Documentació de la primera entrega

Projectes de Programació

Pol Casacuberta Gil
Edgar Moreno Martínez
Maria Prat Colomer
Pablo Vega Gallego

Q1 2021 - 2022
FIB - UPC

Índex

Introducció

En aquest document justifiquem l'implementació de les classes i descrivim les estructures de dades i algorismes utilitzats per implementar les funcionalitats principals de la primera entrega de l'assignatura de Projectes de Programació.

Relació de les classes implementades per cada membre de l'equip

Pol Casacuberta Gil

- ConjuntUsuaris
- ConjuntValoracions
- Programa
- Sessio
- SessioIniciada
- SessioNoIniciada
- Usuari
- Valoracio

Edgar Moreno Martínez

- package jocs_de_prova
- Pair
- recomanador.metode_recomanador.ConjuntPunts
- recomanador.metode_recomanador.KMeans
- recomanador.metode_recomanador.KNN
- recomanador.metode_recomanador.MetodeRecomanador
- recomanador.metode_recomanador.MetodeRecomanadorCollaborative
- recomanador.metode_recomanador.MetodeRecomanadorContentBased
- recomanador.metode_recomanador.Punt
- recomanador.metode_recomanador.SlopeOne
- Main

Maria Prat Colomer

- libs/consola
- ConjuntIdentificat
- ElementIdentificat
- Id
- Item
- TipusItem
- recomanador.filtre.Filtre
- recomanador.filtre.FiltreExclusiu
- recomanador.filtre.FiltreInclusiu
- recomanador.Recomanador
- recomanador.RecomanadorCollaborative
- recomanador.RecomanadorContentBased
- atributs.TipusAtribut
- atributs.distancia.Distancia
- atributs.distancia.DistanciaDiferenciaDeConjunts
- atributs.distancia.DistanciaDiscreta
- atributs.distancia.DistanciaEuclidiana
- atributs.distancia.DistanciaLevenshtein

- atributs.distancia.DistanciaZero
- atributs.valors.ValorAtribut
- atributs.valors.ValorBoolea
- atributs.valors.ValorCategoric
- atributs.valors.ValorConjunt
- atributs.valors.ValorConjuntBoolea
- atributs.valors.ValorConjuntCategoric
- atributs.valors.ValorConjuntNumeric
- atributs.valors.ValorConjuntTextual
- atributs.valors.ValorNumeric
- atributs.valors.ValorTextual

Pablo Vega Gallego

- ConjuntItems
- Contenidor
- EscriptorDeFitxers
- LectorDeFitxers
- recomanador.ConjuntRecomanacions
- recomanador.Recomanacio
- csv.EscriptorDeCSV
- csv.LectorDeCSV
- csv.TaulaCSV

A més a més, per facilitar la implementació dels tests i els drivers, hem creat dues classes: UtilitatsDeLectura i UtilitatsDEscriptura. En aquestes classes hi hem treballat tots els membres de l'equip.

Cadascú ha implementat els tests de les classes que li corresponen.

Justificació i descripció de cada classe

Pol Casacuberta Gil

- **ConjuntUsuaris**

Classe que representa un conjunt d'usuaris. ConjuntUsuari hereta de la classe ConjuntIdentificador.

- **ConjuntValoracions**

Classe que representa un conjunt de valoracions. ConjuntValoracio hereta de la classe ConjuntIdentificador.
Com en el cas de tots els conjunts, aprofitarem les característiques del TreeMap per a mantenir el conjunt ordenat per un Id i en aquest cas relacionat amb unes valoracions.

- **Programa**

Classe que representa l'estat del programa. Conté l'estat de la sessió implementat de la forma State Pattern per a saber si un usuari ha iniciat sessió o no. Conté un atribut instancia_unica que representa l'única instància de Programa que podem crear, seguint el patró de singleton. Finalment, conté un conjuntUsuaris que conté tots els usuaris del programa en aquell moment.

- **Sessio**

Representa l'estat del programa en què s'ha iniciat sessió o bé en el que no, d'aquesta classe abstracta en poden esdevenir dos estats, SessioIniciada i SessioNoIniciada.

- **SessioIniciada**

Representa l'estat del programa en què s'ha iniciat sessió.

- **SessioNoIniciada**

Representa l'estat del programa en què no s'ha iniciat sessió.

- **Usuari**

Classe que representa un usuari. Per a representar el nom i la contrasenya d'aquest hem utilitzat strings. Per a identificar de forma única als usuaris, ja que dos usuaris poden tenir noms iguals, fem servir una classe Id que conté un int que serà diferent de tots els altres usuaris, i un booleà que ens indica si l'usuari s'ha esborrat el compte o no. Finalment per a emmagatzemar els ítems valorats per l'usuari hem usat un map per a poder accedir amb un ítem, a la valoració que l'usuari ha fet a aquest ítem.

- **Valoracio**

Classe que representa una valoració. Conté el valor d'una valoració en un double i conté una instància d'Usuari i d'Ítem que representen l'usuari que ha valorat l'ítem amb aquell valor.

Edgar Moreno Martínez

- **package jocs_de_prova**

Conté un conjunt de jocs de prova per provar les funcionalitats de les classes.

- **Pair**

Classe bàsica que representa la unió de dos objectes de tipus qualssevol. S'ordena pel primer element i en cas d'empat pel segon. Ambdues classes han de ser *Comparable*.

- **recomanador.metode_recomanador.ConjuntPunts**

Aquesta classe representa un conjunt de punts, utilitzant com a base un map de int a punt per tenir els punts identificats en tot moment. Manté en tot moment calculat el baricentre dels punts del conjunt, fent-ho adient per utilitzar amb l'algorisme de k-means. També permet obtenir un identificador que no estigui utilitzat de forma eficient.

- **recomanador.metode_recomanador.KMeans**

Classe que permet processar un conjunt de punts per agrupar-los en clusters. L'únic destacable de l'algorisme és que assigna els clústers inicials als k primers punts tot i que es podrien explorar altres opcions.
L'única funció pública retorna la partició en k conjunts dels punts amb els quals s'ha construït l'objecte resultant de l'algorisme.
Hi ha un seguit de funcions i atributs privats per tal de modulitzar l'algorisme.

- **recomanador.metode_recomanador.KNN**

Classe que permet aplicar l'algoritme de k-Nearest Neighbours.
Es construeix amb el conjunt d'ítems desitjat i després es poden consultar els k items més propers a un ítem donat. Simplement, s'itera per tots els ítems, es calcula la distància a cadascun (mitjançant una funció de la classe ítem) i es retornen els més propers amb ajuda d'una cua de prioritat.

- **recomanador.metode_recomanador.MetodeRecomanador**

Classe abstracta que representa un Mètode Recomanador. Guarda un conjunt d'usuaris, ítems i valoracions que serveixen com a base per fer les recomanacions.
Permet obtenir recomanacions per un usuari concret, es poden especificar les valoracions que s'han d'usar i sobre quins ítems ha de ser la recomanació. Si no s'especifiquen s'utilitzaran els conjunts guardats.

- **recomanador.metode_recomanador.MetodeRecomanadorCollaborative**

Classe que exten MetodeRecomanador a un mètode basat en Collaborative Filtering.
Té un únic atribut que indica quants clústers s'han de crear al pas on s'usa l'algorisme k-means.

Al inicial precalcula les particions. Per això converteix els usuaris a punts a R^n en base a les valoracions que han donat al conjunt d'ítems sobre el que estem treballant. Aquests punts es processen fent servir l'algorisme de K-Means per obtenir clusters d'usuaris.
L'única funció pública genera una recomanació en rebre un usuari, valoracions d'aquest usuari, un conjunt d'ítems sobre els que recomanar i el nombre de recomanacions a generar.
Per això primer troba a quin clúster està l'usuari demanat i, per tant, els usuaris amb el que seguirà el procés.
Amb aquests usuaris es crea una matriu amb les valoracions d'aquests usuaris als elements donats. Seguidament, aquesta matriu es processa amb l'algorisme SlopeOne per obtenir les valoracions inferides per cada ítem per l'usuari a qui recomanar.
Finalment, amb una cua de prioritat s'obtenen els ítems amb millor valoració que es retornen com a recomanació.

- **recomanador.metode_recomanador.MetodeRecomanadorContentBased**

Classe que exten MetodeRecomanador a un mètode basat en filtre per contingut.
Té un únic atribut que indica quin és el llindar mínim per tal que una valoració sigui considerada.
L'única funció genera una recomanació en rebre un usuari, valoracions d'aquest usuari, un conjunt d'ítems sobre els que recomanar i el nombre de recomanacions a generar.
L'algorisme és molt simple. Iterem per tots els ítems valorats per l'usuari i de cadascun busquem els més similars. Portem un comptador de quants cops ha sortit cada ítem ponderat per la valoració de l'ítem del qual és veí cada cop. Al final utilitzem una cua de prioritat per obtenir els ítems amb millor puntuació i retornem la recomanació.

- **recomanador.metode_recomanador.Punt**

Una classe senzilla que exten el concepte de ArrayList<Double> per poder representar punts de R^n i tenir operacions senzilles (suma, resta, multiplicació per escalar, distància i norma).

- **recomanador.metode_recomanador.SlopeOne**

Classe que processa un conjunt de valoracions per tal d'oferir prediccions sobre possibles valoracions seguint l'algorisme de SlopeOne.
S'ha de construir amb una array bidimensional de doubles que representa les valoracions. Es demana en aquest format per no dependre de la implementació de les classes Usuari, Ítem i Valoració.
En construir-se precalcula les desviacions entre ítems per fer més eficient la resta del procés.
Després es poden demanar totes les prediccions o una predicció per un ítem i usuari específic.

- **Main**

El Main ens permet carregar conjunts de dades i obtenir recomanacions per un usuari i avaluar les recomanacions que s'obtenen.

Maria Prat Colomer

- **libs/consola**

Aquesta classe conté funcions estàtiques que fan més fàcil la lectura de valors des de la consola. Permet imprimir missatges de consulta, missatges d'error i comprovar que els valors llegits estan en un determinat rang. És especialment útil a l'hora d'implementar els drivers.

- **ConjuntIdentificat**

Un conjunt identificat és un conjunt tal que els seus elements es poden identificar amb un Id. El conjunt identificat ens permet abstraure la idea de conjunts d'elements identificats i poder implementar així conjunts d'Items o d'Usuaris. Els conjunts identificats s'implementen com a TreeMaps ja que volem que les operacions d'inserció i de consulta siguin ràpides i, a més, en algunes ocasions hem d'iterar sobre els seus elements.

Pel que fa a les valoracions, no es poden implementar com a conjunts identificats ja que no tenen un identificador per si mateixes.

- **ElementIdentificat**

Un element identificat és una abstracció d'Item i Usuari, que són dos elements que es poden identificar amb un Id. És una interfície necessària per definir els conjunts identificats.

- **Id**

Un Id és un identificador. Té dos atributs: un valor enter i un booleà. El valor és el valor de l'identificador i és únic per cada element. És a dir, no hi ha dos elements identificats de la mateixa classe amb el mateix valor enter. Utilitzem valors enters ja que és més eficient que utilitzar Strings i el rang és suficient pels conjunts de dades que tractarem. El booleà ens diu si l'element està actiu o no. Això ens permet conservar les dades dels elements tot i que el client els esborri i fer-les servir pels algorismes de recomanació. És a dir, si un usuari es dona de baixa, en comptes d'esborrar-lo de la base de dades, fem el seu Id inactiu. Aleshores, el client no el veurà però les seves dades rellevants pels algorismes de recomanació (valoracions,...) no s'hauran perdut.

- **Item**

Un Item té un identificador i és d'un determinat tipus i té un conjunt d'atributs amb nom i ha rebut un conjunt de valoracions. Els atributs i les valoracions són TreeMaps per poder iterar i fer consultes eficientment. Podem obtenir la distància entre dos ítems com la suma de les distàncies normalitzades de tots els seus atributs. Això ens permet donar la mateixa importància a tots els atributs independenment del rang de valors que puguin prendre les seves distàncies.

- **TipusItem**

Un ítem és d'un determinat tipus. Representem el tipus d'un ítem amb un nom i un conjunt de tipus d'atribut. Podem deduïr el tipus d'un ítem a partir de la representació d'un Item com a conjunt de Strings intentant obtenir el valor de cada String. Si tenim un conjunt d'ítems, podem obtenir un tipus d'ítem que ens permeti representar tots els ítems (això és cert

perquè, en el pitjor cas, podem considerar tots els atributs com a atributs categòrics o textuals).

- **recomanador.filtre.Filtre**

Un filtre ens permet obtenir recomanacions considerant només un subconjunt dels atributs d'un determinat tipus d'Item. Representem un filtre com el conjunt dels noms dels atributs.

- **recomanador.filtre.FiltreExclusiu**

Un filtre és exclusiu si volem incloure a la recomanació tots els atributs del tipus d'ítem excepte els indicats pel filtre.

- **recomanador.filtre.FiltreInclusiu**

Un filtre és inclusiu si volem incloure a la recomanació només els atributs del tipus d'ítem indicats pel filtre.

- **recomanador.Recomanador**

La classe recomanador ens permetrà aplicar el filtre al conjunt d'ítems abans d'aplicar el mètode recomanador. Els filtres són una funcionalitat extra que hem volgut implementar. L'estructura bàsica d'aquesta classe està implementada però la lògica no està completa i ho farem properament. També hi haurà un recomanador híbrid.

- **recomanador.RecomanadorCollaborative**

Representarà un recomanador amb el mètode col·laboratiu.

- **recomanador.RecomanadorContentBased**

Representarà un recomanador amb el mètode basat en el contingut.

- **atributs.TipusAtribut**

Cada atribut pot ser d'un tipus diferent. Un tipus s'identifica per quin tipus de valor tenen els seus elements i per quina distància s'aplica per trobar la distància entre dos atributs del mateix tipus. Per guardar el tipus del valor, creem una instància del ValorAtribut corresponent amb valor nul.

- **atributs.distancia.Distancia**

Una distància representa la funció de distància que s'aplica entre dos valors d'atributs. Com que volem les distàncies normalitzades, també guarden els paràmetres necessaris per obtenir els factors de normalització.

- **atributs.distancia.DistanciaDiferenciaDeConjunts**

Representa la distància entre dos conjunts. El factor de normalització és el doble de la norma màxima, ja que donats conjunts amb mides inferiors o iguals a un valor, el valor màxim de la distància entre qualsevol parella de conjunts és el doble de la norma màxima

(que té lloc quan els dos conjunts són totalment disjunts i cada un d'ells té la mida de la norma màxima).

- **atributs.distancia.DistanciaDiscreta**

Representa la distància discreta entre dos elements simples. Aquesta distància ja pren valors entre 0.0 i 1.0.

- **atributs.distancia.DistanciaEuclidiana**

Representa la distància euclidiana entre dos elements numèrics. El factor de normalització és el doble de la norma màxima, ja que donats valors en la bola de R^n centrada en el zero de radi la norma màxima, la distànica entre qualsevol parella de punts és, com a màxim, el diàmetre de la bola.

- **atributs.distancia.DistanciaLevenshtein**

Representa la distància de Levenshtein entre dues Strings. El factor de normalització és el doble de la norma màxima, ja que donades dues Strings de normes inferiors o iguals a un valor N, per passar de l'una a l'altra com a màxim necessitem esborrar N caràcters i afegir-ne uns altres N.

- **atributs.distancia.DistanciaZero**

Representa la distància zero. Aquesta distància sempre retorna zero. Serveix per poder ignorar certs atributs quan calculem les distàncies perquè són irrellevants o perquè calcular la distància entre una parella és massa costós. Per exemple, si tenim un atribut que és un text, pot ser que calcular la distància de Levenshtein sigui massa lent i volguem ignorar-lo.

- **atributs.valors.ValorAtribut**

El valor d'un atribut pot ser d'un determinat tipus. ValorAtribut és la classe abstracta que representa el tipus d'un valor d'un atribut. Pot ser simple (booleà, categòric, numèric o textual) o bé pot guardar un conjunt de qualsevol dels tipus simples. Creiem que amb aquests quatre tipus podem incloure tots els atributs rellevants. La distinció entre categòric i textual és conceptualment clara, encara que ambdós es guardin com a Strings.

- **atributs.valors.ValorBoolea**

Guarda un booleà.

- **atributs.valors.ValorCategoric**

Guarda una categoria.

- **atributs.valors.ValorConjunt**

És un ValorAtribut que guarda un conjunt de ValorAtributs. És una classe abstracta.

- **atributs.valors.ValorConjuntBoolea**

Guarda un conjunt de ValorsBooleans.

- **atributs.valors.ValorConjuntCategoric**

Guarda un conjunt de ValorsCategorics.

- **atributs.valors.ValorConjuntNumeric**

Guarda un conjunt de ValorsNumerics.

- **atributs.valors.ValorConjuntTextual**

Guarda un conjunt de ValorsTextuals.

- **atributs.valors.ValorNumeric**

Guarda un valor numèric.

- **atributs.valors.ValorTextual**

Guarda un valor textual.

Pablo Vega Gallego

- **ConjuntItems**

Conjunt ítems crea un conjunt d'ítems del mateix TipusItem, a partir de, o bé una TaulaCSV que contindrà les dades trobades a un fitxer CSV o altrament directament amb un TipusItem ja definit i un TreeMap<Id, Item> on enllacem l'Id del ítem amb l'ítem en concret.

Hereta de la classe ConjuntIdentificat<Item>, per a tenir una estructura semblant a tots els conjunts.

- **Contenidor**

Classe abstracta que representa un contenidor de dades del fitxer desitjat a llegir.
D'aquesta manera reduïm la quantitat de relacions que pot haver-hi si decidim llegir o escriure altre fitxer que no fos del tipus CSV en el cas de la primera entrega.

- **EscriptorDeFitxers**

Clase abstracta que escriu un contenidor a un fitxer.

- **LectorDeFitxers**

Classe abstracta que representa una funció genèrica que emmagatzema les dades a un contenidor.

- **recomanador.ConjuntRecomanacions**

Classe que permet representar un conjunt de recomanacions.
Es construeix amb el conjunt de recomanacions desitjat sempre mantenint l'ordre de la llista pel valor de Recomanacio.seguretat. És a dir, la Recomanacio que tingui una seguretat més gran, serà la que trobem en primera posició.

- **recomanador.Recomanacio**

Classe que permet representar la seguretat d'una recomanació d'un ítem.
Es construeix la variable recomanació a partir d'un Id d'ítem i un double que representa la seguretat d'aquella recomanació.

- **csv.EscriptorDeCSV**

Donada una TaulaCSV, escriu el contingut de la taula a un fitxer amb el format CSV.
És una classe que només implementa funcions per escriure CSV a partir d'una TaulaCSV i la funció abstracta de lectorDeFitxers.

- **csv.LectorDeCSV**

Donat un fitxer que tingui el format .csv, llegeix el contingut del fitxer i el transforma en TaulaCSV.
Classe específica que implementa la funció abstracta de LectorDeFitxers i la de lectorCSV.

- **csv.TaulaCSV**

Representa el contenidor de dades per emmagatzemar el contingut d'un fitxer CSV.
Classe que implementa les funcionalitats per introduir les dades del CSV a una taula i manté la integritat d'aquesta perquè no hi puguin haver incongruències a l'hora de llegir els fitxers.

Justificació de les estructures de dades utilitzades

ConjuntIdentificat, ConjuntUsuaris i ConjuntItems

En aquest cas hem fet servir un TreeMap<Id, ElementIdentificat> per tal de representar els conjunts identificats. D'aquest manera podem tenir ordenat el conjunt segons els identificadors dels elements. Això ens permet tenir un cost de O(n log n) per crear el conjunt amb *n* ElementIdentificats en l'ordre desitjat.

Sobre el TreeMap haurem de fer, principalment, operacions de consulta per obtenir elements en funció del seu identificador i també haurem d'iterar per tots els elements per poder processar els conjunts. Utilitzem un TreeMap per poder fer consultes en O(log n) i per poder iterar sobre *n* elements amb un cost lineal O(n). Si utilitzessim un HashMap, iterar sobre *n* elements tindria un cost de O(n log n).

ConjuntValoracions

Per a representar el conjunt de valoracions farem servir un TreeMap<Pair<Usuari,Item>, Item>. Ens proporcionarà les mateixes característiques que a l'apartat anterior però en aquest cas, ordenarem segons el Pair.

Programa

En aquesta classe podem trobar una atribut ConjuntDeTipusItem que ha sigut implementat amb un HashSet<TipusItem, TipusItem> per tal d'aprofitar les propietats de lectura de cost constant O(1), ja que gran part de les operacions que realitzarem sobre aquest atribut serà d'inserció, consulta i esborrat de TipusItem.

Usuari

La classe Usuari té un atribut anomenat 'valoracions' que està implementat amb un TreeMap<Item, Valoracio>. Sobre aquest atribut, a les funcions definides a la classe només realitzarem operacions de inserció, consulta i esborrat de valoracions en O(log n).

Valoració

La part més complexa de la classe Valoració és el constructor. Com que Valoració és una classe associativa entre Item i Usuari, hem decidit que, quan construïm una valoració, afegim l'objecte creat als contenidors de Valoracions d'Item i d'Usuari. Com que els dos s'implementen com a TreeMaps, la complexitat és de O(log n) on n és el nombre de valoracions.

Item

Els getters de la classe Item fan còpies profundes dels atributs. Per aquest motiu, les complexitats depenen del nombre d'elements de cada conjunt i de la complexitat de fer una còpia profunda de cada classe. Per això, fer la còpia d'un Item o utilitzar un getter d'un atribut té un cost temporal molt elevat.

La funció obtenirDistancia() té un cost temporal, en cas pitjor, de $O(NW^2)$ on N és el nombre d'atributs dels dos ítems (que ha de ser el mateix) i W és una fita de les longituds de les strings dels atributs textuals o categòrics. El cas pitjor té lloc quan tots els atributs son textuals o categòrics.

Com que els atributs i valoracions es guarden com a TreeMap, els operacions d'inserció, consulta i esborrat tenen una complexitat d'O(log n). Utilitzem TreeMaps perquè hem d'iterar sovint sobre tots els elements i, per tant, això ho fa preferible a utilitzar HashMaps.

El constructor més rellevant d'Item és el que utilitza ArrayList<String> dels noms dels atributs i dels valors, codificats com Strings. Utilitzant els mètodes privats que hem implementat, ho podem fer en O(K(n log n)) on K és una fita del nombre d'elements dels conjunts i assumint que podem fer el *parse* dels valors numèrics i booleans en temps constant (aquesta assumpció és raonable ja que són valors amb precisió limitada).

TipusItem

Guarda un map que relaciona el nom de cada atribut amb el seu tipus. Ho emmagatzamem com a TreeMap, així que les operacions de consulta, esborrat i inserció són O(log n).

Tal i com passa amb la classe Item, els seus getters fan còpia profunda, així que són mètodes amb complexitat temporal elevada que depèn del nombre total d'elements i elements dels seus atributs.

El mètode més complex d'aquesta classe és el constructor, que dedueix el TipusItem a partir de n Items candidats codificats com a Strings. El mètode privat trobaTipusAtribut() té complexitat constant ja que és una cadena d'if-else constants. El mètode privat dedueixTipusAtribut() té cost O(w) on w és la longitud de la String que rep com a argument, ja que intenta fer el *parse* d'aquesta String i en retorna el TipusAtribut per defecte en funció del resultat del *parse*. Si n és el nombre de candidats, A és el nombre d'atributs i W és una fita en la longitud dels atributs codificats com Strings, la complexitat del constructor és O(n W A log A).

Distancia

Podem saber si una distància és compatible amb un ValorAtribut determinat utilitzant la funció admet() que ho determina en temps constant.

La funció obtenir ens permet calcular la distància entre dos ValorAtributs del mateix tipus. La complexitat d'aquest mètode depèn de la subclasse de distància. En la taula següent especifiquem les complexitats temporals i espacials de la funció obtenir() de cada distància.

Per la distància de Levenshtein hem implementat una versió de l'algorisme que és eficient espacialment sense penalitzar la complexitat temporal.

Distància	Complexitat temporal	Complexitat espacial
Diferència de conjunts	O(n + m) on n i m són les	O(n + m) on n i m són les

	mides dels conjunts	mides dels conjunts
Discreta	O(1)	O(1)
Euclidiana	O(n) on n és el nombre d'elements del conjunt numèric	O(1)
Levenshtein	O(nm) on n i m són les mides de les strings	O(2n) on n és la mida de la string més curta
Zero	O(1)	O(1)

Conjunt Recomanacions

Representació d'aquest conjunt a partir d'un ArrayList<Recomanacio>. Acostumem a generar les valoracions d'un cop fent servir la constructora, fet que ens ha motivat a fer servir un tipus de dades senzill. Un cop inicialitzada farem servir el sort de java, que ordenarà el contingut segons la Recomanacio(de més gran a més petit).

ValorAtribut

A destacar de les classes que hereden de ValorAtribut són les funcions que obtenen ValorConjunt a partir d'una String que codifica booleans i nombres. Per fer-ho, divideixen l'String pels símbols ';' i intenten fer el *parse* de cadascuna de les parts resultants. La complexitat és lineal en la longitud de la String.

TaulaCSV

Tenim tres contenidor:

- Un que pasa de un índex d'atribut a un atribut. Farem servir un ArrayList<String> que fent el .get(i) d'aquesta estructura ens retorna el nom de l'atribut.
- Un que ens converteix el nom de l'atribut cap a l'índex de l'atribut. HashMap<String, Int> què per a les consultes ens donarà la possibilitat de realitzar-les a temps constant. O(1)
- En últim lloc, trobem la llistaAtributs com ArrayList<String> i les dades en general com ArrayList<ArrayList<String>>. Cada fila correspondrà a un ítem i cada columna d'aquell ítem al valor de l'atribut. No tenim cap necessitat de tenir cap estructura complicada ja que només farem consultes d'accés aleatori fent servir l'índex de la consulta desitjada.

Les operacions de TaulaCSV son de consulta a partir dels atibuts del contingut de la taula, o d'afegir objectes a aquesta TaulaCSV.

Punt

És una classe que fa extend de la classe ArrayList<Double>. Cada valor de l'ArrayList representa el valor del punt en aquella dimensió (cada posició representa la dimensió en el que el punt és representat).

ConjuntPunts

L'objectiu d'aquesta classe és tenir un conjunt de punts identificables. Això ens guia a extendre un Map de int a Punt on el primer element servirà per identificar el segon. Volem poder fer accessos aleatòris i iterar-ho, per tant un TreeMap ens dona la implementació més eficient, amb els costos esperables.

KMeans

La estructura que hem de saber mantenir per poder fer el algorisme són conjunts de punts. La implementació de la classe ConjuntPunts és ideal ja que podem fer insercions i eliminacions en temps logaritmic de punts identificables.

L'algorisme consta de un nombre de iteracions que depen dels punts inicials, sobre el qual no tenim control. Abans d'això hem de fer un clústers inicials. Si això ho fem assignant conjunts als k primers punts i llavors buscant el conjunt més proper a la resta el temps serà O(nkd), com justifiquem al següent paràgraf.

A cada iteració haurem de trobar el baricentre més propers a cada punt. Si guardem els baricentres al principi de cada iteració ho podem fer en temps O(nk) on n és el nombre de punts i k el nombre de clusters. Podriem tenir en compte que les comparacions tarden O(d) on d és la dimensió dels punts i per tant el cost per iteració seria O(nkd). Les operacions de canvi de conjunt queden amagades per aquest temps.

KNN

L'algorisme és senzill i només necessita de poder identificar els k elements a menys distància. Una forma eficient de fer això és utilitzar una cua de prioritat on a cada moment hi ha un màxim de k elements ordenats de més llunyà a menys. Així a cada pas només cal comparar el element considerat amb el primer element de la cua, i afegir-ho o no en funció de si és més proper o no.

Per tant farem n iteracions i a cada iteració haurem de fer com a màxim una inserció i una eliminació de una cua de prioritat. Això dona un temps de O(nlog(k)). Observem que aquest és millor que la implementació trivial de afegir tots els elements a un vector i ordernar-los per agafar els k més petits, que tindria cost O(nlog(n)), ja que n >> k.

MetodeRecomanadorCollaborative

L'important d'aquest mètode és que podem preprocessar la part de obtenir particions. Certament si obtenim les particions de tots els usuaris després només haurem de buscar a quina partició hi ha aquest usuari. Això ho podem guardar en una arraylist amb les particions i una arraylist on s'indiqui a quina partició pertany cada usuari.

Després quan és rep una query s'han de iterar els elements per decidir aquells que hauran de utilitzar-se per al Slope1. Finalment un cop processat el Slope1 només cal obtenir els ítems segons la seva puntuació, que ho farem ordenant-los amb una cua de prioritat.

Slope1

Aquest algorisme també permet un preprocessament. Volem guardar les desviacions entre dos ítems qualssevol per no haver-ho de recalcular a cada query. Un cop fer el preprocessament al fer una query només caldrà iterar pels ítems per calcular les mitjanes necessàries. Aquest preprocessament és pot guardar en una array de arrays sense cap problema, amb les consultes en temps constant.