Sistema de recomanació

Primer lliurament (1.0)

Projectes de programació (PROP)

Grau en Enginyeria Informàtica - FIB - UPC

Curs 2021-2022, Quadrimestre de Tardor

Equip 4.2

Ferran De La Varga Antoja (ferran.de.la.varga)

Alexandru Dumitru Maroz (alexandru.dumitru)

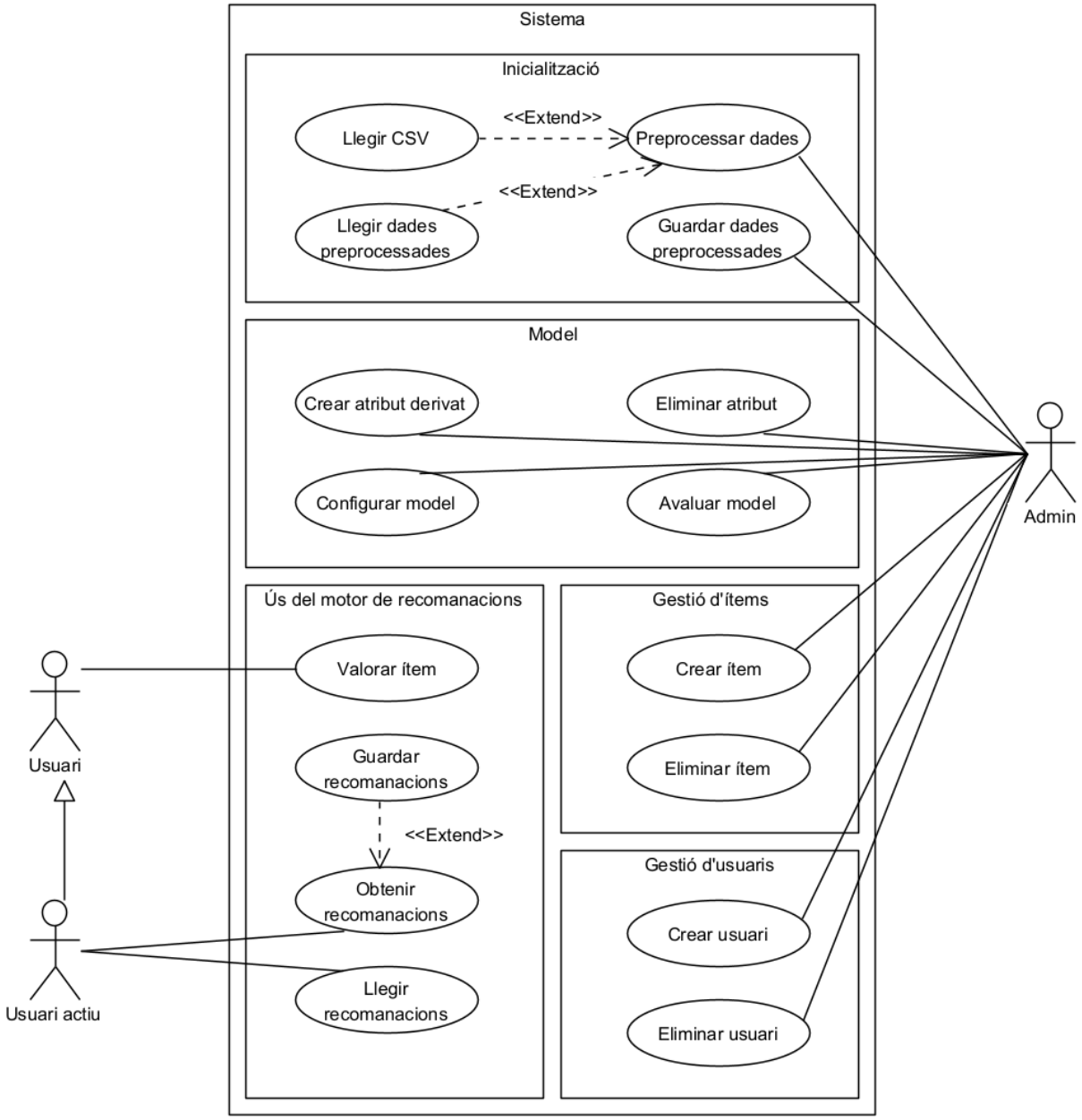
Pablo José Galván Calderón (pablo.jose.galvan)

Pol Rivero Sallent (pol.rivero)

Índex

1. Casos d'ús

1.1 Diagrama de casos d'ús



1.2 Descripcions dels casos d'ús

1.2.1 Preprocessar dades

Actors primaris: Admin

Activació: L'Admin inicia el software.

Escenari principal d'èxit:

1. L'Admin selecciona si vol obtenir les dades d'un fitxer CSV o d'un fitxer de dades ja preprocessades.
2. L'Admin selecciona un fitxer.
3. El Sistema llegeix el fitxer.
4. El Sistema processa les dades i les guarda per utilitzar-les a l'execució.
5. El Sistema mostra les dades.

Extensions:

format-invàlid: El contingut del fitxer seleccionat no és vàlid (4).

1. El Sistema informa l'Admin que el contingut del fitxer seleccionat té un format invàlid.
2. Tornar al pas 1 de l'escenari principal.

fitxer-no-existent: El Sistema no pot trobar el fitxer (3).

3. El Sistema informa l'Admin que el fitxer no existeix o no es pot llegir.
4. Tornar al pas 1 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Preprocessar les dades és el procediment que farem servir per extreure les dades de fitxer CSV i modificar-les de tal manera que; ordenem totes les columnes, seleccionem quines es faran servir per a cada càlcul, eliminem dates errònies/outliers i els convertirem a un format més fàcilment llegible per al software.
- El Sistema accepta 3 tipus d'atributs: numèrics, booleans i categòrics (llistes de tags).

1.2.2 Llegir CSV

Actors primaris: Admin

Activació: Des de “Preprocessar Dades”, l’Admin selecciona l’opció de llegir un fitxer CSV.

Escenari principal d’èxit:

1. L’Admin navega fins al directori on vol trobar el fitxer CSV d'ítems.
2. L’Admin selecciona el fitxer CSV desitjat.
3. El Sistema llegeix el fitxer CSV seleccionat i carrega en memòria els atributs i ítems.
4. L’Admin navega fins al directori on vol trobar el fitxer CSV d’usuaris.
5. L’Admin selecciona el fitxer CSV desitjat.
6. El Sistema llegeix el fitxer CSV seleccionat i carrega en memòria els usuaris i les seves valoracions.
7. El Sistema configura el model amb els paràmetres (funció de distància, algorisme de recomanació, etc.) predeterminats.

Extensions:
avortar-operació: L’Admin cancel·la l’operació (*).

1. El Sistema torna a l’estat inicial.
2. Acaba el cas d’ús.

fitxer-items-no-existent: El Sistema no pot trobar el fitxer (3).

1. El Sistema informa l’Admin que el fitxer no existeix o no es pot llegir.
2. Tornar al pas 1 de l’escenari principal.

fitxer-usuaris-no-existent: El Sistema no pot trobar el fitxer (6).

1. El Sistema informa l’Admin que el fitxer no existeix o no es pot llegir.
2. Tornar al pas 4 de l’escenari principal.

fitxer-invàlid: Els continguts del fitxer CSV violen alguna restricció d’integritat (3, 6).

1. El Sistema informa l’Admin que el fitxer no és correcte.
2. El Sistema torna a l’estat inicial.
3. Acaba el cas d’ús.

Notes:

- Implementa la funcionalitat obligatòria “Permetre la definició de tipus d’ítems a partir dels seus atributs”.
- Fa <<Extend>> de “Preprocessar dades”.

1.2.3 Llegir dades preprocessades

Actors primaris: Admin

Activació: Des de “Preprocessar Dades”, l’Admin selecciona l’opció de llegir un fitxer de dades preprocessades.

Escenari principal d’èxit:

1. L’Admin navega fins al directori on vol trobar el fitxer de dades preprocessades.
2. L’Admin selecciona el fitxer de dades preprocessades desitjat.
3. El Sistema llegeix el fitxer de dades preprocessades seleccionat i carrega les dades en memòria.

Extensions:
avortar-operació: L’Admin cancel·la l’operació (*).

1. El Sistema torna a l’estat inicial.
2. Acaba el cas d’ús.

fitxer-no-existent: El Sistema no pot trobar el fitxer (3).

3. El Sistema informa l’Admin que el fitxer no existeix o no es pot llegir.
4. Tornar al pas 1 de l’escenari principal.

Notes:

- Implementa la funcionalitat obligatòria “S’ha de donar l'opció de què l'usuari pugui guardar dades preprocessades per fer més eficient el procés en treballar de nou amb el mateix conjunt de dades”.
- Fa <<Extend>> de “Preprocessar dades”.

1.2.4 Guardar dades preprocessades

Actors primaris: Admin

Activació: L'Admin prem el botó per guardar les dades actuals.

Escenari principal d'èxit:

1. L'Admin navega fins al directori on vol guardar les dades preprocessades.
2. L'Admin introdueix un nom pel fitxer nou.
3. El Sistema enregistra el nom.
4. L'Admin selecciona l'opció de guardar el fitxer.
5. El Sistema transforma les dades actuals en un format que li permeti llegir-les directament més endavant si es desitja.
6. El Sistema guarda un fitxer de dades preprocessades amb les dades transformades al directori especificat.

Extensions:

nom-repetit: Ja existeix un fitxer de dades preprocessades amb el mateix nom (3).

1. El Sistema sol·licita a l'Admin confirmació per sobreescriure l'arxiu existent.
2. Si l'Admin ho confirma, el Sistema sobreescriu l'arxiu. Si no, torna al pas 1 de l'escenari principal.

no-permès: Guardar el fitxer no és permès pel SO (5).

1. El Sistema informa l'Admin que no és possible guardar el fitxer.
2. Tornar al pas 1 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*)

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "S'ha de donar l'opció de què l'usuari pugui guardar dades preprocessades per fer més eficient el procés en treballar de nou amb el mateix conjunt de dades".

1.2.5 Crear atribut derivat

Actors primaris: Admin

Activació: L'Admin navega a la llista d'atributs i prem el botó per crear-ne un de nou.

Escenari principal d'èxit:

1. L'Admin selecciona un atribut existent.
2. El Sistema enregistra l'atribut existent a partir del qual derivar el nou atribut.
3. El Sistema mostra la llista d'operadors possibles, segons el tipus de l'atribut seleccionat.
4. L'Admin introdueix un operador, un argument per l'operació i el valor numèric que cal assignar al nou atribut si l'operació retorna true.
5. El Sistema enregistra les dades de l'operació.
   Es repeteixen els passos 3 a 5 fins que s'han introduït totes les operacions.
6. L'Admin introdueix el valor numèric per defecte que cal assignar al nou atribut si cap de les operacions retorna true.
7. El Sistema enregistra el valor per defecte.
8. El Sistema crea el nou atribut i computa tots els seus valors.

Extensions:

atribut-no-existent: L'atribut seleccionat no existeix (2).

1. El Sistema informa l'Admin que l'atribut seleccionat no existeix.
2. Tornar al pas 1 de l'escenari principal.

atribut-invàlid: L'atribut seleccionat no és vàlid per la creació d'un atribut derivat (2).

1. El Sistema informa l'Admin que l'atribut seleccionat no és vàlid per la creació d'un atribut derivat.
2. Tornar al pas 1 de l'escenari principal.

operador-no-existent: L'operador seleccionat no existeix (5).

1. El Sistema informa l'Admin que l'operador no existeix.
2. Tornar al pas 4 de l'escenari principal.

argument-invàlid: L'argument seleccionat és del tipus incorrecte (5).

1. El Sistema informa l'Admin que l'argument no és vàlid.
2. Tornar al pas 4 de l'escenari principal.

resultat-invàlid: El valor numèric a assignar al nou atribut no és numèric (5).

1. El Sistema informa l'Admin que el resultat no és vàlid.
2. Tornar al pas 4 de l'escenari principal.

resultat-per-defecte-invàlid: El valor numèric a assignar al nou atribut no és numèric (7).

1. El Sistema informa l'Admin que el resultat no és vàlid.
2. Tornar al pas 6 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat opcional "Afegir atributs nous".
- L'atribut original no pot ser booleà.
- El nou atribut sempre és numèric.
- Operadors per a atributs numèrics: $< > \leq \geq =$
- Operadors per a atributs categòrics (tags): $\in$
- En una operació, el valor a assignar al nou atribut també pot ser "same" per utilitzar l'original, però només si l'atribut original és numèric.
- En una operació, el valor a assignar al nou atribut també pot ser "invalid" per ometre aquell atribut en l'algorisme de recomanacions.

1.2.6 Eliminar atribut

Actors primaris: Admin

Activació: L'Admin navega a la llista d'atributs i prem el botó per eliminar un de concret.

Escenari principal d'èxit:

1. El Sistema enregistra l'atribut seleccionat.
2. El Sistema demana confirmació.
3. El Sistema elimina l'atribut i tots els seus valors.

Extensions:
atribut-no-existent: L'atribut seleccionat no existeix (2).

1. El Sistema informa l'Admin que l'atribut seleccionat no existeix.
2. Tornar al pas 1 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat opcional "Eliminació d'atributs".

1.2.7 Configurar model

Actors primaris: Admin

Activació: L'Admin prem el botó per configurar els paràmetres del model.

Escenari principal d'èxit:

1. L'Admin selecciona un atribut existent.
2. El Sistema enregistra l'atribut seleccionat.
3. L'Admin introdueix el nou pes de l'atribut per a l'algorisme de recomanacions.
4. El Sistema enregistra el pes de l'atribut.
   Es repeteixen els passos 1 a 4 fins que s'han introduït tots els pesos.
5. El Sistema mostra les funcions de distància disponibles.
6. L'Admin selecciona la funció de distància per a l'algorisme de recomanacions.
7. El Sistema enregistra la nova funció de distància.
8. El Sistema mostra els algorismes de recomanació disponibles.
9. L'Admin selecciona un algorisme.
10. El Sistema enregistra el nou algorisme a utilitzar.

Extensions:
atribut-no-existent: L'atribut seleccionat no existeix (2)

1. El Sistema informa l'Admin que l'atribut seleccionat no existeix.
2. Tornar al pas 1 de l'escenari principal.

pes-invàlid: El pes introduït no és de tipus numèric o és negatiu (4)

1. El Sistema informa l'Admin que el pes no és vàlid.
2. Tornar al pas 3 de l'escenari principal.

funció-no-existent: La funció de distància seleccionada no existeix (7)

1. El Sistema informa l'Admin que la funció seleccionada no existeix.
2. Tornar al pas 5 de l'escenari principal.

algorisme-no-existent: L'algorisme de recomanacions seleccionat no existeix (10)

1. El Sistema informa l'Admin que l'algorisme seleccionat no existeix.
2. Tornar al pas 8 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*)

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "Definir funcions de distància o similitud depenent dels atributs utilitzats".

1.2.8 Avaluar model

Actors primaris: Admin

Activació: L'Admin prem el botó per avaluar els paràmetres del model.

Escenari principal d'èxit:

1. L'Admin navega fins al directori on vol trobar el fitxer CSV *Known*.
2. L'Admin selecciona el fitxer CSV desitjat.
3. L'Admin navega fins al directori on vol trobar el fitxer CSV *Unknown*.
4. L'Admin selecciona el fitxer CSV desitjat.
5. El Sistema llegeix els dos fitxers CSV.
6. El Sistema prepara un conjunt de queries a partir dels continguts dels CSV.
7. El Sistema executa cada query i calcula el seu DCG.
8. El Sistema mostra a l'Admin la mitjana del DCG de totes les queries.

Extensions:

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

fitxer-known-no-existent: El Sistema no pot trobar el fitxer (5).

1. El Sistema informa l'Admin que el fitxer no existeix o no es pot llegir.
2. Tornar al pas 1 de l'escenari principal.

fitxer-unknown-no-existent: El Sistema no pot trobar el fitxer (5).

1. El Sistema informa l'Admin que el fitxer no existeix o no es pot llegir.
2. Tornar al pas 3 de l'escenari principal.

fitxer-invàlid: Els continguts del fitxer CSV violen alguna restricció d'integritat (5).

1. El Sistema informa l'Admin que el fitxer no és correcte.
2. El Sistema torna a l'estat inicial.
3. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "Avaluació d'un conjunt de recomanacions (és a dir, calcular la seva qualitat)".

1.2.9 Crear ítem

Actors primaris: Admin

Activació: L'Admin navega a la llista d'ítems i prem el botó per crear-ne un de nou.

Escenari principal d'èxit:

1. El Sistema genera un identificador per l'ítem.
2. El Sistema mostra un dels atributs del model.
3. L'Admin introdueix el valor de l'atribut.
4. El Sistema enregistra el valor de l'atribut.
   Es repeteixen els passos 2 a 4 fins que s'han introduït tots els atributs definits.
5. El Sistema crea un nou ítem amb totes les dades corresponents al nou ítem.
6. El Sistema mostra a l'usuari l'identificador del nou ítem.

Extensions:

valor-invàlid: El valor introduït no és tipus correcte (4).

1. El Sistema informa l'Admin que el valor introduït no és del tipus esperat.
2. Tornar al pas 3 de l'escenari principal.

Notes:

- Implementa la funcionalitat obligatòria "Permetre la gestió d'ítems".
- Si el camp d'un atribut es deixa en blanc, l'atribut serà "invalid" i no s'assignarà cap valor.

1.2.10 Eliminar item

Actors primaris: Admin

Activació: L'Admin navega a la llista d'ítems i prem el botó per eliminar un de concret.

Escenari principal d'èxit:

1. El Sistema enregistra l'ítem seleccionat.
2. El Sistema demana confirmació.
3. El Sistema elimina l'ítem, tots els seus valors associats als atributs del model i totes les valoracions dels usuaris sobre aquest ítem.

Extensions:
item-no-existent: L'item seleccionat no existeix (2).

1. El Sistema informa l'Admin que l'atribut seleccionat no existeix.
2. Tornar al pas 1 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "Permetre la gestió d'ítems".

1.2.11 Crear usuari

Actors primaris: Admin

Activació: L'Admin navega a la llista d'usuaris i prem el botó per crear-ne un de nou.

Escenari principal d'èxit:

1. L'Admin introdueix el nom de l'usuari (identificador).
2. El Sistema enregistra el nom.
3. El Sistema crea un usuari.

Extensions:
usuari-existent: Ja existeix un usuari amb l'identificador proposat.

1. El Sistema informa l'Admin que l'usuari ja existeix.
2. Es torna al pas 1 de l'escenari principal.

Notes:

- Implementa la funcionalitat obligatòria "Permetre la gestió d'usuaris".

1.2.12 Eliminar usuari

Actors primaris: Admin

Activació: L'Admin navega a la llista d'usuaris i prem el botó per eliminar un de concret.

Escenari principal d'èxit:

1. El Sistema enregistra l'usuari seleccionat.
2. El Sistema demana confirmació.
3. Si l'admin ha confirmat, s'elimina l'usuari corresponent i totes les valoracions dels diferents ítems que ha fet l'usuari eliminat.

Extensions:
usuari-no-existent: No existeix un usuari amb l'identificador proposat.

1. El Sistema informa l'Admin que l'usuari no existeix.
2. Es torna al pas 1 de l'escenari principal.

avortar-operació: L'Admin cancel·la l'operació (*).

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "Permetre la gestió d'usuaris".

1.2.13 Valorar ítem

Actors primaris: Usuari

Activació: Usuari navega a la llista d'ítems i prem el botó de valorar un ítem.

Escenari principal d'èxit:

1. El Sistema enregistra l'ítem seleccionat.
2. L'Usuari introdueix la valoració de l'ítem corresponent.
3. El sistema enregistra la valoració de l'ítem.

Extensions:

format-de-valoració-no-acceptat: El format de la valoració és incorrecte (5).

1. El Sistema informa l'Usuari que el format és incorrecte.
2. Torna al pas 4.

avortar-operació: L'Usuari cancel·la l'operació (*)

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

no-existeix-item: L'item seleccionat no existeix (3)

1. El sistema informa l'Usuari que l'item no existeix.
2. Torna al pas 2.

Notes:

- Implementa la funcionalitat obligatòria "Permetre la valoració d'ítems per part d'un usuari".

1.2.14 Obtenir recomanacions

Actors primaris: Usuari actiu

Activació: L'usuari inicia el sistema

Escenari principal d'èxit:

1. El Sistema executa l'algorisme seleccionat per l'Admin sobre el model.
2. El Sistema mostra les recomanacions obtingudes per a l'Usuari.

Extensions:

model-buit: No hi ha cap atribut definit (1)

1. El Sistema informa l'Usuari actiu que hi ha hagut un error.
2. Acaba el cas d'ús.

usuari-no-te-items-valorats: La llista d'ítems de l'usuari està buida

1. El Sistema informa l'Usuari actiu que hi ha hagut un error.
2. Acaba el cas d'ús.

Notes:

- Implementa la funcionalitat obligatòria "Recomanar un conjunt d'ítems a un usuari a partir d'un conjunt de dades".

1.2.15 Guardar recomanacions

Actors primaris: Usuari actiu

Activació: Des de “Obtenir recomanacions”, l’Usuari prem el botó per guardar les recomanacions.

Escenari principal d’èxit:

1. L’Usuari navega fins al directori on vol guardar les recomanacions.
2. L’Usuari escull un nom pel fitxer nou.
3. El Sistema enregistra el nom.
4. L’Usuari selecciona l’opció de guardar el fitxer.
5. El Sistema guarda un fitxer amb els resultats de l’algorisme de recomanació.

Extensions:

nom-repetit: Ja existeix un fitxer de dades preprocessades amb el mateix nom (3)

1. El Sistema sol·licita a l’Usuari confirmació per sobreescriure l’arxiu existent.
2. Si l’Admin ho confirma, el Sistema sobreescriu l’arxiu. Si no, torna al pas 1 de l’escenari principal.

no-permès: Guardar el fitxer no és permès pel SO (5)

1. El Sistema informa l’Usuari actiu que no és possible guardar el fitxer.
2. Tornar al pas 1 de l’escenari principal.

avortar-operació: L’Usuari actiu cancel·la l’operació (*)

1. El Sistema torna a l’estat inicial.
2. Acaba el cas d’ús.

Notes:

- Implementa la funcionalitat obligatòria “Guardar i recuperar el resultat de l'algorisme principal (recomanacions)”.
- Fa <<Extend>> de “Obtenir Recomanacions”.

1.2.16 Llegir recomanacions

Actors primaris: Usuari actiu

Activació: L'usuari prem el botó per llegir les valoracions.

Escenari principal d'èxit:

1. L'Admin navega fins al directori on vol trobar el fitxer de recomanacions.
2. L'Admin selecciona el fitxer de recomanacions desitjat.
3. El sistema llegeix les recomanacions guardades a l'arxiu.
4. Es mostren els resultats per pantalla.

Extensions:

avortar-operació: L'Usuari actiu cancel·la l'operació (*)

1. El Sistema torna a l'estat inicial.
2. Acaba el cas d'ús.

fitxer-no-existent: El Sistema no pot trobar el fitxer (3)

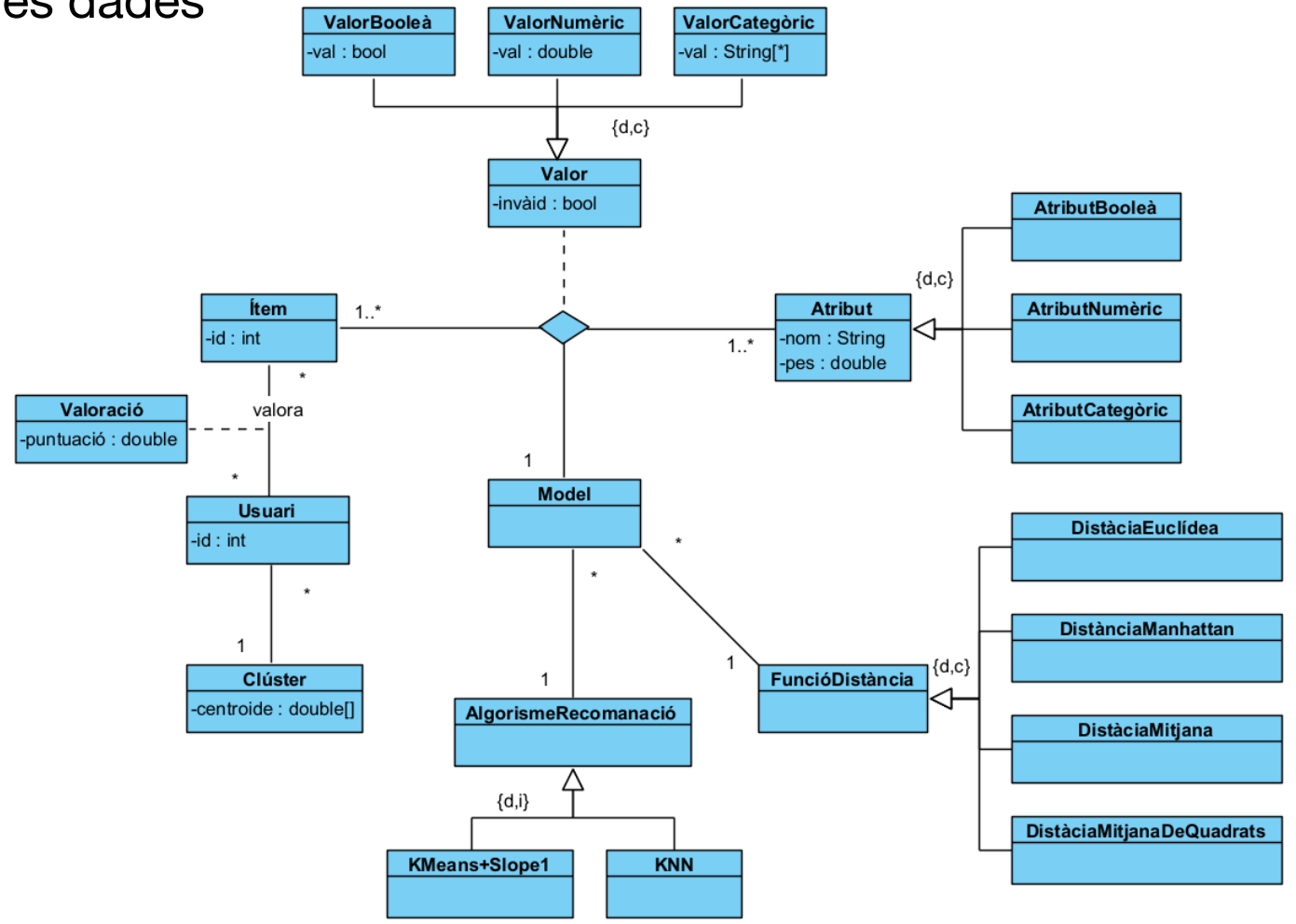
1. El Sistema informa l'Usuari actiu que el fitxer no existeix o no es pot llegir.
2. Tornar al pas 1 de l'escenari principal.

Notes:

- Implementa la funcionalitat obligatòria "Guardar i recuperar el resultat de l'algorisme principal (recomanacions)".

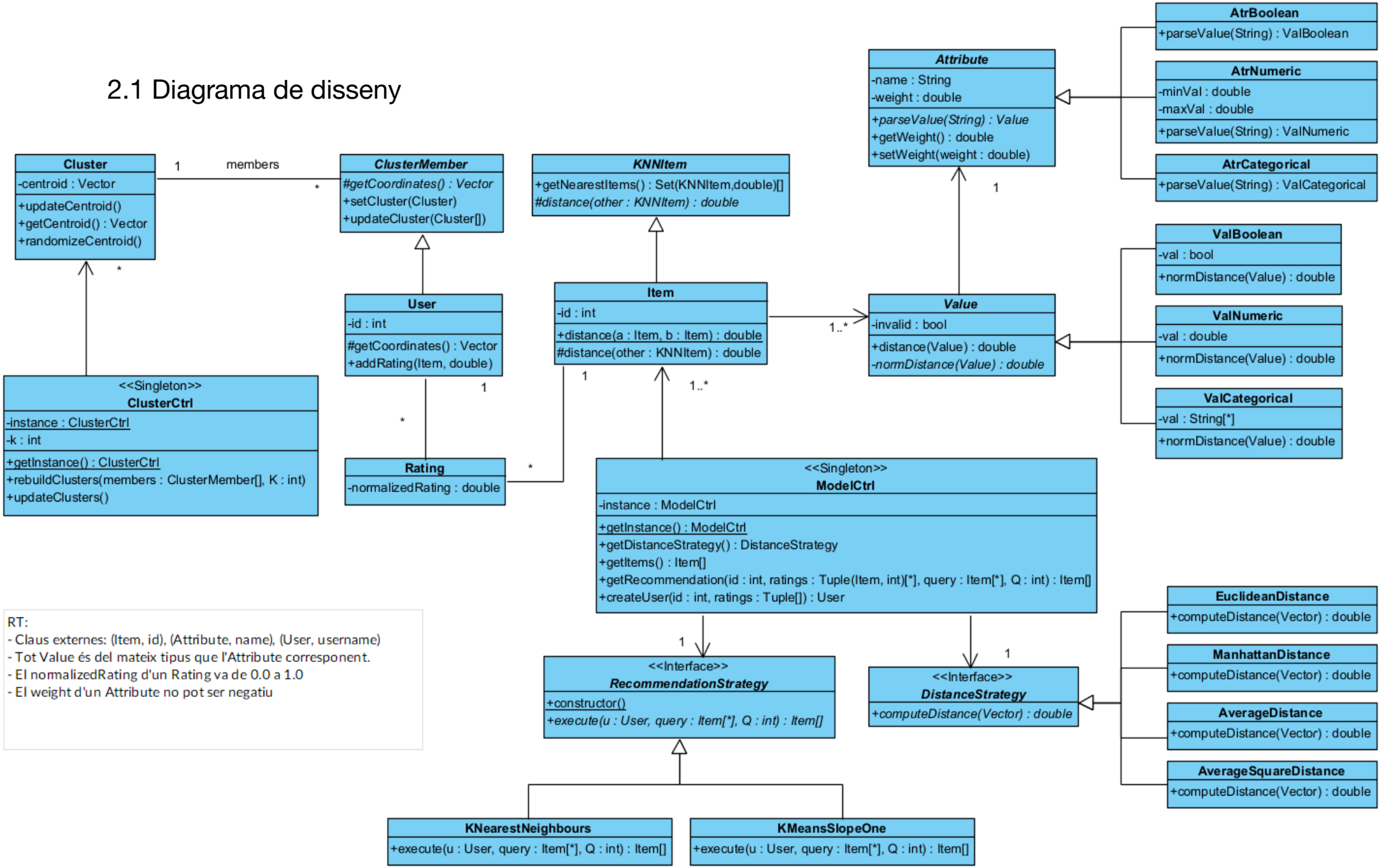
2. Model conceptual de les dades

2.1 Diagrama d'especificació



RT:
- Claus externes: (Item, id), (Atribut, nom), (Usuari, username)
- Un ValorAtribut sempre és del mateix tipus que l'Atribut al que correspon
- La puntuació d'una valoració està normalitzada (va de 0.0 a 1.0)
- El pes d'un Atribut no pot ser negatiu

2.1 Diagrama de disseny



RT:
- Claus externes: (Item, id), (Attribute, name), (User, username)
- Tot Value és del mateix tipus que l'Attribute corresponent.
- El normalizedRating d'un Rating va de 0.0 a 1.0
- El weight d'un Attribute no pot ser negatiu

Repartiment de classes:

- **Value (+fills):** Pol Rivero
- **Attribute (+fills):** Pol Rivero
- **DistanceStrategy (+fills):** Pol Rivero
- **ModelCtrl:** Pol Rivero
- **ClusterCtrl:** Alexandru Dumitru
- **Cluster (+ClusterMember):** Alexandru Dumitru
- **User:** Pol Rivero
- **Rating:** Pol Rivero
- **Item:** Pol Rivero
- **KNNItem:** Pablo José Galván
- **Recommendation (KNN):** Pablo José Galván
- **Recommendation (Slope1):** Ferran de la Varga

2.3 Descripció dels atributs i mètodes de les classes

Nota: S'ometen *setters*, *getters* i mètodes instanciadors d'altres classes.

Item: Ítem genèric guardat en el sistema (pot ser una pel·lícula, llibre, objecte...).
- *Atributs*: Identificador numèric.
- *Mètodes*: Calcular distància entre 2 Ítems, a partir de les distàncies entre els seus atributs.

KNNItem: Classe genèrica sobre la qual treballa l'algorisme K-nearest neighbours (no necessàriament ha de ser un Item, però en el nostre sistema sempre ho serà).
- *Mètodes*: Calcular els k KNNItems més propers.

User: Persona enregistrada al sistema.
- *Atributs*: Identificador numèric.
- *Mètodes*: Obtenir vector de les Valoracions fetes als Ítems valorats.

Rating: Puntuació que un Usuari dona a un Ítem.
- *Atributs*: Puntuació normalitzada (de 0.0 a 1.0).

Cluster: Encarregada de gestionar i agrupar els membres.
- *Atributs*:
  - `members`: conjunt de membres pertanyents a aquest clúster.
  - `coord`: coordenades on es troba el centroide a les N dimensions.
- *Mètodes*:
  - `divideBy`: divideix un `ArrayList` entre un altre.
  - `normalized_addBy`: suma dos `ArrayList` normalitzats i els manté dins el rang [0-1.0].
  - `sum`: métode que suma totes les coordenades de tots els membres i té el record de quantes valoracions s'han sumat a cada component, per després fer la mitjana.
  - `media_coordenadas`: calcula la mitjana de les coordenades de tots els membres actuals del clúster.
  - `check_coords`: compara la diferència entre centroides dins d'una tolerància, per retornar `true` o `false` segons si ha hagut una diferència superior a la tolerància a qualsevol component.
  - `recalculate_centroid`: assigna nou valor del centroide mitjançant la mitjana de coordenades dels membres actuals del clúster.
  - `randomize_centroid`: emplena tots els buits amb valor NaN d'un `ArrayList` amb nombres aleatoris dins d'un rang.

ClusterCtrl: L'encarregat de gestionar l'estat dels clústers, executant l'algorisme K-means.
- *Atributs*:
  - `instance`: emmagatzema la seva pròpia instància.

        - `clusters`: conjunt de clústers que hi ha al sistema.
        - k: quantitat de clústers utilitzats.
        - `initialized`: indica si el clúster està actualitzat pels membres actuals.
- *Mètodes*:
    - `create_clusters`: inicialitza tots els clústers amb una coordenada base.
    - `recalculate_centroids`: crida la funció de recalcular els centroides pels nous membres i indica si s'ha canviat algun o no.
    - `iterate_kmeans_algorithm`: realitza l'actualització de clúster de cadascun dels membres.
    - `run_kmeans`: operació on s'administra tot el funcionament de l'algorisme.
    - `distance_silhouette`: algorisme que calcula la distància entre dues coordenades, utilitzant l'estratègia de distància assignada al sistema.
    - `compute_silhouette`: algorisme principal per calcular la k ideal. Aquest mètode delega tot el treball a la funció `compute_silhouette` dins del membre. Per reduir el temps d'execució estem executant les delegacions en paral·lel.
    - `compute_wss`: així com `compute_silhouette`, delega tot el treball a la classe `ClusterMember` i ho realitza de forma paralel·la.
    - `rebuildClusters`: mètode encarregat d'inicialitzar els clústers, la k i d'executar l'algorisme.
    - `updateCluster`: mètode utilitzat per poder actualitzar l'algorisme amb la configuració i dades actuals en el sistema. És a dir, si s'elimina algun usuari o alguna valoració, poder actualitzar l'algorisme automàticament sense executar-lo des de 0.

ClusterMember: Classe genèrica sobre la qual treballa el ClusterCtrl (no necessàriament ha de ser un User).

- Atributs:
    - `cluster`: instància que indica a quina classe `Cluster` pertany el membre.
- *Mètodes*:
    - `getCoordinates`: classe abstracta encarregada d'obtenir les coordenades de les recomanacions del seu usuari. És a dir, és el conjunt de valoracions ordenades de la mateixa manera entre tots els usuaris.
    - `updateCluster`: és l'encarregat de calcular totes les distàncies d'aquest membre cap als clústers i assignar-lo al clúster amb menor distància.
    - `compute_wss`: calcular l'error al quadrat del membre cap al seu clúster.
    - `compute_silhouette`: calcula el valor Si d'un membre per l'algorisme principal de Silhouette.

Attribute: Paràmetre que té un ítem (per exemple, la durada de la pel·lícula). Pot ser de tipus booleà (`true`/`false`), numèric (coma flotant) o categòric (llista d'etiquetes).

- *Atributs*: Nom del paràmetre, pes en el càlcul de recomanacions.

Value: Valor que pren un Atribut en un Ítem concret. Pot ser de tipus booleà, numèric o categòric (el mateix tipus que l'atribut corresponent).

- *Atributs*: El valor corresponent.
- *Mètodes*: Calcular distància entre 2 valors (cal que siguin del mateix tipus).

ModelCtrl: L'encarregat de gestionar l'estat del model i emmagatzemar les opcions escollides per l'Admin.

- *Mètodes*: Obtenir les recomanacions per un usuari actiu.

RecommendationStrategy: Interfície que implementen els diferents algorismes de recomanació que es vulguin afegir al sistema.

- *Mètodes*: Executar algorisme de recomanació.

DistanceStrategy: Interfície que implementen els diferents algorismes de càlcul de distància que es vulguin afegir al sistema.

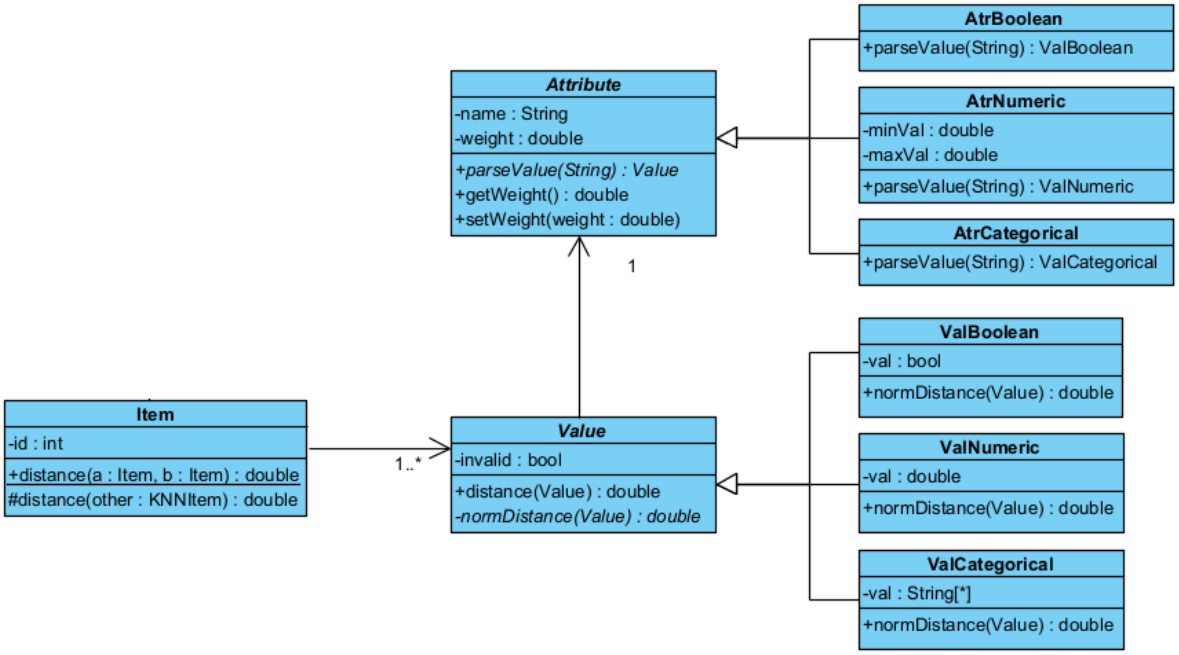
- *Mètodes*: Calcular distància, donat un vector de distàncies parcials.

3. Documentació

3.1 Funcionalitat principal / Model

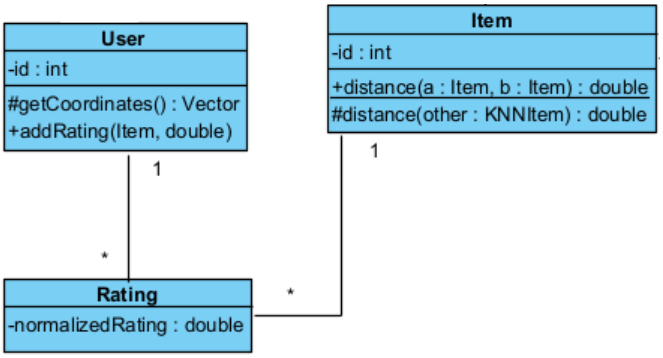
El model es podria dividir en 2 estructures de dades principals: la matriu de valors que prenen els atributs en cada ítem i la matriu de valoracions que els usuaris han fet de cada pel·lícula. Cal tenir en compte que la primera matriu és dispersa (*sparse*), mentre que generalment la segona és densa (*dense*).

La part del model encarregada d'implementar la matriu d'ítems i atributs és la següent:

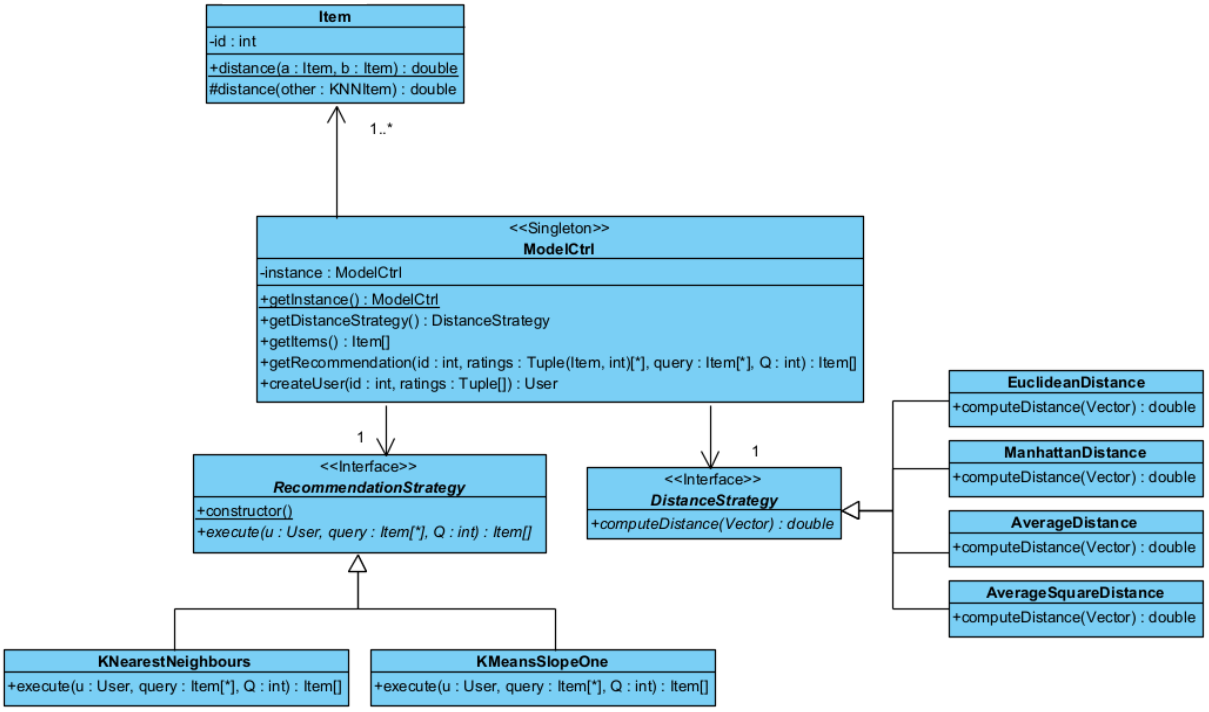


Cada ítem té navegabilitat a tots els seus valors, que són d'un dels tipus acceptats (booleà, numèric o categòric). Des del valor es té navegabilitat a l'atribut corresponent, que ha de ser del mateix tipus que el valor.
Per exemple, el ítem amb id 123 té navegabilitat a un valor categòric "`Titanic`" (que apunta al atribut categòric amb nom "Títol"), un valor numèric `120` (que apunta al atribut numèric amb nom "Duració") i un valor booleà `False` (que apunta al atribut booleà amb nom "Adulta").
L'administrador del sistema podrà invalidar els valors d'un atribut per a un conjunt d'ítems (de forma que no s'utilitzin en l'algorisme) i també podrà modificar el pes de cada atribut en l'algorisme de recomanacions, tot i que no formen part de la funcionalitat principal de la primera entrega.

La matriu *sparse* de valoracions de cada usuari s'implementa com una associació binària entre `User` i `Item`, amb una classe associativa `Rating` que conté la valoració normalitzada (entre 0.0 i 1.0) que l'usuari ha fet. La versió de disseny d'aquesta part del model és la següent:



Finalment, l'administrador ha de poder guardar les preferències de funció de distància i algorisme de recomanació a utilitzar. La classe encarregada de gestionar aquesta funcionalitat és un Singleton.



La primera versió del sistema té 3 objectius principals:

- Recomanació d'ítems utilitzant *collaborative filtering* (**explicat a l'apartat 3.2**): està basada en *K-means* i *Slope one*.
- Recomanació d'ítems utilitzant *content-based filtering* (**explicat a l'apartat 3.3**): està basada en *K-Nearest Neighbours (KNN)*.
- Avaluació d'un conjunt de recomanacions (**explicat a l'apartat 3.4**): utilitza el *Discounted Cumulative Gain (DCG)* de la permutació resultant.

3.2 Recomanació basada en K-Means i Slope1

3.2.1 Algoritme K-Means

L'algorisme K-Means serà l'encarregat d'agrupar usuaris sobre la base d'uns paràmetres, com per exemple, la valoració que han donat a uns certs ítems. Una de les característiques d'aquest algorisme és què funciona amb una sola variable com amb N variables, sent N un número bastant gran.

El funcionament de l'algorisme és el següent, tenint les variables de cada usuari aconseguim les seves coordenades en un espai N-Dimensional (sent N la quantitat de diferents variables que existeixen). A continuació, a través de fer la mitjana de totes aquestes coordenades, aconseguirem el centre de la mostra, que aplicant uns números aleatoris aconseguim generar uns centroides que ens permeten agrupar a tots els usuaris.

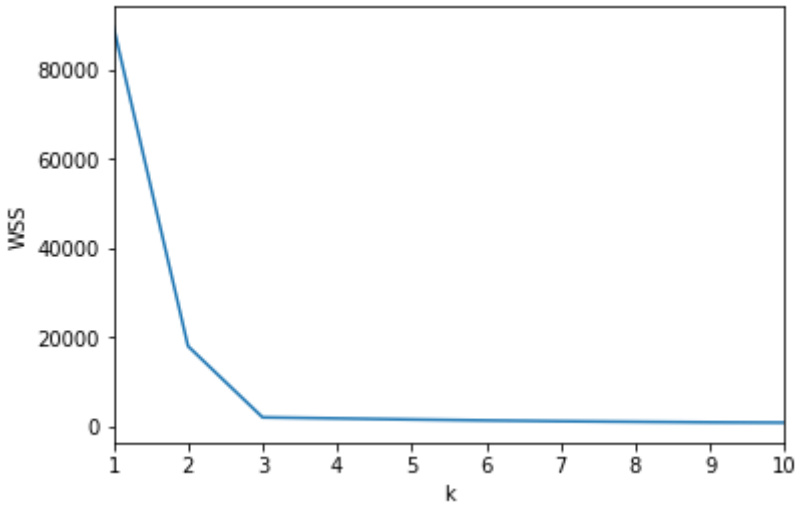
La quantitat de centroides que existiran depèn de la mostra oferta a l'algorisme, per aquest motiu, a través d'algorismes com Elbow Method o Silhouette Method, podem computar com seria l'ideal.

Tornant al funcionament de l'algorisme, el que anirà fent serà el següent:

1. Iterar a través de tots els usuaris calculant la seva distància entre tots els centroides i triant la menor. Destacar que quan hi hagi algun usuari que per a una de les variables no tingui valor, aquesta mateixa s'ignorarà.
2. Recalcular tots els centroides fent la mitjana de coordenades de tots els usuaris pertanyents a ell.
3. Repetir pas 1 i 2 fins a arribar al límit assignat en el programa o fins que, després de recalcular els centroides, no hi hagi un canvi molt significatiu.

D'altra banda, per a fer l'algorisme més independent, hem implementat que l'algorisme s'executi automàticament cada vegada que es crea un nou usuari. D'aquesta manera, ho tindrem actualitzat en tot moment.

En quant als algorismes per a identificar el k ideal, la primera implementació ha estat el Elbow Method, que sol ser el més utilitzat ja que la seva implementació és molt senzilla i eficient algorítmicament parlant. Aquest algorisme consisteix en calcular l'error al quadrat de cada punt cap al seu centroide i sumar-lo a una variable comuna. Una vegada executat per a diferents k, hem d'observar el punt on la funció comença a “aplanar-se” fent una forma de colze, com podem observar en la següent gràfica on la k ideal seria 3.



En quant al silhouette, encara que estigui implementat i pot arribar a ser automàtic, és molt ineficient ja que és un algorisme $\Theta(N^2)$ i fins i tot estant paral·lelitzat, continua trigant 10 minuts per cada k en un dataset gran, com per exemple Movielens 6750. Quant a la part automàtica, és molt més fàcil identificar la k ideal amb aquest algorisme ja que solament hem d'identificar el número màxim dels quals ens retorna l'algorisme, que treballa dins d'un rang [-1, 1].

En quant al càlcul de distàncies hem implementat quatre mètodes diferents:

- Distància Euclidiana
- Distància Manhattan
- Distància Mitjana distància
- Mitjana Quadrada

Finalment, per la eficiència, sabent que:

- k, nombre de clústers.
- N, nombre d'usuaris.
- Dim, nombre de variables (nombre d'ítems en el sistema).
- iter, nombre d'iteracions màximes

tenim una complexitat de $O(iter \cdot k \cdot N \cdot Dim)$ i una $\Omega(k \cdot N \cdot Dim)$. Assumint que k i iter són valors constants podem suposar una complexitat de tal $\Theta(N \cdot Dim)$.

3.2.2 Algoritme Slope1

Donat un usuari *u* i un ítem *i*, l'algorisme Slope1 s'encarrega d'obtenir la possible valoració que donarà l'usuari *u* a l'ítem *i*.

Per dur-ho a terme, l'algorisme es basa en les valoracions que ha fet l'usuari *u* de tots els ítems que hi ha en el sistema (incloent-hi l'ítem *i*); i també es basa en les valoracions que han fet un grup d'usuaris de tots els ítems (incloent-hi també *i*).

Primer de tot vam pensar que podríem utilitzar com a grup d'usuaris tots els usuaris del sistema, però gràcies a l'algorisme K-Means, ens vam adonar que és millor utilitzar com a grup d'usuaris el clúster al qual pertany l'usuari *u*. D'aquesta manera, la possible valoració d'*u* sobre l'ítem *i* és més fiable, ja que només es té en compte les valoracions d'usuaris semblants a *u*, amb els mateixos interessos. D'altra banda és més eficient perquè només es recorren un subconjunt dels usuaris del sistema.

Més profundament, SlopeOne fa es basa en la regressió lineal: $f(x) = n \cdot \Delta x + m$. En aquest algorisme, $\Delta x = 1$ i la funció esdevé $n + m$.
Primer de tot, hi ha una funció `get_recommendations()` que retorna les possibles valoracions de tots els ítems que *u* no ha valorat. Aquesta, per a cada valoració no feta, crida a `slope_one()`, que retorna la possible valoració de l'ítem corresponent. Per a cada ítem del sistema *j* que l'usuari *u* ha valorat, es calcula una predicció de la valoració de *i*: és la suma de la valoració que l'usuari *u* ha fet a *j* (equivaldria a la *m* de la fórmula) més el que retorna l'`average_slope()` (que equivaldria la *n*). La predicció final és la mitjana de totes les prediccions.
Finalment, en quant a `average_slope()`, aquesta funció calcula el pendent mitjà entre *i* i *j*. Per cada usuari (del clúster al qual pertany *u*), calcula la diferència entre la valoració de l'ítem *j* i la valoració de l'ítem *i*.

Per tal que SlopeOne funcioni correctament, l'usuari *u* ha d'haver valorat almenys un ítem i com a mínim hi haurà un usuari en el clúster (l'usuari *u*). Evidentment, com més usuaris hi hagi en el clúster i com més valoracions hagin fet aquests, més precís serà la possible valoració de l'usuari *u*.

Sigui:

- uc el nombre d'usuaris del clúster
- it el nombre d'ítems
- $ival$ el nombre d'ítems valorats per l'usuari actiu

llavors el cost de predir la valoració de 1 item amb SlopeOne és: $\Theta(it + ival \cdot uc)$.

3.3 Recomanació basada en K-Nearest Neighbours

Per obtenir recomanacions per a l'usuari actiu basades en els ítems que li han agradat, es retorna una llista d'ítems ordenada descendentment segons la probabilitat estimada que a l'usuari li agradin a partir d'una llista d'ítems valorats per aquest usuari, un subconjunt dels ítems existents i un nombre d'ítems a seleccionar per la recomanació. Això s'aconsegueix fent ús de l'algorisme *K-Nearest Neighbours*.

3.3.1 Algorisme K-Nearest Neighbours

Per començar, *K-Nearest Neighbours* té com a funcionalitat trobar els k (o menys si la mostra no en té suficients elements) ítems a menys distància d'un altre ítem donat. Aquest paràmetre k és arbitrari, i dependrà de si volem que un ítem valorat afecti a més o menys ítems del conjunt a ordenar.

Per aconseguir aquest resultat, `KNNItem` té implementat un mètode `getNearestItems`, el qual, donada una llista d'n ítems i un enter k, retorna una llista de tuples de valors enters i reals: l'enter indica l'índex d'un ítem a la llista, i el real la distància a la que es troba aquest element de l'ítem sobre el que s'executa el mètode. La llista de tuples tindrà k elements com a màxim (només en tindrà menys si n és menor que k, i en aquest cas es retornaran n tuples), i representarà els ítems de la llista més propers a l'ítem de referència (que anomenarem **veïns**) en ordre creixent segons la distància.

Aquest mètode està implementat mitjançant una `PriorityQueue` de capacitat màxima fixada a k. D'aquesta forma, aquesta mai tindrà més de k elements, i ja que s'ordena mitjançant un *heap* (de manera que, considerant que el nombre d'atributs és constant, afegir un element té cost $O(log(k))$), afegir els n elements de la llista d'entrada a la cua té cost $O(n{\cdot}log(k))$. L'única altra acció del mètode amb cost no constant és la conversió de `PriorityQueue` a `ArrayList`, on simplement es crea una nova instància d'`ArrayList` i s'hi afegeixen els elements de la cua un a un, i això també té cost $O(n{\cdot}log(k))$. Per tant, el cost total de l'algorisme és $O(n{\cdot}log(k) + n{\cdot}log(k)) = O(n{\cdot}log(k))$.

3.3.2 Content-based filtering: càlcul d'afinitats

No obstant tot això, l'algorisme *K-Nearest Neighbours* sol, no aconsegueix el nostre objectiu, ja que és capaç d'ordenar una llista d'ítems segons la seva semblança a un ítem concret, però nosaltres necessitem un algorisme que faci això segons tots els ítems que li agradin a l'usuari actiu.

Per aquesta raó, hem creat un algorisme per obtenir aquest resultat executant *K-Nearest Neighbours* per cada ítem `i` valorat per l'usuari (que tingui una valoració superior o igual a un valor arbitrari anomenat `THRESHOLD`) i obtenint un valor (que anomenarem **afinitat**) per cada veí d'aquest ítem `i`, que és major com més probable és que a l'usuari actiu li agradi l'ítem (per defecte és -1). Al final de l'execució, els ítems seran ordenats segons la mitjana de les afinitats trobades per cada ítem.
Sigui:

- I_i un ítem valorat per l'usuari en la posició i de la llista amb valoració R_i superior o igual a `THRESHOLD`.
- V_j el j veí més llunyà d'aquest ítem.
- $D(I_i, V_j)$ la distància entre I_i i V_j.
- A_{V_j} la nova afinitat per afegir a les afinitats trobades per aquest veí.

Tenim 3 mètodes per trobar les afinitats dels ítems:

- `NORMAL`: $A_{V_j} = R_i$. Aquest mètode és el més senzill i ràpid, però té un problema greu: les distàncies dels veïns als ítems valorats en si no tenen cap pes en l'afinitat (és a dir, que, siguin V_0 i V_1 dos veïns d'I_i, sempre es compleix que $A_{V_0} = A_{V_1}$, fins i tot si $D(I_i, V_1)$ és molt més gran que $D(I_i, V_0)$). Per tant, aquest mètode pot no reflectir correctament les diferències entre els ítems valorats i els seus veïns.
- `LOG`: $A_{V_j} = \frac{R_i}{log(10 + D(I_i, V_j))}$. Aquesta forma d'obtenir l'afinitat no és excessivament lenta, però, a diferència de l'anterior mètode, sí utilitza les distàncies dels veïns per reduir l'afinitat. El motiu pel qual s'utilitza un logaritme i se suma 10 al seu paràmetre i no es divideix directament per la distància és per dos motius: d'una banda, a priori no sabem quina pot ser la grandària de les distàncies, i pot donar-se el cas que aquestes són massa grans, causant que les afinitats siguin massa petites, cosa que es pot alleujar utilitzant una escala logarítmica per la distància; d'altra banda, d'aquesta manera s'evita la possibilitat de divisió entre zero, ja que el paràmetre del logaritme serà 10 com a mínim, de forma que el valor mínim del logaritme és 1 (i així, si la distància és 0, l'afinitat serà exactament R_i, ja que si dos ítems són iguals, tindria sentit que rebessin la mateixa valoració).
- `ORDERED`: el mètode `LOG` és ràpid i pot donar bons resultats, però es pot donar el cas on no vulguem descartar un ítem de la llista a ordenar si és veí d'un ítem ben valorat per l'usuari actiu, fins i tot si aquest ítem és llunyà, però `LOG` penalitza molt els ítems distants. Per provar si l'alternativa proposada funciona millor, s'ha inclòs el

mètode ORDERED, que permet que ítems distants tinguin una bona posició si són veïns d'ítems ben valorats.

Aquest mètode comença ordenant la llista d'ítems valorats per l'usuari segons les valoracions en ordre decreixent. A continuació, per cada I_i troba la distància màxima D_{max} a la qual es troba d'un veí seu, i per cada V_j calcula un nombre $x = (R_i - R_{i+1}) \cdot \frac{D(I_i, V_j)}{D_{max}}$, on R_{i+1} o bé és la valoració immediatament inferior de la llista o bé, si no hi ha més valoracions superiors o iguals a TRESHOLD a continuació, simplement és TRESHOLD. L'afinitat nova serà $A_{V_j} = R_i - x$, que sempre donarà un nombre en l'interval $[R_{i+1}, R_i]$ i estarà interpolat segons la distància del veí al ítem valorat: si aquesta és D_{max} l'afinitat serà R_{i+1}, i si aquesta és 0 l'afinitat serà R_i.

Es pot veure que aquest algorisme és més costós que els altres dos, però ens permetrà comprovar si l'alternativa explicada dona millors resultats que LOG.

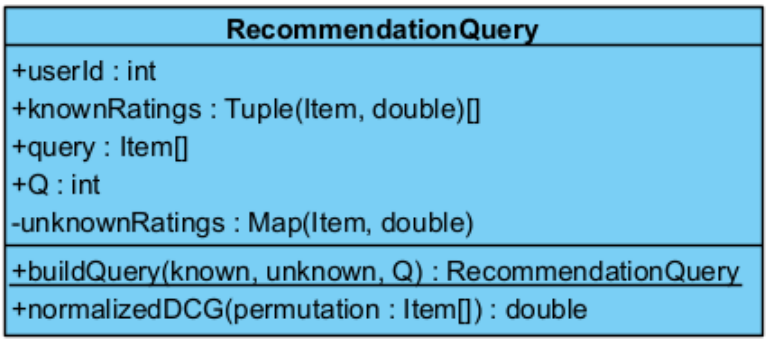
Finalment, sigui $|R|$ el nombre d'ítems valorats per l'usuari actiu, $|Q|$ la mida de la llista d'ítems a partir de la qual es genera la recomanació i k el nombre de veïns per ítem valorat màxims, el cost temporal d'aquest algorisme és $O(|R| \cdot |Q| \cdot log(k))$, ja que per cada ítem valorat es calculen els seus k veïns més propers amb l'algorisme de *K-Nearest Neighbours*, i cada càlcul d'aquests té, com s'ha calculat a la secció anterior, un cost temporal de $O(|Q| \cdot log(k))$. Hi ha altres bucles que iteren sobre els veïns o els ítems valorats, i en el mètode ORDERED també s'ordenen els ítems valorats al principi, però cap d'aquests supera el del càlcul d'afinitats, de forma que el cost total és $O(|R| \cdot |Q| \cdot log(k))$.

3.4 Avaluació d'un conjunt de recomanacions

Un cop s'ha executat l'algorisme en qüestió, es puntua el seu resultat calculant el *Discounted Cumulative Gain (DCG)* de la permutació que ha retornat.

Per avaluar el model, l'usuari ha de llegir 2 fitxers. El primer codifica una matriu *Known* que conté un conjunt d'usuaris nous (no existeixen al model) i les valoracions que han fet a alguns ítems (ja existents). Aquestes valoracions seran utilitzades per l'algorisme de recomanacions a l'hora de fer prediccions.
El segon fitxer codifica una matriu *Unknown* que conté, per cada usuari del conjunt anterior, una sèrie de valoracions que han fet a ítems ja existents (diferents dels anteriors). Aquestes valoracions són les que l'algorisme haurà de de predir (i aquestes "autèntiques valoracions" són les que s'utilitzen per avaluar la qualitat del model).

El sistema compta amb una classe especialitzada per gestionar aquest procés. Donats els conjunts de valoració *known* i *unknown*, i la mida Q del resultat, crea les queries en el format compatible amb els algorismes de recomanació i guarda l'autèntic resultat per calcular posteriorment el DCG.



El `ModelCtrl` rep les `RecommendationQuery` i s'encarrega de crear l'usuari actiu (identificat pel camp `userId` de la query) i actualitzar les estructures de dades dels algorismes de recomanació, en cas que sigui necessari. Posteriorment, passa l'usuari, els ítems de la query i la mida Q als algorismes de recomanació. Finalment, esborra l'usuari actiu i retorna la permutació calculada per l'algorisme de recomanació.

És aleshores quan s'utilitza `RecommendationQuery::normalizedDCG()` per calcular la qualitat de la permutació retornada, en forma de DCG normalitzat.

Donada una permutació `Perm` amb Q elements dels quals coneixem l'autèntica puntuació donada per l'usuari (guardada a `RecommendationQuery`), el càlcul del DCG consisteix en realizar el següent sumatori:

$$\sum_{i=1}^{Q} \frac{2^{Perm[i].trueRating}-1}{log(i+1)}$$

Per rebre una puntuació normalitzada entre 0.0 i 1.0, podem calcular el DCG ideal (és a dir, el DCG de la permutació òptima, aquella que està ordenada segons les autèntiques valoracions) i fer la divisió:

$$NormalizedDCG = \frac{DCG(Perm)}{DCG(Permutació\ òptima)}$$