està associat) pel TipusAtribut de cada un d'aquests, de forma que, donat un ítem, poguem saber per a quins TipusAtribut tenim informació.

- LLavors, per a cada TipusAtribut que tinguem informació de item1, mirarem, per una banda, si aquest TipusAtribut és calculable (tipusDada == true, de TipusAtribut), i de l'altre, si també tenim informació d'aquest TipusAtribut (és a dir atributs associats a aquest TipusAtribut) per a ítem2. En cas que alguna d'aquestes dues coses no es compleixi, no faríem res per a aquest TipusAtribut, ja que no podem comparar els dos items amb aquest TipusAtribut.

  Però si les dues coses es compleixin, si que podrem. LLavors, el que farem és calcular una diferència per al TipusAtribut actual entre els atributs d'aquest TipusAtribut de item1, respecte item2. Aquesta diferència, anirà de 0, a 10 (essent 0 la similitud completa, i 10 el contrari). Però segons el tipusDada del TipusAtribut actual, calcularem aquesta diferència d'una forma u altre:

  - Cas tipusDada == "**Bool**": Aquest cas és fàcil, ja que un booleà només pot ser cert o fals. A més, un ítem només pot tenir un atribut d'aquest TipusAtribut. Per tant, si el valor del atribut de item1 és diferent que el de item2, llavors la diferència serà 10. En canvi, si és igual, serà 0.

  - Cas tipusDada == "**String**": Aquest cas és el més complicat, ja que aquí un ítem pot estar associat amb diversos atributs que estiguin associats a aquest TipusAtribut (per exemple: un item pot estar associat amb els atributs: Drama, i Tragèdia; pertaneixents tots dos al TipusAtribut amb nom: Gènere). LLavors el que farem serà:

    - Primer, calcular quants amb quants atributs d'aquest TipusAtribut està associat l'ítem (item1 o item2) que té menys associacions d'aquest. A aquesta quantitat l'anomenarem divisor.

    - Llavors calcularem el nombre d'atributs coincidents per aquest tipusAtribut entre els dos items.

    - Finalment farem: (1.0 - (atributscoincidents/divisor))*10. D'aquesta manera, si no coincideix cap atribut d'aquest tipusAtribut pels dos items, la diferència seria 10; mentre que si coincideixen tots, la diferència seria 0.

- Cas tipusDada == "**Int**": En aquest cas, com que per una part, de la mateixa forma que amb tipusDada == "Bool", un ítem només pot tenir un atribut d'aquest TipusAtribut, i perquè els valors dels atributs són números, farem el següent:

    - Calcularem la diferència que hi ha entre l'atribut amb el valor més alt d'entre tots els que hi ha per a aquest TipusAtribut, i el més petit. A aquesta diferència l'anomenarem diferènciaMàxima.

    - Calcularem la diferència que hi ha entre el valor de l'atribut associat a aquest TipusAtribut de item1, respecte el de item2. A aquesta diferència l'anomenarem diferènciaAtributs

    - LLavors la diferència verdadera que obtindrem serà = (diferènciaAtributs/diferènciaMàxima) * 10.

- Cas tipusDada == "**Double**": El mateix que en el cas tipusDada == "Int", però amb double.

Un cop tenim aquesta diferència, calcularem el quadrat d'aquesta i l'afegirem al double suma.

- Un cop ja haguem fet el pas anterior per tots els TipusAtribut dels que tenim informació de item1, fariem l'arrel quadrada del double suma, calculant així la distància euclídea entre els dos items.

  Això serà així en la majoria de casos, excepte un:

    - En cas que per a cap dels TipusAtribut que tenim informació de item1 en tenim per a item2, llavors no podrem comparar aquests dos items, de forma que farem veure que la diferència sigui infinita. És a dir, diferència == Double.MAX_VALUE. Però això és molt poc probable que passi, casi impossible.

3.2.2 K-NEAREST NEIGHBOURS

L'algorisme k-nearest neighbours fa ús dels ítems que ha valorat un usuari. Donat un ítem a classificar busca els seus respectius ítems més semblants per, posteriorment, seleccionar l'ítem més abundant entre tots els grups de cada ítem valorat per l'usuari. Per aquest motiu, en el nostre projecte, si un usuari no té cap valoració l'algorisme k-nn farà saltar un error.

Primerament k-nn s'assegura que hi hagi suficients ítems per poder crear recomanacions i suficients ítems valorats per l'usuari abans de passar a descartar-los. En el nostre projecte hem suposat que les valoracions que "agraden" a un usuari són aquelles amb valors de 3 o més de 3, per tant descartem aquelles valoracions amb menys de 3. Tanmateix, com que ens podem trobar amb que l'usuari no ha puntuat suficients valoracions que li "agradin" en els casos en que no hi ha cap valoració de més de 3 agafarem aquelles que tenen puntuacions més grans o iguals que 2, i de la mateixa forma que existeixen aquests casos, també repetim el procediment si no hi ha puntuacions més grans o iguals que 2 amb les puntuacions més grans o iguals que 1.

Seguidament l'algorisme rep la distància calculada per DistItems (Distàncies entre Ítems, més informació a l'apartat 4.2.1) entre l'ítem valorat per l'usuari i tots els ítems existents, que no ha valorat l'usuari. La selecció d'ítems semblants es fa a partir de les distàncies, si la distancia calculada és menor a la distància màxima (que en el nostre cas hem assignat 20) llavors entrarà a la llista de "possibles ítems recomanats". Cada "possible ítem recomanat" tindrà un comptador, per veure quantes vegades es repeteix, però aquest comptador no s'incrementa sempre igual. Com que cada ítem té una recomanació el que es farà és que el comptador es veurà incrementat per la puntuació que tenia l'ítem amb el que s'assembla, per exemple, si l'usuari ha valorat ens ítems 1 i 2, amb puntuació 3 i 5 respectivament, i l'ítem 1 s'assembla a l'ítem 4 i l'ítem 2 s'assembla a l'ítem 5 la recomanació final serà l'ítem 5, ja que el seu comptador valdrà 5 mentre que l'ítem 4 tindrà puntuació 3. A més per trencar empats afegim una part fraccional a aquest comptador mitjançant la fórmula (distanciaMaxima-distància)/disanciatMaxima * puntuació, de forma que aquells ítems que s'assemblen més a l'ítem amb bona puntuació sortiran beneficiats.

Finalment, un cop tenim la nostra llista amb puntuacions, el que fem és agafar els k ítems amb major valor, de manera que serien els que surten més vegades repetits, i a més en els ítems que més han agradat a l'usuari.

4. JUSTIFICACIÓ DE LES ESTRUCTURES DE DADES

Un cop ja tenim explicats els algorismes, s'ha de justificar quines estructures de dades hem utilitzat per a l'implementació d'aquests, és a dir, perquè la implementació que hem utilitzat és beneficiosa en termes de cost de cara a l'optimització temporal del càlcul d'aquests:

4.1 COLLABORATIVE FILTERING

Per explicar les estructures de dades d'aquest algorisme, abans necessitem entendre les de la classe "DistUsuari".
A DistUsuari tal com s'ha explicat a l'apartat anterior, calculem les distàncies que van des d'un determinat usuari (UAnalitzat), cap a tots els altres. La constructora de DistUsuari té com a argumet un Set<Uusari> que conté un set amb tots els usuaris. Posteriorment es calculen totes les distàncies entre el UAnalitzat i cada usuari del Set i s'emmagatzema dins un HashMap <Usuari, Double>.
L'elecció de fer servir un HashMap és que podem emmagatzemar parells de key, valor directament i que és la implementació de Map que més ens interessa. A l'hora de calcular les distàncies per a un usuari, necessitem mirar tots els ítems del segon usuari, per tant hem d'iterar sobre un Set de ítems de l'usuari per a cada usuari. Per tant la complexitat és O(n) on n és el número d'usuaris ja que hem de fer n iteracions i en cada una d'elles k iteracions on k és el número de ítems valorats per l'usuari i per tant constant O(1), i una operació d'incersió en un HashMap de també cost O(1).

D'aquesta forma la classe de Collaborative Filtering disposa d'un HashMap<Integer,HashMap<Integer, Double>> que emmagatzema les distàncies entre usuaris i un HashMap<Integer,Usuari> que conté tots els usuari i la seva corresponent identificació. El primer HashMap de HashMaps ens interessa especialment per poder accedir a les distàncies de forma fàcil i ràpida, per el HashMap d'usuaris passa el mateix, en és molt útil poder accedir ràpidament a un usuari per posteriorment fe-li consultes.
El cost de calcular totes les distàncies entre cada parell d'usuaris és de $O(n^2)$ ja que hem de calcular les distàncies de n usuaris cap a n usuaris.

D'altra banda, per a poder calcular els clústers on disposem cada usuari necessitem executar el k-Means. Aquest se li passa com a paràmetre una copia de les distàncies entre usuaris per tal de poder-les consultar quan es vulgui. A l'hora de calcular els clústers, primer de tot s'assignen de forma random k centroides que són usuaris per tant fem k iteracions per assignar els centoides. Aquests centroides s'emmagatzemen en un ArrayList<HashSet<Integer>> on el integer és l'identificador d'un usuari. Cada posició de l'ArrayList representa un cluster per tant tindrem una ArrayList de k posicions i cada posició conté un HashSet amb tots els usuaris que formen part del clúster. La utilització d'una

ArrayList és deguda a que és l'estructura de dades més eficient que ens permet guardar tots els clústers. Altrament, un HashMap ens permet guardar tots els identificadors, consultar-los i veure si estan en temps constant O(1). Per tant tenim O(k) per assignar els k centroides a un cluster.
Per tal de col·locar tots i cada un dels usuaris en un cluster i que aquest sigui el adequat executem 20 vegades iteracions de n vegades, on n són el número de usuaris el següent procés:

- buscar el centroide a mínima distància. Per fe-ho necessita recórrer la ArrayList de HashMaps clusters, és a dir, k vegades, n-k usuaris com a màxim. $O(k \cdot n)$.
- assignar el usuari al cluster a menys distància. O(1).

Per tant, el càlcul dels clústers és de $O(k \cdot n^2)$.

Finalment, creem un HashSet de Ítems a valorar per a poder guardar tots on guardem tots aquells items que han valorat els usuaris del mateix cluster que l'UActiu però no han estat valorats per l'UActiu. L'utilització d'un HashSet no té gaire relevancia aquí ja que només emmagatzema Items. Tanmateix, creem un HashMap de tots aquest items i la seva corresponent predicció. I d'aquesta forma només queda parlar del Slope One.

L'algorisme de Slope One rep com a argument un HashMap<Integer, Usuari> que són els usuaris veïns de l'UActiu. Aquesta estructura ens permet accedir fàcilment i ràpidament a la informació de cada Usuari veí.
Dins de la funció on es calcula la predicció de valoració per un item j, iterem sobre totes le valoracions de l'usuari actiu, O(k). A més, iterem sobre cada usuari veí, i en ell sobre les seves valoracions per tal de poder saber les desviacions de puntuacions del items. Per tant tenim $O(k \cdot n)$. Tot seguit récorrem un HashMap<Item,ArrayList<Double>> diferències que conté totes les diferències de valoracions de cada item per a cada usuari que l'ha valorat que no sigui el UActiu. El cost de recórrer aquest HashMap és de $O(k \cdot u) = O(k)$ on k són els items valorats per l'usuari actiu. En resum, el cost de l'algorisme SlopeOne és de $O(k \cdot n)$.

Conclusió: Com hem vist, les estructures de dades que s'utilitzen principalment en el collaborative filtering són HashMap i HashSet ja que són les implementacions més edients de Map i Set. A més, no ens interessa tenir ordenats els elements, sinó que volem accedir, modificar, afegir, eliminar de forma ràpida, i aquestes estructures de dades ens ho permeten sense comprometre el cost, com hem vist. També s'utilitzen ArrayList quan es volen guardar elements d'una forma més ordenada i la implementació ens ho demana.

4.2 CONTENT-BASED FILTERING

En aquest algorisme, com en l'anterior, primer necessitem també entendre primer l'estructura de dades amb que funciona la classe que li passa les distàncies a aquest algorisme.

En aquest cas, és la classe DistItem. El primer que cal destacar, és que aquí, a diferència del que passa amb DistUsuari (en que només es calculen les distàncies d'un usuari cap a altres), directament li introduïm un set d'items a la classe, i aquesta automàticament ja crea totes les distàncies de tots els ítems, cap a tots els ítems, creant així totes les distàncies per a totes les combinacions possibles d'ítems introduïts.

Hem decidit fer això així perquè com que al calcular les distàncies entre items, comparem aquests segons els seus atributs, i un cop un ítem ja s'ha creat els seus atributs ja no es poden modificar, de forma que aquí és molt menys probable que s'hagin de recalcular distàncies entre ítems (si u comparem amb usuaris i DistUsuari).

De fet, no haurem de recalcular les distàncies entre dos items mai, ja que els atributs entre aquests dos mai canviaràn, de forma que si ho fem d'aquesta manera, un cop hem calculat les distàncies per primera vegada, ja no tenim que fer res més. A més, si un usuari actiu per exemple volgués crear un ítem, tenim la possibilitat d'afegir aquest a la classe DistItem de forma que només s'hagin de calcular les distàncies dels ítems que ja tenia aquesta classe cap a l'ítem nou. I si per contra, un usuari vol eliminar un ítem, només haurem d'eliminar les distàncies que van cap (o des de) aquest ítem.

Per tant, per tal d'implementar aquest funcionament de la classe DistItem, i a més, intentant-ho fer de la forma més eficient possible, hem implementat aquesta classe amb dos estructures:

- HashMap<Item, HashMap <TipusAtribut, HashSet<Atribut> > > tipusAtributsItems;
- HashMap<Item, HashMap<Item, Double> > distàncies;

Primerament podem observar que no utilitzem cap llista, sinó només maps, això és perquè no ens interessa guardar cap ítem (o tipus atribut de item) en cap ordre en concret.

La primera estructura és necessària per calcular la segona (és a dir, per calcular les distàncies entre items). A la primera estructura podem observar que, per a cada item, classifiquem els atributs d'aquests segons els seus TipusAtribut. Això, ho fem d'aquesta manera ja que a l'hora de calcular la distància entre dos items, ho farem comparant els diversos atributs d'aquests dos segons el TipusAtribut que siguin aquests. D'aquesta forma, a l'hora de comparar dos items, ja tenim els atributs d'aquests dos ordenats i podem fer el càlcul ràpidament.

La segona estructura, és similar a: HashMap<Integer,HashMap<Integer, Double>>; la qual es troba a l'algorisme Collaborative Filtering, tot i que aquí no es tracta d'usuaris, sinó d'items, i a més aquesta estructura ja es creada directament dins de la classe DistItem.

Per tant, d'aquesta manera, per calcular totes les distàncies que van d'un item cap a tots els altres, trigarem O(n), i com que tenim n items, totes les combinacions de distàncies possibles es poden realitzar en temps $O(n^2)$. Això és possible ja que el calcul d'una distància el podem considerar constant (O(1)), pel simple fet que tenim molts menys TipusAtributs que Items.

Un cop explicat això, ja podem passar a l'algorisme. L'estructura de la qual disposa la classe CBFiltering és la següent:

- HashMap<Item, HashMap<Item, Double>> totesDistancies;

Si ens fixem, aquesta estructura ja la tenim, ja que és la que ha calculat la classe DistItem, de forma que la podem crear sencera sense cost extra.

Llavors, pel que fa al cost temporal de la classe CBFiltering, ens hem de fixar en com es fa la creació del que retorna aquesta, que és:

- ArrayList<Item>

Aquí sí que hem optat per una llista, perquè ens interessa retornar la recomanació d'ítems amb un ordre en que els primers ítems de la llista siguin els millors, és a dir, els més recomanats. No obstant la ordenació d'aquests no comporta cap penalització en el cost temporal.

De fet, el cost temporal de la classe CBFiltering ve marcat per la creació de

- HashMap<Item, Double> posicions

és a dir, per la computació dels elements més propers d'un item. No obstant, aquest calcul no serà mai superior a $O(n^2)$. Per tant, finalment el cost temporal de l'algorisme Content-Based Filtering serà $O(n^2)$.

Conclusió: Finalment, per a aquest algorisme podem obtenir unes conclusions bastant similars a les que hem acabat obtenint per a l'algorisme Collaborative Filtering. Hem vist que les estructures de dades més adients son el HashMap i HashSet, ja que no ens interessa tenir ordenats els elements, sinó que volem accedir, modificar, afegir, eliminar de forma ràpida… I finalment, aquí ja si que obtar per una llista, on retornar de forma directa una llista de recomanacions.

5. RELACIÓ DE LES CLASSES IMPLEMENTADES PER CADA MEMBRE DE L'EQUIP

Tot i que principalment la creació de les diferents classes va ser repartida pels diferents membres de l'equip, també s'ha de dir que finalment cada membre de l'equip ha acabat aportant també el seu gra de sorra a altres classes que principalment no li pertocaven, amb l'objectiu de reforçar i progressar en la creació del projecte.

Miquel:

- kMeans.
- SlopeOne.
- COLFiltering (junt amb DCG).

Hector:

- TipusAtribut.
- Atribut.
- Les subclasses d'atribut (AtrInt, AtrBool, AtrString, AtrDouble).
- DataReader

Agnès:

- CBFiltering.
- Ítem.
- Valoracio.

David:

- Usuari (i la seva subclasse UActiu).
- DistItem.
- DistUsuari.