Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Recomanació de llibres

Sistemes Basats en el Coneixement

Aubach Altès, Artur
Fuster Palà, Llum
Gálvez Serrano, Paula
Molina Sedano, Òscar

Índex

1 Introducció

En l'era actual de l'abundància d'oferta i la creixent presència de llibreries en línia, la elecció d'una lectura adequada pot resultar abrumadora. Per abordar aquest problema, s'han desenvolupat sistemes de recomanació de llibres que busquen proporcionar als lectors suggeriments personalitzats basats en una varietat de factors.

Concretament, aquesta pràctica es centra en el desenvolupament d'un Sistema Basat en Casos per a la recomanació de llibres, utilitzant el Raonament Basat en Casos. Aquest treball es construeix sobre els fonaments establerts en la pràctica prèvia, on es va implementar un sistema inicial de recomanació de llibres utilitzant el programari CLIPS. L'elavoració de la pràctica anterior va ser fonamental per adquirir una comprensió més profunda del domini de recomanació de llibres, amb un enfocament particular en les novel·les de ficció i els perfils dels usuaris.

1.1 Descripció del domini

L'objectiu principal d'aquest projecte és desenvolupar un sistema capaç de proporcionar recomanacions de fins a tres llibres. Aquestes recomanacions es personalitzen segons les necessitats específiques de cada usuari, basant-se en les dades obtingudes de la seva interacció amb el sistema.

Un dels elements més innovadors d'aquesta pràctica és la inclusió d'un sistema de feedback de l'usuari. Mitjançant aquest sistema, els usuaris poden expressar el seu grau de satisfacció amb les recomanacions rebudes.

Per a això, disposem d'un **historial de casos**. Cada cas inclou el perfil d'un usuari amb les seves interaccions, les recomanacions que ha rebut i el feedback donat respecte a aquestes. Aquest historial és fonamental per al funcionament del sistema, ja que es basa en casos. Cal destacar que aquest historial és dinàmic, enriquit cada vegada que es fa una nova recomanació, si es requereix. Això facilita l'aprenentatge de nous casos, característica inherent al CBR.

El **perfil de l'usuari** recull informació clau com els gèneres preferits, la longitud habitual de llibres que llegeix, entre altres. Això ens ajuda a definir el tipus de lector.

En variació del projecte anterior, en aquest els **llibres** ocupen un lloc secundari. Només es consideren en relació al perfil de l'usuari i es tenen en compte les valoracions que els usuaris han fet de cada llibre.

La **recomanació** es realitza un cop hem obtingut un perfil de l'usuari, i es basa en el sistema de les **4 R's**. La fase de recuperació selecciona perfils d'usuari similars al que s'estudia. En la fase de reutilització, s'adapten les solucions de casos similars per generar la recomanació. Després, en la fase de revisió, s'avalua si la recomanació necessita modificacions.

Finalment, amb el feedback de l'usuari, obtenim un cas complet. Aleshores completem la última fase, de retenció, on decidim si incorporar-lo a la biblioteca de casos.

1.2 Seccions del document

El primer apartat tracta sobre la metodologia del nostre mètode de recomanació, on s'exposa l'origen de les dades utilitzades per al sistema de recomanació i les decisions clau que han configurat l'estructura del sistema.

El capítol següent avança cap a la formalització del nostre sistema, detallant amb profunditat el disseny i la construcció del mateix. Es descriuen les diverses classes i les seves interrelacions, proporcionant una visió de l'arquitectura i de la dinàmica de funcionament intern del sistema.

El tractament posterior se centra en la implementació específica, dividida segons les fases de les 4 r's: retrieve, reuse, revise i retain, juntament amb el manteniment del sistema. Aquesta secció descriu els processos algorísmics i lògics que defineixen el comportament del sistema de recomanació.

Un cop s'ha explicat el projecte en la seva totalitat, l'atenció es dirigeix cap als detalls com les mètriques i els casos d'estudi específics. Aquests detalls són fonamentals per a l'apartat d'experimentació, on es posa a prova l'eficàcia del sistema i s'avalua la seva capacitat de generació de recomanacions.

En la part final de la memòria, s'introdueix la interfície gràfica desenvolupada, la qual permet una interacció directa amb l'usuari.

Abans de concloure amb el capítol de conclusions, es dedica una secció a explorar idees addicionals que han sorgit durant el desenvolupament del projecte. Aquestes idees reflecteixen el pensament crític de l'equip sobre com pot ser millorat el sistema de recomanació, indicant camins per a futures investigacions i desenvolupaments.

2 Mètode de recomanació

2.1 Obtenció de dades

2.1.1 Origen de les dades

Hem utilitzat un conjunt de dades extret de aquesta pàgina web. Aquestes dades van ser recollides a finals de 2017 des de `goodreads.com`. Per a la seva obtenció, es van extreure únicament les pàgines dels usuaris amb estanteries públiques, accessibles per a qualsevol sense necessitat d'iniciar sessió. És important destacar que tant els ID d'usuari com els de les revisions són anònims, garantint així la privacitat dels usuaris.

En la pàgina web destaca que el conjunt de dades recollit es va obtenir amb fins exclusivament acadèmics. En cap moment es pretén la seva redistribució ni el seu ús amb propòsits comercials. Per tant, compleix amb els requisits ètics i legals necessaris per a la seva utilització en el nostre projecte. Aquest conjunt de dades inclou estadístiques bàsiques impressionants: 2.360.655 llibres escrits per 829.529 autors. A més, consta de 876.145 usuaris i un total de 228.648.342 interaccions entre usuaris i llibres, que inclouen tant lectures com valoracions.

De tota la informació que conté la nostra font d'informació, nosaltres ens hem quedat amb els tres axius principals:

- `goodreads_interactions.csv`, que conté tots els llibres que ha llegit cada usuari amb la puntuació que els hi han donat (del 0 al 5), i si els han posat alguna ressenya. Aquest arxiu té una mida de 4.1 GB.
- `goodreads_books.json.gz`, que conté informació detallada de tots els llibres (nom, format, nombre de pàgines, isbn...). Aquest arxiu té una mida de 2 GB.
- `goodreads_book_genres.json`. En l'aplicació de *goodreads*, quan l'usuari acaba de llegir un llibre i l'ha de valorar, ha d'introduir a quin gènere considera que pertany el llibre. Aquest arxiu conté els resultats de totes les puntuacions de gènere de cada llibre.

2.1.2 Preprocessat

Cal tenir en compte que, per a una manipulació àgil i eficient de les dades, s'ha de reduir la mida dels arxius, ja que són massa grans.

En tots els arxius apareix la columna de `book_id`, però cada base de dades conté un conjunt de llibres diferents. Per tant, el primer pas per a la reducció de dades ha estat seleccionar només aquells llibres presents en els tres arxius.

Posteriorment, i sabent que l'arxiu de llibres conté informació més detallada de la desitjada, s'ha procedit eliminar aquelles columnes que a priori ja se sap que no seran útils. D'aquesta manera, únicament s'han mantigut les columnes: book_id, num_pages, format, ratings_count, average_rating, title.

Després, s'ha transformat l'arxiu de `goodreads_interactions` en un conjunt de dades genèriques sobre cada usuari. És a dir, en lloc de tenir una fila per cada llibre llegit, amb la seva puntuació i si ha estat comentat, s'ha creat una fila per usuari amb una llista dels llibres llegits, la mitjana de puntuacions i el percentatge de llibres comentats. Això ha permès començar a perfilar informació rellevant de l'usuari, essent aquesta la base per a cada perfil. Aquesta nova base de dades s'ha desat en un arxiu anomenat raw_problem.csv, i conté les columnes: user_id, read_books, avg_rating, review_proportion.

A partir d'aquí, s'ha seguit una sèrie de passos per definir més acuradament el perfil del lector i els casos:

- S'ha eliminat 'review_proportion' de rawproblems.csv, considerant que el percentatge de lectors que escriuen ressenyes és baix i, per tant, les dades no aporten valor significatiu (resultant poc útils i complicades de tractar).

- S'ha afegit a l'arxiu books.csv les columnes de gèneres de l'arxiu goodreads_book_genres.
- S'han revisat els títols dels llibres per comprovar el seu format, detectant que alguns no estaven en lletres itàliques (i per tant eren de illegibles per a nosaltres). Per això, s'han filtrat els llibres segons el nom, mantenint només aquells amb caràcters itàlics. També s'ha eliminat informació addicional dels títols, com el número d'edició o l'editorial.
- En eliminar llibres de books.csv, també ha estat necessari actualitzar la base de dades raw_problems, modificant la variable 'read_books'.
- S'han filtrat els usuaris per incloure només aquells que han llegit més de 10 llibres. Això es justifica pel fet que gran part de la informació relativa al lector es basa en els llibres llegits, requerint una quantitat suficient per extreure conclusions sòlides.
- S'ha afegit a raw_problems una variable 'counts' que indica el nombre de llibres llegits per cada usuari.
- Utilitzant la base de dades de llibres, s'ha calculat la mitjana de pàgines que l'usuari acostuma a llegir, afegint-la com a columna en l'arxiu dels lectors.
- De la mateixa manera, s'ha afegit la mitjana de gèneres llegits per cada usuari.
- Amb la finalitat de millorar la interacció del sistema amb l'usuari, i considerant que es demanarà la preferència de gènere, s'ha observat que disposar d'informació decimal resulta massa inespecífica. Per això, s'ha optat per crear un rànquing dels 10 gèneres principals, assignant el número 10 al gènere amb major percentatge i el número 1 al de menor.
- S'ha seleccionat els tres últims llibres llegits de 'read_books' per a cada usuari, tractant-los com a recomanacions. També s'ha registrat la puntuació d'aquests llibres.
- S'ha modificat l'ordre de les variables per facilitar la manipulació posterior de les dades.

Aquesta nova base de dades s'ha desat en l'arxiu `problems.csv`. A més, considerant que en aquest arxiu definitiu només s'inclouen els IDs dels llibres, els quals són més pràctics per a la manipulació, hem creat un diccionari on la clau és book_id i el valor corresponent és el títol del llibre. Aquest diccionari s'ha guardat en un fitxer anomenat books_dict.txt.

2.2 Descripció i desicions del procés de recomanació

En el desenvolupament del nostre sistema de recomanació, s'ha optat per un **enfocament híbrid**, combinant tècniques de Sistemes Basats en Casos (SBC) amb elements de l'aprenentatge automàtic. Aquesta decisió s'ha pres amb l'objectiu d'aportar major robustesa i precisió al nostre model.

En les etapes de 'retrieve', 'reuse', 'retain' i manteniment, el sistema s'ha centrat en l'ús exclusiu de mètodes propis dels CBR purs. És a dir, es busquen en la base de dades casos anteriors que presentin similituds amb el problema actual i s'utilitzen les solucions d'aquests casos com a base per a resoldre el nou problema. Aquest procés es basa en el principi que situacions similars requereixen solucions semblants.

No obstant això, per a l'etapa de l'avaluació del resultat obtingut, s'ha implementat un model de machine learning, concretament el **xgboost**. Aquest model ha servit per predir quina seria la valoració real de l'usuari, i la compara amb els resultats obtinguts pel nostre CBR. A partir d'aquesta comparació s'extreu com de bo és el nostre sistema. Tot i que l'aplicació del model no serveix per a generar la resposta com a tal, si que forma part del procès de les 4 r's característic del CBR. A més, l'ús d'aquest model ha permès reforçar i aportar una major qualitat a les respostes del sistema, oferint una justificació més fonamentada i precisa.

Tot i que les etapes inicials del procés són pures, l'aplicació d'un model extern com el xgboost converteix el nostre sistema en híbrid. Aquest enfocament híbrid permet combinar el millor dels dos mons: la capacitat dels SBC per a tractar casos específics basant-se en experiències anteriors i la potència del machine learning per a optimitzar les decisions i les valoracions.

2.3 Estructura del Sistema

Existeixen tres maneres principals d'enfocar la estructura dels sistemes de recomanació. Per al desenvolupament del nostre sistema, s'ha seleccionat un enfocament col·laboratiu, que prioritza les interaccions i valoracions dels usuaris per sobre de les característiques intrínseques dels ítems, tal com es fa en les estructures basades en contingut.

Aquesta **metodologia col·laborativa** resulta especialment efectiva en àmbits on els gustos i comportaments passats dels usuaris són indicadors claus dels seus interessos futurs. Atès que la naturalesa de les nostres dades es basa gairebé només en les experiències prèvies dels usuaris i no en qualitats inherents a aquests, s'ha determinat que aquest enfocament era la millor decisió.

A més, aquest enfocament permet al sistema oferir recomanacions dinàmiques i personalitzades, adaptant-se contínuament als canvis en les tendències i preferències dels usuaris al llarg del temps.

3 Disseny del sistema CBR

3.1 Estructura d'un Cas

La estructura dels nostres casos s'ha basat en les definicions proporcionades per les diapositives de teoria, on es defineix que un cas és un conjunt de característiques o *features* sobre un problema, la seva resolució, l'èxit del cas i la seva utilitat. Per això, s'ha creat la classe `Cas`, que inclou les següents característiques:

1. ***id***: Identificador únic del cas.
2. ***descripcio***: Informació detallada sobre l'usuari, és a dir, el problema a tractar. Aquesta informació no es presenta en format vector atribut-valor, sinó que les variables han estat escalades per a facilitar la seva manipulació.
3. ***derivacio*:** Aquesta informació es troba només en els nous casos incorporats al CBR. Consisteix en una llista dels casos més similars que ha utilitzat el CBR per a extreure la solució del problema.
4. ***solucio***: Llista dels identificadors dels tres llibres recomanats.
5. ***avaluacio***: Per a cada llibre recomanat, la puntuació atorgada per l'usuari, en una escala de l'1 al 5.
6. ***historial***: Llistat dels llibres llegits anteriorment per l'usuari.
7. ***UM***: En el procés de recomanació s¡aplica una fórmula per calcular la mesura d'utilitat d'un cas. Aquest atribut correspon a aquesta mesura d'utilitat. Els valors considerats per calcular-la són els següents.
8. ***UaS***: Nombre de vegades que s'ha utilitzat el cas per a predir un nou cas, on almenys un dels llibres de la recomanació és comú amb el cas anterior i la valoració d'aquest nou usuari ha estat satisfactòria.
9. ***S***: Nombre de vegades que s'ha utilitzat el cas per a predir un nou cas, i la valoració d'aquest nou usuari ha estat satisfactòria.
10. ***UaF***: Nombre de vegades que s'ha utilitzat el cas per a predir un nou cas, on almenys un dels llibres recomanats és comú amb el cas anterior i la valoració d'aquest nou usuari ha estat dolenta.
11. ***F***: Nombre de vegades que s'ha utilitzat un cas per a predir un nou cas, i la valoració d'aquest nou usuari ha estat dolenta.

3.2 Plantejament sobre el tipus d'indexació

En aquesta secció de l'informe, es discuteixen les diferents opcions d'indexació d'un CBR, justificant la decisió presa pel nostre equip. Es compara primer la memòria plana amb l'estructurada, i després es detallen els diferents tipus de memòria estructurada, com ara l'arbre de característiques discriminants o compartides. La informació per a aquesta discussió prové de les diapositives vistes a teoria sobre *Çomponents dels Sistemes CBR"*.

3.2.1 Memòria Plana vs Estructurada

A continuació s'explica els avantatges i desavantatges de l'ús de la memòria plana i estructurada per a l'indexació de casos en un CBR:

Memòria plana La memòria plana emmagatzema tots els casos en una única col·lecció sense una estructura jeràrquica ni agrupacions específiques. Aquest enfocament destaca per la seva simplicitat, essent fàcil d'implementar i mantenir, així com per la seva flexibilitat en permetre modificacions i actualitzacions directes de la base de dades de casos.

No obstant això, presenta desavantatges, com una eficiència de recerca reduïda en grans volums de dades i la manca d'un context natural per capturar relacions o jerarquies entre casos.

Memòria estructurada Per contra, la memòria estructurada organitza els casos segons una estructura jeràrquica o en grups, basada en característiques comuns o criteris específics. Aquesta aproximació millora l'eficiència en la recerca, especialment en bases de dades grans, mitjançant la classificació de casos en subgrups. També facilita la representació de relacions i similituds entre casos, proporcionant un context addicional.

No obstant això, la seva principal desavantatge és la necessitat d'una planificació i implementació més complexa.

Decisió Després d'una anàlisi detallada, es va optar per implementar una **indexació estructurada**, basant aquesta elecció en la necessitat d'eficiència en la recerca i la capacitat d'estructurar i contextualitzar millor els casos. Aquesta decisió permetrà gestionar eficientment l'actual conjunt de casos i adaptar-se a l'escalat i complexitat creixents de la col·lecció de llibres.

3.2.2 Arbres de Característiques: Discriminants vs Compartides

Dins de les memòries estructurades d'un CBR, existeixen diversos mètodes per generar arbres de característiques. A continuació, es discuteixen els avantatges i desavantatges dels arbres de característiques discriminants i compartides.

Arbres de Característiques Discriminants Els arbres de característiques discriminants classifiquen els casos segons criteris específics per distingir-los entre ells. És a dir, es centren en l'ús de característiques úniques o significatives que diferencien un cas d'un altre.

Com a avantatge, podem trobar que són molt precisos i específics. No obstant això, poden requerir una comprensió profunda del domini i ser més complexos de configurar i mantenir.

Arbres de Característiques Compartides Per contra, els arbres de característiques compartides són una estructura que agrupa casos segons característiques comunes. Són útils quan es desitja agrupar casos que comparteixen una sèrie d'atributs generals, facilitant una cerca més ràpida i simplificada. A més, són especialment efectius en entorns on la classificació precisa no és tan crítica o quan no es disposa de coneixement expert profund.

No obstant això, poden presentar una menor precisió en la classificació dels casos comparats amb els arbres discriminants.

Decisió Després d'analitzar detalladament ambdues opcions, s'ha decidit optar pels **arbres de característiques compartides**. Aquesta elecció s'ha basat en la manca de coneixement expert específic del nostre equip i en la necessitat d'una implementació més ràpida i flexible, adaptant-se millor a les condicions i recursos disponibles.

3.3 Decisió Final i implementació de la indexació

En la fase inicial del projecte, es va considerar implementar una **memòria estructurada** de **característiques compartides** seguint una estructura d'**arbre binari**. Aquesta estructura preveia que cada node no fulla actuaria com a model de clúster per determinar l'assignació dels elements segons clústers similars, mentre que els nodes fulla representarien un cas únic. No obstant això, s'ha detectat

que una estructura estrictament jeràrquica provoca una alta quantitat de nodes, augmentant el cost de trobar casos similars.

Per afrontar aquest problema, s'ha decidit adoptar una **estructura mixta**, que combina elements de la memòria plana amb la memòria estructurada.

Concretament, cada node fulla de l'arbre, en lloc de contenir un únic cas, alberga un conjunt de casos, reduint la profunditat de l'arbre i el nombre de passos necessaris en la recerca. Aquesta metodologia facilita una recerca més ràpida, ja que cada node representa un grup de casos amb característiques similars.

Aquesta decisió aporta un millor equilibri entre la complexitat de la gestió de l'arbre i la flexibilitat en el maneig de grans volums de dades, millorant significativament l'eficiència en la localització de casos similars.

Finalment, en la nostra recerca de l'estratègia d'indexació més adequada per al nostre sistema de recomanació de llibres basat en CBR, s'ha optat per implementar un mètode de clustering, específicament l'algoritme **K-means**. K-means és un mètode popular i eficient de clustering que agrupa els casos en clústers basats en les seves característiques. Es va triar per la seva simplicitat i eficiència en l'entrenament, així com per la seva capacitat de permetre especificar el nombre de clústers, facilitant una categorització clara dels casos.

Per a la mesura de distància dins de l'algoritme K-means, s'ha seleccionat la **distància euclidiana**, considerant la seva eficàcia en contextos amb dades numèriques i proporcionant una mesura clara i intuitiva de la similitud entre casos. L'ús de K-means amb la distància euclidiana representa una solució equilibrada que combina eficiència i simplicitat, facilitant així la recomanació precisa de llibres en el nostre sistema CBR.

3.3.1 Ontologia

La nostra ontologia defineix les classes que representen els components fonamentals el nostre recomanador CBR.

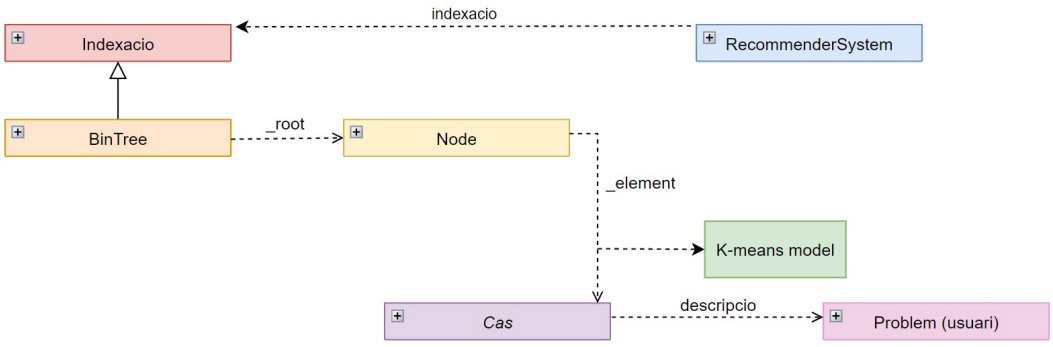

Figura 1: Ontologia

El nostre sistema es centra en un `RecomenderSystem` que conté diferents atributs. A més té una relació amb `Indexacio`, que és una de les principals parts del sistema CBR. Aquesta indexació és una subclasse de `BinTree` i en els nodes d'aquest arbre tenim llistes de instàncies de `Cas`, en cas de que siguin fulles, i en cas que no ho siguin, models Kmeans entrenats per a fer la separació dels clusters quan entra un nou cas. A més, la representació del cas té una característica descripció definida per una instància de `Problem`, classe que representa les preferències de l'usuari. A continuació podem veure tots els atributs, mètodes i propietats que té cada una de les classes:

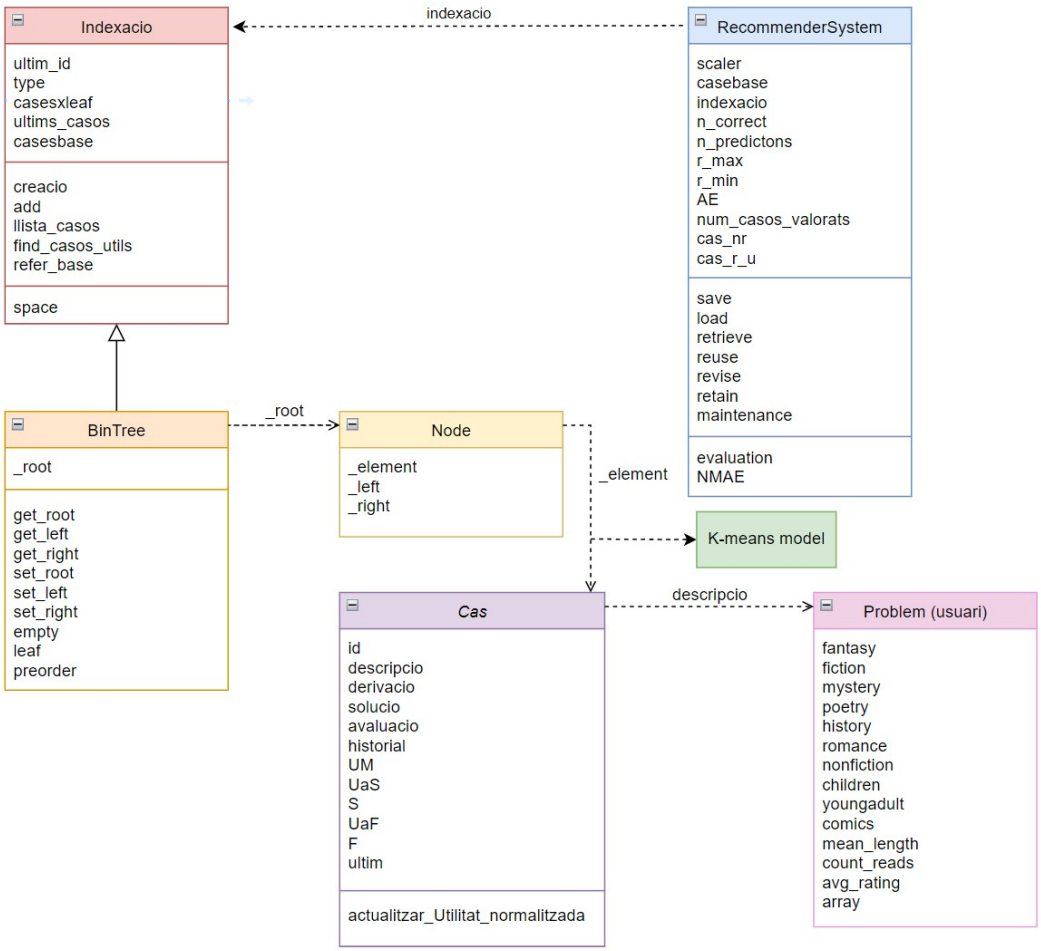

Figura 2: Ontologia amb atributs

Dins la representació de cada classe en la Figura 2 veiem en el primer requadre els atributs, en el segon els mètodes i en el tercer les propietats.

3.4 Metodologia Incremental i Primer Prototip

En aquest apartat descriurem l'estructura del codi del nostre CBR. A més a més presentarem el primer prototip que vam crear i del qual han anat construint-se les noves versions fins arribar a la final. En els següents apartats es presentaran les versions més noves de cada punt.

El nostre primer prototip es pot trobar al directori `miniCBR` amb la següent estructura:

```
|-- miniCBR
|   |-- cases
|   |   |-- bintreelinked.py
|   |   |-- case.py
|   |   |-- index.py
|   |   |-- __init__.py
|   |   |-- problem.py
|   |-- fases
|   |   |-- __init__.py
|   |   |-- retain.py
|   |   |-- retrieve.py
|   |   |-- reuse.py
```

```
|  |  |-- revise.py
|  |-- __init__.py
|  |-- recommender.py
```

Com es pot veure a l'arbre de fitxers i directoris, un CBR es conformarà per una carpeta on tindrem l'**ontologia** i tots els seus integrants (que hem anomenat *cases*), una altra per a emmagatzemar les funcionalitats i la lògica de les 4 fases (anomenada *fases*) i un arxiu `recommender.py` que contindrà la classe RecommenderSystem i aprofitarà els altres components tant de l'ontologia com de la lògica de les quatre fases del CBR.

3.4.1 Lògica inicial del nostre RecommenderSystem i del `main.py`

Idealment en un CBR voldriem que el nostre `main` tingués la següent forma:

```
from miniCBR import *
import pandas as pd

df_problemes = pd.read_csv('basedecasos.csv')
rs = createRS(casebase=df_problemes, maxcasesxleaf=10)

# Exemple de prova: [1,0,0,0,0,0.3,0,0,0,0,0.5,0.5,0.5]

while True:
    newp = Problem(*eval(input('Introdueix la nova instancia de Problem(): ')))
    newcas = Cas(id=(len(rs.indexacio)),
                 descripcio=rs.scaler.transform([newp.array])[0])

    distance, case, path = rs.retrieve(newcas)
    rs.reuse(case, newcas)
    rs.revise(newcas)
    rs.retain(distance, newcas, path)

    exit = eval(input('Vols seguir amb una nova recomanació? (True/False): '))
    if not exit:
        break
```

I és que reflexa l'esquema típic d'un CBR on adquireixes un nou problema i, mitjançant una base de casos extraus una solució que, després de ser revisada és presentada a l'usuari que la valorarà. Posteriorment es decideix si afegir aquest nou cas a la base de casos o no i s'espera al nou problema per entrar.

Per acompanyar aquest esquema del CBR per a Recomanació hem implementat la classe RecommenderSystem que incorpora les 4 fases super simplificades per entendre el funcionament base d'un sistema basat en casos.

```
from miniCBR.fases import *
from miniCBR.cases import *
import random

class RecommenderSystem:
    def __init__(
            self,
            maxcasesxleaf:int = 5,
            casebase:pd.DataFrame=None
        ) -> None:
        db, self.scaler = createDB(casebase)
```

```
        self.casebase = db
        self.indexacio = Indexacio(casesbase=self.casebase, casesxleaf=maxcasesxleaf)

    def retrieve(self, newcase:Cas) -> Tuple[float, Cas, List[int]]:
        return most_similar(self.indexacio, newcase)

    def reuse(self, case, newcas) -> bool:
        '''
        True if we use the returned case from retrieve. Else False

        In this minimal form we don't do any adaptation.
        '''
        newcas.solucio = case.solucio.copy()

    def revise(self, newcas):
        '''
        Assigna un valor enter aleatori com a recomanació. No dona
        l'opció a no llegir el llibre.
        '''
        newcas.avaluacio = [random.randint(1,5) for _ in range(3)]
        return newcas.avaluacio

    def retain(self, dist, newcas, path, T=0.9):
        '''
        Acull el nou cas si la distància es superior al threshold T.

        Sols mira redundància com a criteri. No té en compte l'avaluació
        de l'usuari.
        '''
        if dist > T:
            self.indexacio.add(newcas, ruta=path)
            print('Adding the new case.')
        else:
            print('New case is redundant')

def createRS(casebase:pd.DataFrame, maxcasesxleaf:int = 10):
    rs = RecommenderSystem(maxcasesxleaf=maxcasesxleaf,casebase=casebase)
    return rs
```

En aquesta versió simplificada, el nostre RecommenderSystem sols extreu **un únic cas més similar** del qual **copia la solució**, l'entrega a l'usuari que hem simulat amb un comportament **aleatori** i on després escollim si ens quedem el nou cas en base a la **distància** obtinguda al `retrieve()` entre el cas nou i el cas més similar.

La classe RecommenderSystem incial ha quedat, per tant, de la següent forma:

```
from miniCBR.fases import *
from miniCBR.cases import *
import random

class RecommenderSystem:
    def __init__(
            self,
            maxcasesxleaf:int = 5,
            casebase:pd.DataFrame=None
        ) -> None:
        db, self.scaler = createDB(casebase)
        self.casebase = db
```

```
        self.indexacio = Indexacio(casesbase=self.casebase, casesxleaf=maxcasesxleaf)

    def retrieve(self, newcase:Cas) -> Tuple[float, Cas, List[int]]:
        return most_similar(self.indexacio, newcase)

    def reuse(self, case, newcas) -> bool:
        '''
        True if we use the returned case from retrieve. Else False

        In this minimal form we don't do any adaptation.
        '''
        newcas.solucio = case.solucio.copy()

    def revise(self, newcas):
        '''
        Assigna un valor enter aleatori com a recomanació. No dona
        l'opció a no llegir el llibre.
        '''
        newcas.avaluacio = [random.randint(1,5) for _ in range(3)]
        return newcas.avaluacio

    def retain(self, dist, newcas, path, T=0.9):
        '''
        Acull el nou cas si la distància es superior al threshold T.

        Sols mira redundància com a criteri. No té en compte l'avaluació
        de l'usuari.
        '''
        if dist > T:
            self.indexacio.add(newcas, ruta=path)
            print('Adding the new case.')
        else:
            print('New case is redundant')

def createRS(casebase:pd.DataFrame, maxcasesxleaf:int = 10):
    rs = RecommenderSystem(maxcasesxleaf=maxcasesxleaf,casebase=casebase)
    return rs
```

3.4.2 Comentaris sobre l'Adaptació Nul·la

La informació i metodologia per implementar aquest enfocament es basen en el document "*(SBC-CBR Part II – COMPONENTS DELS SISTEMES CBR)*".

L'adaptació nul·la va ser seleccionada inicialment per la seva rapidesa i facilitat d'implementació i aixi tenir un model bàsic ràpid. Aquest mètode consisteix en recomanar directament els tres llibres que prèviament s'han recomanat al perfil més proper a l'usuari actual. Les raons principals per aquesta elecció van ser:

Avantatges

- Simplicitat d'Implementació: Permet una integració ràpida en el sistema.
- Rapidesa en les Recomanacions: Genera recomanacions immediates sense processos addicionals.

Limitacions de l'Adaptació Nul·la

Tot i els avantatges, l'adaptació nul·la presenta limitacions significatives:

- Dependència d'un Únic Cas Similar: Només té en compte el cas més similar.

- Risc de Recomanacions Inadequades: Si la recomanació realitzada al cas més similar ha estat poc adequada, aquest mateix error es repetirà amb l'usuari actual.

Posteriorment veurem les alternatives escollides que ens permetran dur a terme millors recomanacions, més sòlides i aprofitant tot el potencial d'un sistema basat en casos.

3.4.3 Elements de l'Ontologia

Com s'ha explicat prèviament tant a l'ontologia com a l'estructura d'un cas, el nostre **recomanador** consta d'una **indexació** (un arbre binari) dels casos amb estructura mixta, on a cada fulla conté un conjunt de **casos** que tenen com a descripció les propietats d'un **problema**. Aquest problema seria el perfil de l'usuari que busca una recomanació. En aquesta breu secció s'explica la seva forma.

BinTree Així doncs, en la primera carpeta trobem l'arxiu de l'arbre binari `bintreelinked.py` del qual es basa la indexació. L'estructura és molt simple amb els mètodes mencionats a l'ontologia i el següent constructor:

```
class BinTree:
    #------------------- nested _Node class --------------------------
    class _Node:
        __slots__ = '_element', '_left', '_right'

        def __init__(self, element, left = None, right = None):
            self._element = element
            self._left    = left
            self._right   = right

    #--------------- BinTree public methods -----------------------
    def __init__(self,v=None,left=None,right=None):
        assert (v is None and left is None and right is None) or v is not None
        if v is None:
            self._root = None   # Empty tree
        else:
            l = left._root  if (left  is not None) else None
            r = right._root if (right is not None) else None
            self._root = self._Node(v, l, r)
```

Indexació La indexació més simple del CBR inicial consta de dos mètodes: `creacio()` i `add()`. El primer ajuda a la creació de manera recursiva de la indexació de casos fins als nodes fulla. El segon, donat un cas i una ruta, afegeix aquell en la nova fulla i separa en dos si supera el límit de nodes per fulla.

Mètode constructor i `creacio()`:

```
class Indexacio(BinTree):
    def __init__(
        self,
        casesbase=None,
        left=None, right=None,
        casesxleaf:int = 10
    ) -> None:

        super().__init__(casesbase, left, right)
        self.casesbase = casesbase
        self.__length = len(self.casesbase)
```

```
        self.casesxleaf = casesxleaf
        self.creacio(casos=casesbase)

    def __len__(self):
        return self.__length

    def creacio(self,casos:List[Cas]) -> None:
        if len(casos) <= self.casesxleaf:
            # Si només hi ha un cas, retorna un arbre amb aquest cas.
            self.set_root(casos)
        else:
            # Aplicar KMeans per a dividir els casos en grups.
            kmeans = KMeans(n_clusters=2, random_state=0, n_init='auto')
            kmeans.fit([cas.descripcio for cas in casos])

            # Dividir els casos segons les etiquetes del clustering.
            cluster1 = [casos[i] for i in range(len(casos)) if kmeans.labels_[i] == 0]
            cluster2 = [casos[i] for i in range(len(casos)) if kmeans.labels_[i] == 1]

            # Crear els subarbres per a cada cluster.
            subarbre1 = Indexacio(casesbase=cluster1, casesxleaf=self.casesxleaf)
            subarbre2 = Indexacio(casesbase=cluster2, casesxleaf=self.casesxleaf)

            # Retorna un arbre amb l'objecte KMeans com a arrel i els subarbres com a fills.
            self.set_root(kmeans)
            self.set_left(subarbre1)
            self.set_right(subarbre2)
```

Cas i Problema L'estructura dels casos i problemes de l'ontologia es veuen simplificats en aquesta primera versió en aquestes dues classes:

Classe *Cas*:

```
class Cas:
    def __init__(
        self,
        id: str,
        descripcio: ndarray[Any], # El nostre problema ja preprocessat
        derivacio = None,
        historial = [],
        solucio: List[int] = [],
        avaluacio: List[int] = []

    ) -> None:
        self.id = id
        self.descripcio = descripcio
        self.derivacio = derivacio
        self.solucio = solucio # llista de 3 llibres recomanats
        self.avaluacio = avaluacio # num del 1-5
        self.historial = historial

    def __repr__(self) -> str:
        return f"Cas(id={self.id}, solucio={self.solucio}, avaluacio={self.avaluacio})"
```

Classe *Problem*:

```
class Problem:
    def __init__(
```

```
        self,
        fantasy:float, fiction:float, mystery:float, poetry: float,
        history: float, romance: float, nonfiction: float, children: float,
        youngadult: float, comics: float, mean_length: float,
        count_reads: int, avg_rating: float
    ) -> None:

        self.fantasy = fantasy
        self.fiction = fiction
        self.mystery = mystery
        self.poetry = poetry
        self.history = history
        self.romance = romance
        self.nonfiction = nonfiction
        self.children = children
        self.youngadult = youngadult
        self.comics = comics
        self.mean_length = mean_length
        self.count_reads = count_reads
        self.avg_rating =  avg_rating

        self.array = list(self.__dict__.values())

    def __repr__(self):
        return f"Problem({self.array})"
```

Entre els dos arxius, a més a més, trobem la lògica de conversió del cas i el problema a quelcom que un model KMeans pugui tractar. Això està implementat a les funcions `createDB()` i `scale_features()`, que, a més a més, ens retornarà l'escalador per a processar futurs casos nous.

4 Etapes de la recomanació

4.1 Etapa de Retrieve

En la fase de Retrieve del nostre sistema, ens centrem en seleccionar un conjunt de casos similars que serviran com a base per a la recomanació final. Aquest procés és crucial, ja que la qualitat de les recomanacions depèn significativament de l'eficàcia amb què es recuperen casos pertinents.

La nostra estructura d'indexació s'organitza mitjançant clústers, on els nodes fulla representen llistes de casos. Aquests casos són considerats similars dins del seu respectiu clúster. Per identificar el cas més similar, el sistema navega a través dels clústers fins a arribar al node fulla més pertinent.

4.1.1 Primera estrategia

Inicialment, es va considerar la possibilitat de seleccionar directament un node fulla com a conjunt de casos similars. No obstant això, aquest enfocament va presentar limitacions, com la variabilitat en el nombre de casos per fulla. Això es deu a la forma en què es divideixen les poblacions en els clústers, resultant de vegades en nodes amb un únic cas o amb un nombre insuficient de casos per a una recomanació robusta.

4.1.2 Estrategia final

Per superar aquestes limitacions, es va decidir establir un requisit fix per al nombre de casos a recuperar en cada recomanació. El procediment es manté similar al inicial, però amb una adaptació clau: si un node fulla no conté el nombre mínim de casos requerits, el sistema buscarà el node germà pertinent. Si aquest node germà no és un node fulla, es repetirà el procés de cerca començat inicialment. Aquesta estratègia assegura que sempre disposarem del nombre necessari de casos per a cada recomanació. Això s'aconsegueix mitjançant una definició prèvia que estipula que un node ha de contenir un nombre màxim de casos per ser considerat clúster, garantint així la disponibilitat dels casos requerits en el node germà.

4.2 Etapa de Reuse

Durant l'etapa de reuse cal trobar una recomanació per al nou cas, prenent en compte factors com les recomanacions prèvies a casos similars i les valoracions d'aquests llibres per part dels usuaris. El mètode ha d'identificar llibres recomanats a usuaris amb perfils semblants i analitzar l'èxit d'aquestes recomanacions. La funció del `RecommenderSystem` responsable d'aquesta tasca és:

```
def reuse(self, casos_sim, newc):
        result = confianza(casos_sim, newc)
        return result
```

Aquesta funció agafa el nou cas i una llista de casos similars, retornant un diccionari amb informació dels tres llibres a recomanar. Cada entrada del diccionari inclou les valoracions predites i una llista dels casos similars que contenen el llibre.

4.2.1 Valor de Confiança

Com a millora respecte al prototip inicial amb Adaptació Nul·la, s'ha implementat una adaptació més sofisticada anomenada **Valor de Confiança**. Inspirada en la metodologia del document *Ïmproving Case Representation and Case Base Maintenance in Recommender Agents"*, aquesta tècnica millora significativament la fase de Reuse en el nostre sistema de recomanació de llibres.

El Valor de Confiança es basa en considerar quins són els millors llibres recomanats del conjunt de casos similars. Això ho fa considerant la distància entre el cas anterior que va recomanar el llibre i

el nou cas de l'usuari, realitzant un càlcul ponderat que inclou tant les valoracions com la proximitat dels casos respecte al nou usuari.

Aquest enfocament permet al sistema adaptar-se millor a les necessitats específiques de cada usuari, aportant major flexibilitat i una anàlisi més detallada de les preferències. No es limita només a les recomanacions anteriors, sinó que amplia la perspectiva incloent un conjunt de casos similars i la seva distància relativa. Aprofundeix en les valoracions de les recomanacions, analitzant tant els casos ben valorats com els mal valorats per a perfeccionar el procés de recomanació.

Aquest mètode calcula una valoració ponderada per cada llibre, prenent en compte la qualitat de les recomanacions prèvies i la proximitat dels casos. Els resultats s'ordenen i es seleccionen els tres llibres més pertinents com a recomanacions finals, oferint una solució equilibrada que combina eficiència i simplicitat.

4.3 Etapa de Revise

L'etapa de revise en els CBRs és essencial per a avaluacions de qualitat del sistema. En aquesta fase, es rep una valoració de l'usuari que es compara amb l'avaluació proposada pel CBR. Això permet conèixer la qualitat de les respostes que genera el sistema. La funció del `RecommenderSystem` per a aquesta etapa és la següent:

```
def revise(self, cas, casos, diccion, user_eval:bool = False):
    if user_eval:
        user_evaluation(cas, self)  # Forma manual
    else:
        predict_evaluation(cas, self) # Usant predictiu
    recompte_valors_utilitzats(cas, casos, diccion)
    calcAE(cas,diccion,self)
```

4.3.1 Revisió Manual

En la revisió manual, la funció `user_evaluation` interactua amb l'usuari i recull la seva valoració. Aquesta interacció té dues versions: una a través del terminal i l'altre en la interfície gràfica.

La versió en terminal utilitza la funció `valorate` i s'efectua a través de preguntes directes sobre la qualitat dels llibres recomanats, on l'usuari assigna una puntuació de l'1 al 5.

Un exemple d'interacció amb l'usuari és:

```
Recorda que si no has comprat un llibre has d'escriure '0' estrelles
Què t'ha semblat el llibre {llibre1}, quantes estrelles li dones? (1-5):
- 1 -

Què t'ha semblat el llibre {llibre2}, quantes estrelles li dones? (1-5):
- 5 -

Què t'ha semblat el llibre {llibre3}, quantes estrelles li dones? (1-5):
- 0 -
```

La versió de la interfície gràfica obté la mateixa informació diferent però la implementació canvia. En el codi entregat, `user_evaluation` està implementat amb aquesta versió.

4.3.2 Revisió Automàtica

En cas que l'usuari no vulgui o no pugui proporcionar una valoració, es fa ús d'un model XGBoost per simular la predicció de l'usuari. Aquesta aproximació predictiva ajuda a determinar la qualitat de

la recomanació feta pel sistema. El model XGBoost calcula una predicció que simula la valoració de l'usuari, la qual es compara amb la valoració real per avaluar l'efectivitat de la recomanació.

Finalment, la funció `calcAE` calcula l'error absolut mitjà normalitzat, proporcionant una avaluació quantitativa de la qualitat de les recomanacions.

Entrenament model XGBoost

Aquest model de revisió automàtica s'ha realitzat a través de un XGBoost, entrenat amb 40000 dades. El primer que fem és eliminar aquells casos dolents ja que no aporten cap informació. Amb aquest filtratge, creem la base de casos i obtenim una llista amb tots els casos creant després un DataFrame. Així tenim un dataframe que representa diferents usuaris, però per a entrenar el model, necessitem més informació, així que hem fet un merge d'aquest dataframe amb un dataframe que representa les característiques de cada un dels llibres. Les dades finals que utilitzem per entrenar el model consisteixen en diferents files que representen un usuari, un llibre i la valoració que li ha donat l'usuari.

A partir d'aquestes dades hem seguit entrenant un `RandomForestRegressor` per a obtenir les importàncies de les variables i eliminar aquelles que no siguin importants per a la predicció filtrant per un mínim de importància de 0.01.ç

Eliminant aquestes variables hem entrenat un model de regressió amb XGBoost. Primer hem experimentat amb diferents paràmetres per saber quins són els òptims i finalment ens hem quedat amb el model amb `n_estimators = 1500` i `max_depth = 6`. En l'arxiu *model_xgb.ipynb* es pot trobar tot el procediment d'entrenament i també una prova de utilització del model a partir de una instància de cas.

4.3.3 Càlcul de Valors d'Utilitat

La funció `recompte_valors_utilitzats` realitza els càlculs necessaris per a determinar la utilitat dels casos, actualitzant els atributs pertinents. Aquest procés es duu a terme durant l'etapa de revise per eficiència, aprofitant l'oportunitat per valorar directament la utilitat de la recomanació. Així es redueix la necessitat s'haver d'entrar cada cop a l'etapa de manteniment, i s'evita haver de recórrer tot l'arbre posteriorment per a fer els càlculs pertinents.

4.4 Etapa de Retain

Finalitzada l'etapa de revise, on es qualifica l'avaluació del sistema, es disposa de tota la informació completa d'un cas. Aquest punt marca el final del procés de recomanació i s'inicia la decisió sobre si cal mantenir el cas per la seva utilitat o descartar-lo per ser redundant. Aquesta decisió es basa en la presència o no de casos molt similars dins de la biblioteca de casos existents.

La funció del `RecomendedSystem` encarregada d'aquesta etapa és:

```python
def retain(self, cas, cas_dist:List[Tuple[float, Cas]], list_path:List[int], T: int = 2):
    redundancia = redundant_cas(cas, cas_dist, T)
    if redundancia == 0:
        # Cas no redundant i s'ha indexat
        self.indexacio.add(cas,list_path)
        self.cas_nr += 1
        return 0
    elif redundancia == 1:
        # Cas redundant
        return 1
    elif redundancia == 2:
        # Cas redundant i ÚTIL
        self.cas_r_u += 1
        return 2
```

Aquesta funció classifica els casos en tres categories: útils, redundants, i rellevants. Per a saber a quina categoria pertany el cas, es fa a través de la funció `redundant_cas`, que compara els casos nous amb els més similars segons la seva distància. Depenent de la categoria a la que correspongui es prenen les mesures corresponents. Concretament, cada tipus de cas es descriu de la següent manera:

- **Cas no redundant (útil):** Aquests casos nous no són redundants i per tant s'indexaran a la base de casos del node on està situat el cas més similar. Si l'indexació d'aquest cas supera el nombre màxim de casos per fulla, es generarà un altre clúster amb dos nodes fulla més.
- **Cas redundant no útil:** Aquests casos són molt similars a un cas ja indexat i no aporten informació valuosa addicional, per tant, s'obliden.
- **Cas redundant útil (rellevant):** Tot i ser similars a un cas ja indexat, aquests casos aporten informació més valuosa, com per exemple, recomanacions de llibres que han agradat més a l'usuari. En aquests casos, es realitza un intercanvi: el cas nou s'indexa en la posició del cas redundant i el cas redundant es descarta. Aquesta metodologia permet millorar contínuament la qualitat de les recomanacions. Concretament, la funció `better` compara els casos per veure si el cas nou ofereix una millor recomanació que el cas existent, i `swap` realitza l'intercanvi dels casos en cas que el nou sigui més valuós.

4.5 Manteniment General de la Base de Casos

Per a tenir un model que sigui escalable, és necessari realitzar un procés de manteniment de la base de casos. El manteniment té dues parts, una que s'executa per a cada nou cas i un altre que s'executa cada cert temps i en cumplir uns requisits. Per a aquestes dues aproximacions necessitem entendre diferents conceptes sobre els casos que emmagatzemem o no a la base de casos: utilitat, redundància, i rellevància.

En aquestes subseccions del nostre informe, ens centrem en la identificació i gestió de casos rellevants. La informació per a la implementació d'aquests mètodes provenen dels documents "SBC-CBR Part III – Aplicacions del CBR", i "SBC-CBR Part IV – Problemes en el Desenvolupament de Sistemes CBR".

4.5.1 Tractament de casos útils, redundants, i rellevants

El tractament dels casos mai és senzill, requereix tenir una idea clara sobre aquells casos que es volen indexar o no.

Desafiaments Identificats: Un dels principals desafiaments en la gestió de la nostra base de casos és l'equilibri entre el cost espacial i temporal i el risc de biaixos. La inclusió de tots els casos sense cap filtre pot resultar en una base de casos excessivament gran, augmentant significativament tant l'espai de memòria física requerit com el temps necessari per trobar casos similars. A més, l'emmagatzematge de casos massa similars entre ells pot conduir a biaixos en les recomanacions, afectant la qualitat i la diversitat d'aquestes. Així doncs, hem decidit implementar criteris específics per determinar la rellevància de cada cas a la nostra base de dades.

Criteris per Determinar la rellevància d'un Cas: Per tant en aquesta secció ens centrem en trobar aquells casos que per nosaltres hem considerat rellevants, tenint en compte dos factors diferents:

- **Utilitat:** Un cas és considerat útil si ha contribuït significativament en alguna recomanació anterior, demostrant ser pràctic i pertinent en el context de les necessitats dels usuaris.
- **No Redundància:** Un cas es considera útil si aporta una perspectiva o informació nova que no està coberta per altres casos ja existents a la base de dades, assegurant diversitat i evitant la redundància d'informació.
- **Redundància útil:** Com s'ha explicat en l'Etapa de Retain existeixen redundants útils.

L'objectiu d'aquesta estratègia és optimitzar l'eficàcia del sistema CBR, millorant la qualitat de les recomanacions i gestionant eficientment els recursos del sistema. Això inclou no solament la selecció

acurada de casos per a emmagatzemar sinó també una reflexió constant sobre com els casos seleccionats poden influir en el comportament global del sistema de recomanació.

Aquests tres factors es tracten de forma diferent en el nostre sistema CBR. Per una banda, els casos redundants es tracten per a cada nova recomanació, decidint si afegir-la, canviar-la pel seu similar o deixar-la fora (explicat ja a la seccio de Retain). D'altra banda, la gestió dels casos útils es realitza cada cert nombre de iteracions ja que pel seu càlcul, es requereixen moltes noves recomanacions. En aquest segon cas, es regenera l'índex tenint en compte només els casos útils i aquells que encara no es pot considerar útil.

4.5.2 Procés de regeneració d'indexos

Com ja hem dit, la regeneració d'indexos és una de les parts del manteniment de la base de casos. Aquesta generació es realitza quan es cumpleixen uns certs requisits:

El mètode *maintenance* del `RecomenderSystem` només és cridarà si hi ha al menys **100 casos nous indexats**.

A més, ha de cumplir un mínim de memòria ocupada **o** temps d'obtenció dels casos més similars:

- **Mínima memòria ocupada**: el `RecomenderSystem` ha d'ocupar un mínim de 350000 Bytes
- **Mínim temps**: la funció `most_similar` ha de tardar més de 0.002 segons

Aquests valors s'han obtingut a partir de l'experimentació.

Per a realitzar la regeneració de la indexació hem utilitzat el mètode `refer_base` de la pròpia `Indexació`. Aquest mètode crida al mètode creacio, que torna a crear tota la indexació a partir d'una llista de casos filtrats únicament per als **casos útils**. Així, el manteniment crea un nou arbre que, a més d'eliminar aquells casos no útils, pot canviar la divisió dels clusters que ja inclouran la informació dels nous casos afegits.

Per a trobar aquests casos útils hem utilitzat el mètode de Utilitat Normalitzada. Aquest métode, explicat a *"SBC-CBR Part III – Aplicacions del CBR"* consisteix en tenir un recompte de la utilització del cas en recomanacions bones i dolentes:

$$UM(C) = \frac{\frac{\#UaS}{\#S} - \frac{\#UaF}{\#F} + 1}{2}$$

- **#UaS** és el nombre de vegades que es va utilitzar el cas i hi va haver un èxit, quan el cas es trobava entre els casos recuperats.
- **#S** és la quantitat total d'èxits quan el cas es trobava entre els casos recuperats.
- **#UaF** és el número de vegades que s'ha utilitzat el cas i va ser un fracàs, quan el cas es trobava entre els casos recuperats.
- **#F** és el número total de fracassos quan el cas es trobava entre els casos recuperats

Durant el procés de `revise` utilitzem la funció *recompte_valors_utilitzats* per a modificar els valors UaS, UaF, S i F del cas així després calculant i modificant la seva Utilitat_normalitzada (UM). Així doncs, a l'hora de trobar els casos útils en la regeneració de l'índex, només cal obtenir la llista de casos filtrant la UM amb un llindar. L'aproximació del mètode fa que els últims casos afegits puguin tendir a tenir una UM molt baixa i, per això, deixem de banda només aquells que no estiguin entre els últims 50 casos afegits i no superin el llindar.

5 Qualitat del funcionament

5.1 Mètriques d'avaluació

És fonamental establir criteris d'avaluació que ens permetin mesurar l'eficàcia i eficiència del sistema. En aquesta subsecció, es descriuen els conceptes clau que seran la base per a l'avaluació del nostre sistema. Es consideren dues perspectives principals: una enfocada en l'aspecte operacional, incloent l'ús de recursos com l'espai i el temps, i l'altra orientada cap a la qualitat i la validesa de les recomanacions i prediccions generades pel sistema.

5.1.1 Espai i temps

Des d'una perspectiva operacional, és crucial avaluar com el nostre sistema gestiona l'espai de memòria i el temps de procés. L'avaluació de l'espai implica determinar la quantitat de memòria necessària per emmagatzemar i gestionar eficaçment els casos, mentre que l'avaluació del temps se centra en la rapidesa amb la qual el sistema pot recuperar, adaptar i reutilitzar la informació dels casos per fer recomanacions.

Per avaluar aquestes mètriques ho fem de la mateixa manera com ho hem fet a la secció de "Manteniment General de la Base de Casos", per l'espai es calcula quanta memòria física es necessita per emmagatzemar la indexació i pel temps calculem el temps que es tarda en fer una recomanació.

5.1.2 Qualitat i Realitat de les Recomanacions

Per altra banda, la qualitat i la realitat de les recomanacions són aspectes fonamentals per avaluar l'efectivitat del sistema des del punt de vista de l'usuari final. Aquesta avaluació busca determinar fins a quin punt les recomanacions fetes pel sistema són útils, precises, i en quina mesura reflecteixen les necessitats i preferències reals dels usuaris. L'objectiu és maximitzar l'adequació de les recomanacions a les expectatives i necessitats dels usuaris, garantint així que la resposta del sistema sigui la més apropiada i personalitzada possible. Indirectament per fer la recomanació a l'etapa de reuse, estem fent una predicció de la puntuació que li posaria l'usuari un determinat llibre, si la predicció no és correcte li recomanarà mals llibres.

5.2 Conceptes

Aquesta subsecció s'endinsa en els mètodes utilitzats per avaluació del nostre sistema de recomanació de llibres basat en raonament basat en casos (CBR). El focus es posa en dues dimensions principals: la qualitat de les recomanacions i la realitat de les prediccions.

5.2.1 Avaluació de la Qualitat

La qualitat de les recomanacions s'estima a partir de l'agregació de les valoracions emeses pels usuaris per a cada recomanació de llibre. Les valoracions són puntuacions numèriques que oscil·len entre 0 i 5. Per a obtenir una mesura estandarditzada sobre 10, sumem totes les valoracions proporcionades i les dividim per la quantitat total de prediccions realitzades, multiplicada per 2. Això ens dóna una mitjana ponderada que reflecteix la satisfacció general dels usuaris amb les recomanacions rebudes, on un valor més alt indica una major qualitat percebuda. La fórmula és la següent:

1. Avaluació de la Qualitat: La qualitat de les recomanacions s'estima a partir de l'agregació de les valoracions emeses pels usuaris per a cada recomanació de llibre. Les valoracions són puntuacions numèriques que oscil·len entre 0 i 5. Per a obtenir una mesura estandarditzada sobre 10, sumem totes les valoracions proporcionades i les dividim per la quantitat total de valoracions, i tot multiplicada per 2. Això ens dóna una mitjana ponderada que reflecteix la satisfacció general dels usuaris amb les recomanacions rebudes, on un valor més alt indica una major qualitat percebuda. La fórmula és la següent:

$$\textit{Qualitat de les Recomanacions} = \frac{\sum \text{valoracions de l'usuari} \times 2}{\text{nombre de valoracions}} \tag{1}$$

Per avaluar com de realistes són les prediccions del nostre sistema utilitzem la mètrica NMAE (Normalized Mean Absolute Error), tal com es descriu en el paper "A Cluster-indexing CBR Model for Collaborative Filtering Recommendation". El NMAE mesura la desviació mitjana absoluta entre les valoracions predites pel sistema i les valoracions reals que els usuaris han donat. Aquesta mètrica s'expressa com un percentatge del rang màxim de valoracions, proporcionant una visió normalitzada de l'error. La fórmula per calcular el NMAE és:

$$NMAE = \frac{\frac{1}{m_a} \sum_{j=1}^{m_a} |P_{a,j} - V_{a,j}|}{r_{\max} - r_{\min}} \tag{2}$$

On m_a és el nombre de valoracions predites per l'usuari actiu, $P_{a,j}$ són les puntuacions predites, $V_{a,j}$ són les puntuacions reals, i $r_{\max}$ i $r_{\min}$ són les valoracions màxima i mínima de l'escala respectivament. Un NMAE més baix indica que les prediccions són més properes a les valoracions reals, i per tant, més "reals".

6 Experimentació i Resultats

A l'hora de fer la implementació, hi ha alguns llindars que no tenim el coneixement expert suficient per a poder definir els seus valors. Per això, s'ha decidit fer experimentació variant aquests threshold, per a saber quin és aquells que genera un sistema més robust i eficaç.

6.1 Casos per Fulla

La definició del nombre màxim de casos per fulla en un arbre de decisió és un element crucial durant la seva construcció. Aquest paràmetre influeix directament en l'eficiència del sistema i, per tant, requereix una atenció especial per determinar el seu valor òptim.

Per a l'ajust d'aquest valor, s'han considerat diverses mètriques com la qualitat de l'avaluació, la precisió dels resultats mitjançant la métrica de l'error mitjà absolut normalitzat (NMAE), el percentatge de casos redundants, el temps necessari per generar la base de dades, així com el temps requerit per completar les quatre fases del sistema de recomanació. D'aquestes mètriques, la que ha mostrat una correlació més significativa amb el nombre de casos per fulla ha estat el temps de generació de la indexació.

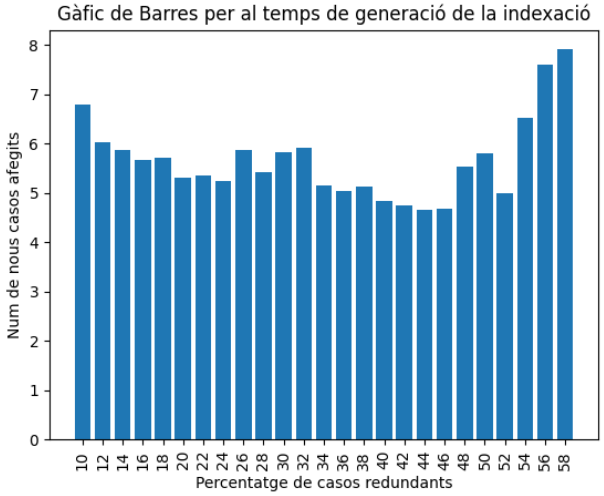

Figura 3: Evolució del temps de generació de la indexació en funció dels Casos per Fulla

Com es pot observar en el gràfic, existeix un punt en què el temps de generació de la indexació convergeix, suggerint un valor òptim per al nombre màxim de casos per fulla. Aquest punt es troba en el valor 44; per tant, es fixarà aquest com el nombre de casos màxims a contenir en cada node fulla de l'arbre.

6.2 Redundància en l'Etapa de Retain

L'etapa de retain en els sistemes basats en casos col·laboratius inclou una funció crítica que determina la redundància dels casos. Aquesta decisió es basa en gran mesura en la distància als casos més similars. Per tant, és vital realitzar un estudi detallat per identificar la distància òptima que eviti retenir casos redundants sense excloure els que poden ser útils. Per avaluar aquesta característica, s'han utilitzat mètriques com la qualitat de l'avaluació, la precisió dels resultats a través de l'error mitjà absolut normalitzat (NMAE) i el percentatge de casos redundants.

Els resultats dels experiments s'il·lustren en dos gràfics centrats en la redundància:

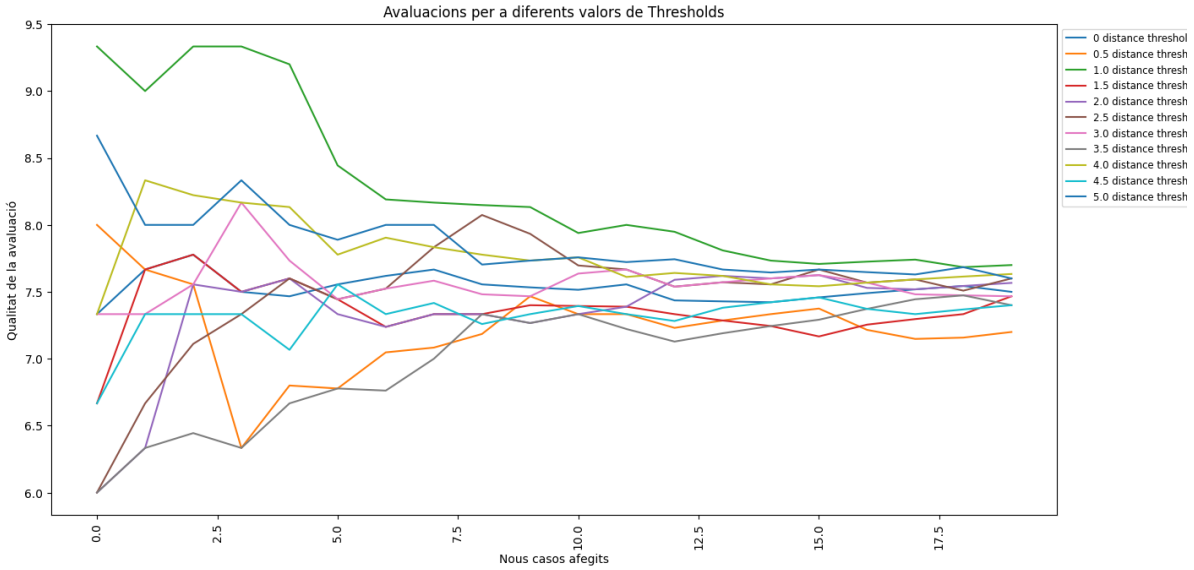

Figura 4: Avaluació del sistema per a diferents distàncies de redundància

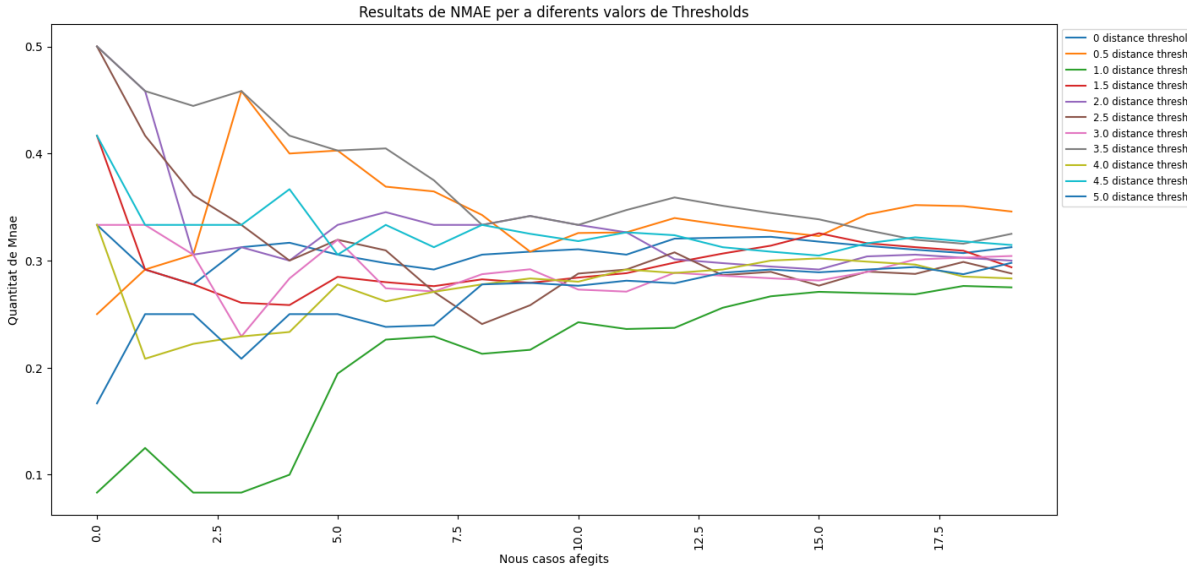

Figura 5: NMAE per a diferents distàncies de redundància

El primer gràfic, relacionat amb l'avaluació del sistema, busca assolir el valor més alt possible, indicant una millor valoració de l'usuari. D'altra banda, el segon gràfic, que representa l'error mitjà absolut normalitzat, aspira a ser el més baix possible, reflectint una menor desviació dels resultats reals. En ambdós casos, la distància que ha proporcionat els millors resultats ha estat 1.0. Així doncs, aquest serà el valor establert per a la distància de redundància en el nostre sistema de recomanació.

6.3 Evolució de l'Espai per Casos Afegits

A partir d'aquesta experimentació no buscarem el paràmetre òptim per a algun procés en concret sinó que buscarem comprendre el funcionament del nostre CBR i com escala a llarg termini.

En aquesta secció buscarem com creix l'espai ocupat per la indexació i tots els seus components a mesura que anem augmentant el nombre de casos a indexar. Mitjançant una addició de casos per blocs de 100, hem obtingut els següents resultats:

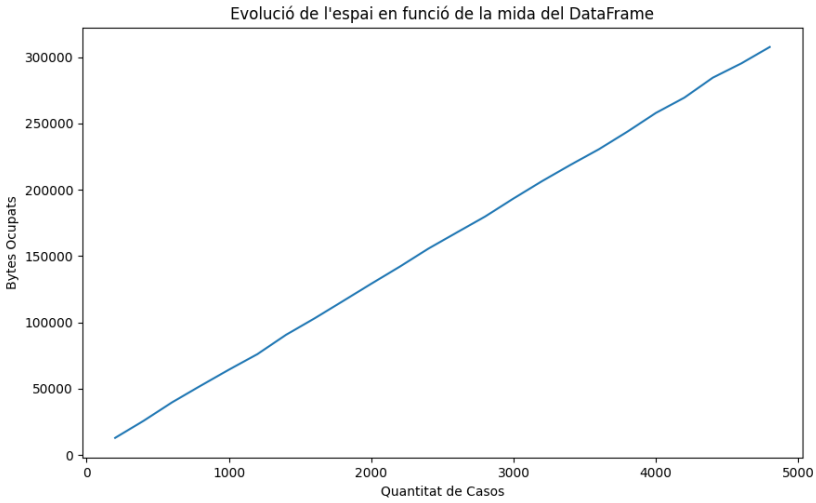

Figura 6: Evolució de l'Espai per Casos Indexats

L'eix x representa el nombre de casos indexats i l'eix y els Bytes ocupats per aquesta indexació i tots els seus components.

6.4 Explotant l'Espai d'Estats

Rati de Redundància i Avaluació de les Recomanacions

Per comprovar que el CBR actua de forma correcta primer s'ha de plantejar què és actuar de manera correcta per un CBR. Un bon CBR hauria de ser capaç de millorar-se a mesura que va rebent feedback de l'usuari.

Caldrà tenir en compte que els usuaris actuen de manera estàtica en el temps, és a dir, el model de simulació de l'usuari no canviarà. El que sí s'alterarà seran els casos de la indexació. Ara bé, en aquest apartat i el següent es mostren els resultats en base a dues maneres de canviar els casos de la indexació, de fer-los **swap**.

Les dues implementacions es troben a la funció *redundant_cas()*. La primera consta d'un *swap* simple que s'executa si es compleix la condició de què el nou cas té una millor mitjana de valoració a la del cas més similar a ell. Noti's que la mitjana de valoració prové de la simulació de l'usuari, que és estàtica i per tant té una única mitjana. Això ens durà posteriorment a una convergència de les avaluacions dels casos de la indexació.

Amb aquest tipus de *swap* hem executat 200 blocs de 100 casos aleatoris per així explotar l'espai de casos possibles i observar el comportament del CBR. Els resultats han estat els següents:

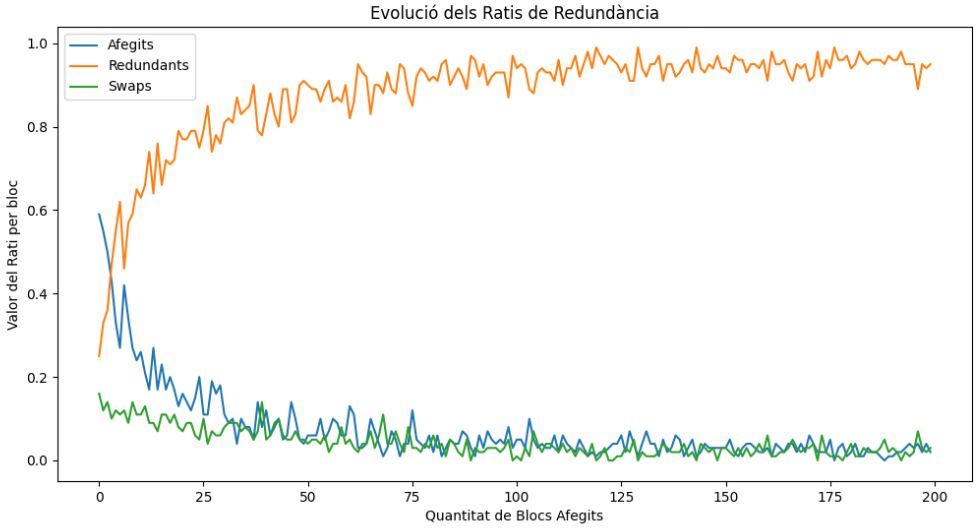

Figura 7: Rati de Redundants, Afegits i Swaps per Blocs Generats

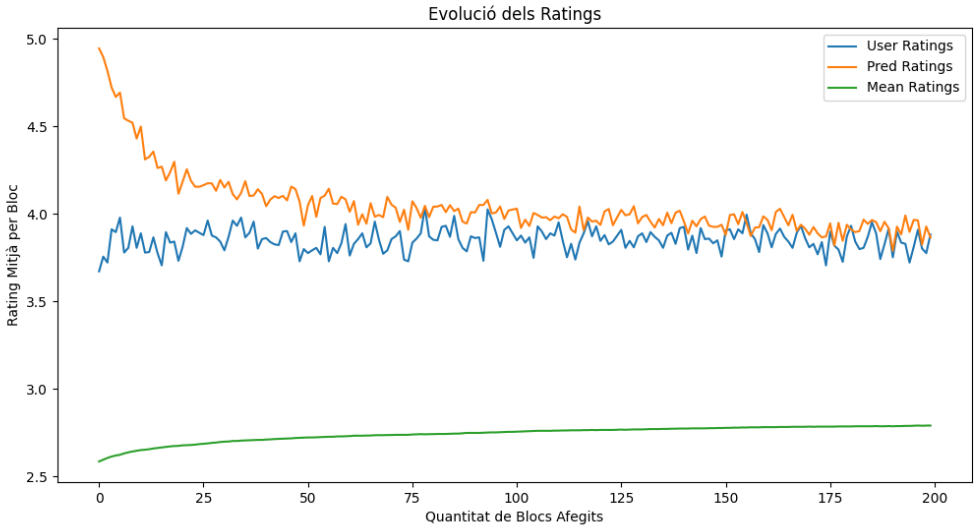

Figura 8: Mitjanes dels Ratings per Usuari, Prediccio i Casos

El primer gràfic mostra com el rati de redundants comença sent molt més baix ja que s'afegeixen molts casos nous. A mesura, però, que això succeeix, es va omplint l'espai de casos i cada cop apareix una major quantitat de redundants. El rati de swaps comença una mica més elevat però convergeix força aviat vora el 0.1

En el segon gràfic es mostra la mitjana de tres mètriques: les avaluacions d'usuari simulades, les avaluacions predites pel propi CBR i les avaluacions dels casos dins l'indexació.

Es pot apreciar com les prediccions de l'usuari es mantenen força regulars degut a què el model generador de valoracions està entrenat amb les dades inicials. Les prediccions fetes pel CBR comencen excessivament positives però decauen a mesura que els swaps van canviant casos com [0,1,5] per casos com [3,3,4]. Això provoca que els 5's disminueixin i, per tant, poc a poc es vagin aproximant a la mitjana del model generador de valoracions.

Finalment veiem com respecte a la mitjana de les avaluacions dels casos de la indexació aquesta

augmenta lleugerament degut a la addició i els swaps. Sembla que amb un límit tendint a infinit, s'establiria al voltant d'aquesta mitjana de generació de valoracions.

Ara però, es mostraran els resultats d'aplicar l'altre tipus de swap. Aquest està implementat a la funció `greedyswap()` i el què fa és aplicar una regla molt senzilla de coneixement que funciona en obtenir la millor unió possible de recomanacions donats dos casos similars. Si aquests són redundants entre sí, el que es farà serà alterar la solució del cas que ja està a la indexació tal que aquesta solució agafi el millor de les solucions dels dos casos.

S'implementa de la següent forma:

```
def greedyswap(old:Cas, new:Cas):

    na, ns, oa, os = new.avaluacio.copy(), new.solucio.copy(),
                     old.avaluacio.copy(), old.solucio.copy()

    l = []
    for idx in range(3):
        l.extend([(na[idx], ns[idx]), (oa[idx], os[idx])])
    l.sort(key=lambda x:x[0],reverse=True)

    avaluacio, solucio = [], []
    for a,s in l:
        if len(solucio) == 3:
            break
        if s not in solucio:
            avaluacio.append(a)
            solucio.append(s)

    old.avaluacio = list(avaluacio).copy()
    old.solucio = list(solucio).copy()
```

Un cop usem aquest swap, els resultats es veuen alterats de manera positiva promovent la millora del CBR:

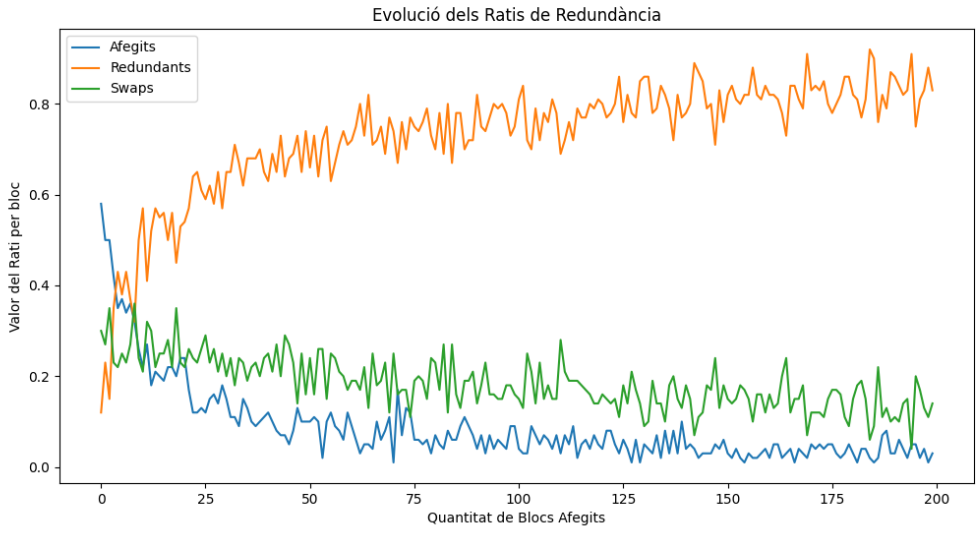

Figura 9: Rati de Redundants, Afegits i Swaps per Blocs Generats

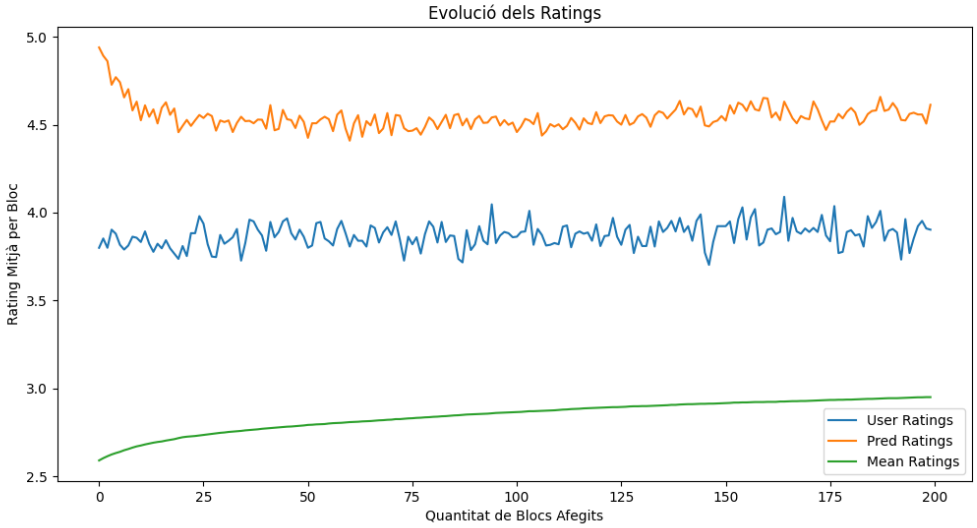

Figura 10: Mitjanes dels Ratings per Usuari, Prediccio i Casos

En el primer gràfic es nota una gran diferència en el rati de swaps ja que ha crescut notablement. Això fa que el rati de redundants no sigui tan elevat. El rati d'afegits es correspon amb el de l'apartat anterior, cosa que té sentit ja que no canviem els criteris de redundància.

En el segon veiem com, tot i que el model generador de valoracions segueixi generant una mitjana estàtica, la mitjana de valoracions dels casos de la indexació i les prediccions del CBR augmenten a mesura que anem generant casos nous, cosa que indica un **aprenentatge del sistema CBR**.

7 Interfície gràfica

Per a millorar l'experiència de l'usuari hem desenvolupat una interfície gràfica implementada amb la llibreria de python `Streamlit`. Aquesta permet a un nou usuari contestar una sèrie de preguntes per a obtenir unes recomanacions. A més, també dona la possibilitat de valorar els llibres que li hem recomanat donant un codi del 5% de descompte en cas que ho faci.

Figura 11: Elecció de les preferències

L'usuari ha d'introduir primerament tots els gèneres en ordre de preferència; primer els que més li agraden i després els que menys. A continuació ha de lliscar el cursor que representa la llargària dels llibres que prefereix, indicar una aproximació de la quantitat de llibres llegits en els darrers 10 anys i finalment indicar si és una persona que sol ser molt exigent amb els llibres que llegeix.

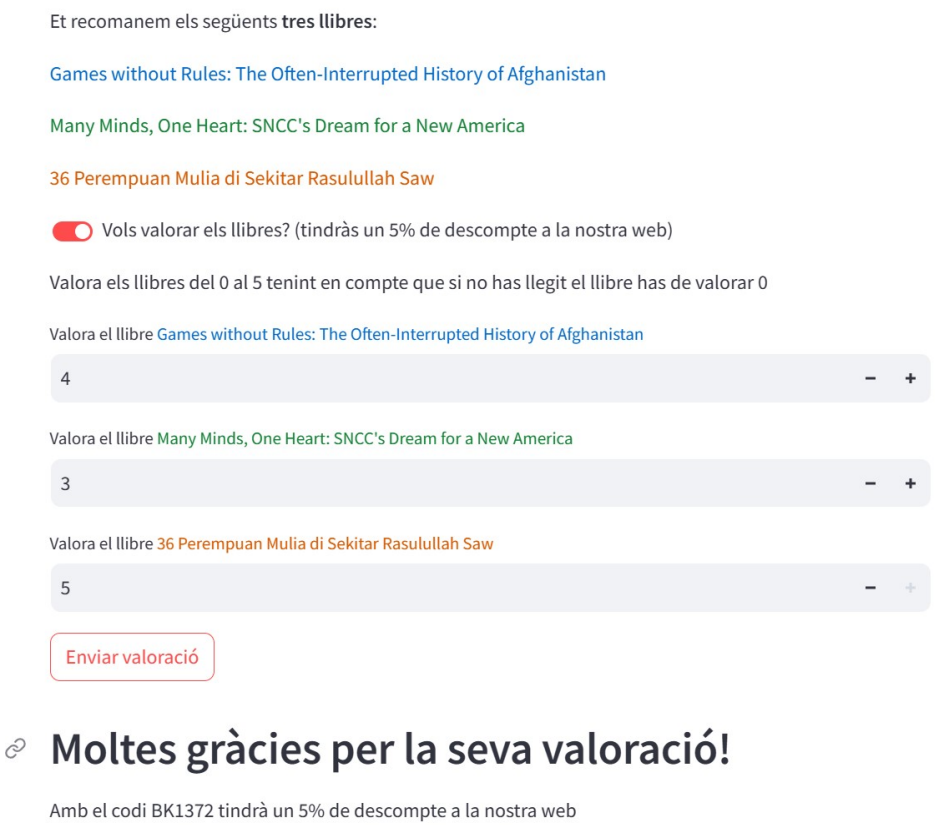

Figura 12: Elecció de les preferències

Amb la informació introduida per l'usuari, aconseguim una recomanació del nostre CBR i donem la opció que l'usuari valori aquesta recomanació. Pot valorar cada llibre del 0 al 5, indicant amb un 0 que no ha llegit el llibre finalment.

7.1 Implementació

Per a que funcioni la aplicació necessitem haver creat i guardat un `RecomenderSystem` amb el nom *recomanador.pkl*, que es pot obtenir executant el fitxer *primer_recomanador.py*. La interfície gràfica utilitza les funcions dins *us_recomender.py*. Després que l'usuari envii les seves valoracions o digui que vol finalitzar la sessió, es realitza la tasca de revise i retain, es comprova si s'ha de realitzar un manteniment i finalment s'actualitza el fixer que emmagatzema el recomanador encara que no es faci manteniment ja que s'ha pogut afegir el nou cas.

8 Futures Ampliacions

Per acabar amb la documentació, ens agradaria mostrar aquelles propostes que aplicaríem en un projecte gran a llarg termini.

Activació del Manteniment L'activació de quan es refà la indexació de la base de casos i altres processos de manteniment pot ser molt més complexa que una comprovació de thresholds. Una de les millores a implementar podria ser augmentar la complexitat d'aquesta activació tenint en compte altres paràmetres.

Altres tipus d'Indexació Per a indexar els casos de manera estructurada ens hem basat en un BinTree que emmagatzema als nodes intermitjos instàncies de KMeans de $k = 2$ i Casos a les fulles. Una opció per a continuar la recerca en aquest projecte seria comprovar l'eficàcia a nivell d'espai i temps amb altres tipus d'indexació com, per exemple, un MultiTree amb KMeans de $k > 2$. És possible que l'accés als casos més similars es duguessin a terme de manera més eficient, tot i que ja hem comprovat una gran *performance* amb l'arbre binari. Els costs variarien de ser $O(\log_2(n))$ a $O(\log_k(n))$. Hem trobat documentació que respatlla aquestes propostes: "*A Cluster-indexing CBR Model for Collaborative Filtering Recommendation*"

Canvi del Tractament de Casos Dolents En la nostra proposta de CBR hem tractat els casos dolents des del mateix mètode que els casos bons ja que permet penalitzar els casos mal puntuats i fer èmfasi en aquells amb bona puntuació. Tot i així es podrien provar altres mètodes de tractament dels casos dolents com per exemple separar-los en una indexació nova i realitzar algun procés de filtratge en la fase d'adaptació. Els casos no valorats directament no influeixen en en procés de recomanació. Potser seria útil algún mètode per tractar-los.

Prediccions de la Valoració de la Recomanació Ja hem parlat del model usat per simular el comportament de l'usuari en quant a les valoracions donades. Com a treball a futur es podrien buscar altres mètodes per dur-ho a terme. També per realitzar la predicció del recomanador segons els casos similars del nou cas. Aquestes millores podrien alhora lligar-se amb l'activació del manteniment i reentrenar el predictor.

Base de dades i Característiques del Lector Si fossim capaços d'obtenir un major nombre de *features* de la base de dades es podrien obtenir una major complexitat en els problemes i es podrien aplicar mètodes de reducció de la dimensionalitat amb semàntica per a decidir quines propietats són les més importants. Idea extreta de *A Cluster-indexing CBR Model for Collaborative Filtering Recommendation* amb la implementació del PCA.

Personalització de les Avaluacions L'última millora que proposem és ser capaços d'aplicar regles basades en el coneixement després de la valoració de l'usuari per tal de tornar a la fase d'adaptació i oferir una millor recomanació i personalitzar la recomanació depenent del perfil del lector. Aquesta idea esta extreta del paper *Evaluating recommender systems from the user's perspective: survey of the state of the art.*

9 Conclusions

Les conclusions que es poden extreure d'aquest treball han estat les que es mencionen a continuació:

Evaluació del Model La simulació del nostre model d'avaluació automàtica ha demostrat ser eficaç, rebent una valoració mitjana favorable dels clients. Aquest resultat reafirma la validesa del model en un context pràctic tot i que, com hem vist a l'experimentació, un model dinàmic que es reentrenés amb els nous casos podria aportar una major robustesa.

Eficiència i Escalabilitat Encara que el nostre model gestiona una extensa col·lecció de casos, destaca per la seva lleugeresa en termes d'ocupació d'espai. A més, la seva capacitat per recomanar llibres de manera gairebé instantània evita temps d'espera, destacant la seva escalabilitat i eficiència operativa.

Manteniment de Casos Les etapes implementades per a mantenir només els casos rellevants s'han demostrat efectives. Això indica que el model pot aprendre i evolucionar a mesura que interactua amb diferents usuaris i situacions.

Flexibilitat i Tolerància a Dades Imperfectes En comparació amb els sistemes basats en regles, el nostre sistema CBR mostra una major flexibilitat. Pot adaptar-se a situacions imprevistes i manejar problemes més complexos. La seva tolerància a dades incompletes o amb soroll és notable, un avantatge clau sobre altres enfocaments.

Complementarietat amb Altres Models El CBR pot ser integrat amb models més avançats de machine learning o deep learning, superant així les limitacions inherents al CBR i enriquint les seves capacitats de resolució de problemes.

Reusabilitat i Adaptabilitat El model col·laboratiu implementat, que no considera les característiques específiques dels llibres, demostra ser reutilitzable per a altres aplicacions. Canviant les propietats del cas i ajustant els llindars necessaris, es pot adaptar a diferents contextos.

Eficiència de la Clusterització És destacable com l'aplicació de tècniques de clusterització i l'anàlisi de casos similars pot resultar en un model eficaç, fins i tot sense un coneixement profund del domini.

Limitacions en les Recomanacions Una limitació a considerar és que el sistema només pot recomanar llibres que ja han estat recomanats anteriorment. A més, la necessitat de valoracions dels usuaris, que en molts casos és difícil o impossible d'aconseguir.

Inconsistències i Valoracions Pot aparèixer una incongruència, com en el cas de dos perfils idèntics rebent recomanacions diferents. Això ressalta una limitació dels CBRs a l'hora de tractar-los, un aspecte a considerar per a futures millores.