1 DRIVER SLOPE ONE

1.1 Probar constructora vacía

Objetivos:
Observar que pasara cuando se acaba de inicializar una instancia del slope one

Otros elementos integrados en la prueba:
Para ver los efectos en la clase llamaremos al método printDiff mediante el mismo driver y printFreq para ver cómo se inicializan los atributos de la clase a los que podemos acceder.

Drivers construidos y integrados en la prueba:
Este test solo usará el driver de slope one tanto para la impresión como para la inicialización.

Entrada:
COMANDOS:
constructor vacio
get diff matrix
get freq matrix
get pred matrix

Salida:
El test no da error y no se ha imprimido nada por pantalla.

Resultado de la prueba: ok

Operativa: Se escribe “constructor vacío” por pantalla, posteriormente "get diff matrix", "get freq matrix", "get pred matrix". Para comprobar que los outputs de estos sean vacíos.

1.2 Test Slope One completo

Objetivos:
Comprobar el funcionamiento general del slope one usando nuestro driver.

Drivers construidos y integrados en la prueba:
El test usará el driver de Slope One y las clases usuarios, usuario.

Entrada: dataset, opcionalment subconjunt d’usuaris
Subconjunto de usuarios [id: (idItem, val)]:
1: (0, 5.1), (1, 2.6), (2, 9.3), (8, 3.6), (3, 8.3), (10, 6.2)
2: (2, 7.0)
3: (1, 6.5), (3, 5.72), (8, 1.23), (7, 6.3), (2, 3.4)
4: (10, 3.0), (4, 5.0), (2, 5.0), (9, 7.0), (1, 1.0)
5: (6, 3.0), (2, 5.0), (9, 5.0), (1, 7.0), (3, 1.0)

Usuario: 0: (0, 9.2), (1, 10.5), (4, 5.5), (6, 1.1), (8, 3.9)

COMANDOS:

"set usuarios"
"set usuario"
"computar desviaciones"
"computar predicciones"
"get mediana"
"get diff matrix"
"get freq matrix"
"get pred matrix"
"print diff"
"print freq"
"print pred"

Salida:

> Se ha setteado correctamente el nuevo conjunto de usuarios!
> Se ha setteado correctamente el nuevo usuario!
> Matriz de desviaciones y frequencias se han calculado correctamente!
> Vector de predicciones calculado correctamente.
> La mediana de la clase instanciada de slope one es: 6.040000000000001
>
[0] (0,0.0) (1,2.4999999999999996) (2,-4.200000000000001) (3,-3.200000000000001) (8,1.4999999999999996) (10,-1.1000000000000005)
[1] (0,-2.4999999999999996) (1,0.0) (2,-1.4000000000000004) (3,0.3599999999999997) (4,-1.0) (6,4.0) (7,0.20000000000000018) (8,2.135) (9,-2.0) (10,-0.5333333333333332)
[2] (0,4.200000000000001) (1,1.4000000000000004) (2,0.0) (3,0.8933333333333334) (4,0.0) (6,2.0) (7,-2.9) (8,3.9350000000000005) (9,-1.0) (10,2.3666666666666667)
[3] (0,3.200000000000001) (1,-0.3599999999999997) (2,-0.8933333333333334) (3,0.0) (4,-4.0) (6,-2.0) (7,-0.5800000000000001) (8,4.595000000000001) (9,-4.0) (10,0.050000000000000266)
[4] (1,1.0) (2,0.0) (3,4.0) (4,0.0) (6,2.0) (9,-1.0) (10,2.0)
[6] (1,-4.0) (2,-2.0) (3,2.0) (4,-2.0) (6,0.0) (9,-2.0) (10,0.0)
[7] (1,-0.20000000000000018) (2,2.9) (3,0.5800000000000001) (7,0.0) (8,5.07)
[8] (0,-1.4999999999999996) (1,-2.135) (2,-3.9350000000000005) (3,-4.595000000000001) (7,-5.07) (8,0.0) (10,-2.6)
[9] (1,2.0)(2,1.0)(3,4.0)(4,1.0)(6,2.0)(9,0.0)(10,3.0)
[10] (0,1.1000000000000005) (1,0.5333333333333332) (2,-2.3666666666666667) (3,-0.050000000000000266) (4,-2.0) (6,0.0) (8,2.6) (9,-3.0) (10,0.0)
>
[0] (0,1)(1,1)(2,1)(3,1)(8,1)(10,1)
[1] (0,1)(1,4)(2,4)(3,3)(4,2)(6,1)(7,1)(8,2)(9,2)(10,3)
[2] (0,1)(1,4)(2,5)(3,3)(4,2)(6,1)(7,1)(8,2)(9,2)(10,3)
[3] (0,1)(1,3)(2,3)(3,3)(4,1)(6,1)(7,1)(8,2)(9,1)(10,2)
[4] (1,2)(2,2)(3,1)(4,2)(6,1)(9,2)(10,2)
[6] (1,1)(2,1)(3,1)(4,1)(6,1)(9,1)(10,1)
[7] (1,1)(2,1)(3,1)(7,1)(8,1)
[8] (0,1)(1,2)(2,2)(3,2)(7,1)(8,2)(10,1)
[9] (1,2)(2,2)(3,1)(4,2)(6,1)(9,2)(10,2)
[10] (0,1)(1,3)(2,3)(3,2)(4,2)(6,1)(8,1)(9,2)(10,3)

>
(2, 7.193500000000001)
(3, 6.219375000000001)
(7, 8.475000000000001)
(9, 7.040000000000001)
(10, 6.319166666666668)
>
Item 0: (0,0.0) (1,2.4999999999999996) (2,-4.200000000000001) (3,-3.200000000000001)
(8,1.4999999999999996) (10,-1.1000000000000005)
Item 1: (0,-2.4999999999999996) (1,0.0) (2,-1.4000000000000004)
(3,0.3599999999999997) (4,-1.0) (6,4.0) (7,0.20000000000000018) (8,2.135) (9,-2.0)
(10,-0.5333333333333332)
Item 2: (0,4.200000000000001) (1,1.4000000000000004) (2,0.0) (3,0.8933333333333334)
(4,0.0) (6,2.0) (7,-2.9) (8,3.9350000000000005) (9,-1.0) (10,2.3666666666666667)
Item 3: (0,3.200000000000001) (1,-0.3599999999999997) (2,-0.8933333333333334) (3,0.0)
(4,-4.0) (6,-2.0) (7,-0.5800000000000001) (8,4.595000000000001)(9,-4.0)
(10,0.050000000000000266)
Item 4: (1,1.0)(2,0.0)(3,4.0)(4,0.0)(6,2.0)(9,-1.0)(10,2.0)
Item 6: (1,-4.0)(2,-2.0)(3,2.0)(4,-2.0)(6,0.0)(9,-2.0)(10,0.0)
Item 7: (1,-0.20000000000000018)(2,2.9)(3,0.5800000000000001)(7,0.0)(8,5.07)
Item 8: (0,-1.4999999999999996) (1,-2.135)(2,-3.9350000000000005)
(3,-4.595000000000001) (7,-5.07) (8,0.0) (10,-2.6)
Item 9: (1,2.0)(2,1.0)(3,4.0)(4,1.0)(6,2.0)(9,0.0)(10,3.0)
Item 10: (0,1.1000000000000005) (1,0.5333333333333332) (2,-2.3666666666666667)
(3,-0.050000000000000266) (4,-2.0) (6,0.0) (8,2.6) (9,-3.0) (10,0.0)

Item 0: (0,1)(1,1)(2,1)(3,1)(8,1)(10,1)
Item 1: (0,1)(1,4)(2,4)(3,3)(4,2)(6,1)(7,1)(8,2)(9,2)(10,3)
Item 2: (0,1)(1,4)(2,5)(3,3)(4,2)(6,1)(7,1)(8,2)(9,2)(10,3)
Item 3: (0,1)(1,3)(2,3)(3,3)(4,1)(6,1)(7,1)(8,2)(9,1)(10,2)
Item 4: (1,2)(2,2)(3,1)(4,2)(6,1)(9,2)(10,2)
Item 6: (1,1)(2,1)(3,1)(4,1)(6,1)(9,1)(10,1)
Item 7: (1,1)(2,1)(3,1)(7,1)(8,1)
Item 8: (0,1)(1,2)(2,2)(3,2)(7,1)(8,2)(10,1)
Item 9: (1,2)(2,2)(3,1)(4,2)(6,1)(9,2)(10,2)
Item 10: (0,1)(1,3)(2,3)(3,2)(4,2)(6,1)(8,1)(9,2)(10,3)

PREDICTION ITEM 2 IS 7.193500000000001
PREDICTION ITEM 3 IS 6.219375000000001
PREDICTION ITEM 7 IS 8.475000000000001
PREDICTION ITEM 9 IS 7.040000000000001
PREDICTION ITEM 10 IS 6.319166666666668

1.3 Setters
Descripción: “setValoraciones” “setOriginalUserId” “addValoracio”

Objetivos:
Comprobar el correcto funcionamiento de los setters en la clase Usuario.
Otros elementos construidos e integrados para la prueba:
No hacen falta más elementos aparte del driver.
Entrada: valor del atributo que queremos modificar.
Caso “setOriginalUserId”:
2378
Salida: valor del atributo, efectivamente modificado.
Caso “setOriginalUserId”:
2378
Resultado de la prueba: Ok
Operativa: Escribir el comando por pantalla, y a continuación introducir el valor del elemento a modificar.

1.4 Booleans
Descripción: “equals” “tieneValoracion”
Objetivos:
Comprobar el correcto funcionamiento de las funciones booleanas de la clase Usuario.
Otros elementos construidos e integrados para la prueba:
No hacen falta más elementos aparte del driver.
Entrada:
Caso “equals”: identificador de los dos usuarios que queremos comparar
Caso 1: 1 1
Caso 2: 1 2
Salida: True si ambos usuarios tienen el mismo identificador, falso en caso contrario.
Caso 1: True
Caso 2: False
Resultado de la prueba: Ok
Operativa: Escribir el comando por pantalla, y a continuación introducir los ids de los usuarios a comparar. En el caso de tieneValoración con el comando ya basta.

2 DRIVER K-MEANS

2.1 Probar constructor vacío

Descripción: "constructor vacio"
Objetivos:
Comprobar el correcto funcionamiento de la constructora por defecto de KMeans
Otros elementos construidos e integrados para la prueba:
Para la inicialización no hacen falta más elementos aparte del driver.
Entrada:
Solamente hace falta introducir el comando.
Salida: "Hecho"
Resultado de la prueba: Ok
Operativa: Escribir el comando por pantalla. Si funciona correctamente se escribirá la máxima valoración.

2.2. Getters

Descripción: "*getmaxsc" "getminsc" " get centroides" "get k" "get cota" "get cM" "get users cluster"*
Objetivos:
Comprobar que se devuelve el valor correcto al llamar a los getters de la clase KMeans
Otros elementos construidos e integrados para la prueba:
Elementos de las clases KMeans, Usuario, Usuarios, Centroide.
Uso de ArrayLists y Maps.

Entrada: valor para el elemento que vamos a recuperar a continuación.
Caso "get centroides"
n de centroides a crear, seguido de n identificadores
3 0 1 2
Salida:
Caso "get centroides"
identificadores de todos los centroides en ArrayList<Centroide>
Centroide: 0
Centroide: 1
Centroide: 2
Resultado de la prueba: Ok
Operativa: Escribir el comando por pantalla. Si funciona correctamente se devolverá el valor del elemento pasado en la entrada

2.3. Setters

Descripción: *" set centroides" "set k" "set cota" "set cM" "restore"*

Objetivos:

Comprobar que se devuelve el valor correcto al llamar a los setters de la clase KMeans

Otros elementos construidos e integrados para la prueba:

Elementos de las clases KMeans, Usuario, Usuarios, Centroide.

Uso de ArrayLists y Maps.

Entrada: valor para el elemento que vamos a modificar a continuación.

Caso "set centroides"

n de centroides a crear, seguido de n identificadores, cada uno con sus valoraciones

3

0 2 0.5 5.0

1 2 3.0 2.5

2 2 1.0 1.0

Salida:

Caso "set centroides"

identificadores de todos los centroides en ArrayList<Centroide> antes de llamar al setter

identificadores de todos los centroides en ArrayList<Centroide> después de llamar al setter

0

1

2

Resultado de la prueba: Ok

Operativa: Escribir el comando por pantalla. Si funciona correctamente se devolverá el valor del elemento que queríamos modificar, efectivamente modificado

2.4. Funciones y métodos

Descripción: Funciones que devuelven resultados parciales o modifican una parte del estado del algoritmo de KMeans

Objetivos:

Comprobar el correcto funcionamiento estas funciones y métodos en KMeans

Otros elementos construidos e integrados para la prueba:

Elementos de las clases KMeans, Usuario, Usuarios, Centroide.

Uso de ArrayLists y Maps.

Entrada:

"distancia euclídea"

ArrayList<Centroide> c = [Centroide (0, {0=4.5, 1=2.2, 2=0.0, 3=1.0})]

usuario 1 = {0=3.5, 1=4.5, 2=4.0, 3=1.0}

usuario 2 = {0=2.5, 1=5.0}

usuario 3 = {}

"nearest centroids"

centroide 0 = {0=5.0, 1=5.0, 2=5.0, 3=5.0}
centroide 1 = {0=2.5, 1=2.5, 2=2.5, 3=2.5}
centroide 2 = {0=5.0, 1=5.0, 2=5.0, 3=5.0}

usuario 1 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}
usuario 2 = {0=1.0, 1=1.0, 2=1.0, 3=1.0}
usuario 3 = {0=2.0, 1=2.0, 2=2.0, 3=2.0}

“centroid relocation”
centroide 0 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}

usuario 1 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}
usuario 2 = {0=5.0, 1=5.0, 2=5.0, 3=4.5}
usuario 3 = {0=4.0, 1=4.5, 2=3.0, 3=5.0}

“random centroids”
introducir valor de k:
2

“furthest centroids”
centroide 0 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}
centroide 1 = {0=1.0, 1=2.0, 2=1.5, 3=0.5}
centroide 2 = {0=4.5, 1=4.0, 2=5.0, 3=4.5}
centroide 3 = {0=1.2, 1=1.7, 2=1.6, 3=0.3}
centroide 4 = {0=3.0, 1=2.0, 2=2.5, 3=3.0}
centroide 5 = {(0=2.7, 1=2.2, 2=2.4, 3=2.9}

“add to centroid”
centroide 0 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}
usuario 0 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}

“terminacion temprana”
clusterMap1
clusterMap2

Salida:
“distancia euclídea"
4.851803788283282
3.4409301068170506
0.0

“nearest centroids”
0
1
2

“centroid relocation”
4.5
4.333333333333333
4.166666666666667
4.666666666666667

“random centroids”
Centroide {id} valoraciones: {0=random, 1= random...}

“furthest centroids”
centroide 0 = {0=4.5, 1=4.5, 2=4.5, 3=4.5}
centroide 1 = {0=1.0, 1=2.0, 2=1.5, 3=0.5}
centroide 2 = {0=3.0, 1=2.0, 2=2.5, 3=3.0}

“add to centroid”
Imprime el size de los clusters, antes y después de añadir un elemento a uno de ellos.
Estado del cluster antes de la adición
Estado del cluster después de la adición
Centroide 0: 1

“terminacion temprana”
si los clusterMaps son iguales devuelve “El estado no ha variado. El algoritmo terminará.”
si no lo son, "Distintos estados. El algoritmo continuará su ejecución."

Resultado de la prueba: Ok
Operativa: Escribir el comando por pantalla y añadir la información pedida por el programa.

3 DRIVER NEAREST NEIGHBORS

3.1 Probar constructora vacía

Objetivos:
Observar que pasara cuando se acaba de inicializar una instancia del nearest neighbors

Drivers construidos y integrados en la prueba:
Este test solo usará el driver de nearest neighbors tanto para la impresión como para la inicialización.

Entrada:
COMANDOS:
constructor vacio

Salida:
Se ha llamado a la constructora vacía!

Resultado de la prueba: ok

Operativa: Se escribe "Se ha llamado a la constructora vacía!" por pantalla.

3.2 Probar constructora con parámetros

Objetivos:
Observar que pasara cuando se acaba de inicializar una instancia del nearest neighbors con el parámetro k

Drivers construidos y integrados en la prueba:
Este test solo usará el driver de nearest neighbors tanto para la impresión como para la inicialización.

Entrada:
COMANDOS:
constructor no vacio

Salida:
Se ha llamado a la constructora con parametros!

Resultado de la prueba: ok

Operativa: Se escribe "Se ha llamado a la constructora con parámetros!" por pantalla.

3.3 Test Nearest Neighbors completo

Objetivos:

Comprobar el funcionamiento general del nearest neighbors usando nuestro driver.

Drivers construidos e integrados en la prueba:

El test usará el driver de Nearest Neighbors y las clases item, items y usuario.

Entrada: Conjunto de items con sus propiedades y un usuario

El test nos irá pidiendo la información. Cabe remarcar que a la hora de agregar atributos a objetos, siempre tienen ser los mismos atributos para todos los items creados: el nombre y el tipo del atributo han de estar presentes y ser iguales para todos los items, lo que varía es el valor que le asignamos a este atributo.

COMANDOS:

"computeAllPredicciones"
"distanceNums"
"distanceBooleans"
"distanceCategorico"
"distanceCategoricoMultiples"

Salida:

Los k items en orden de preferencia para el usuario son: (y aquí se imprimen los k items)

4 DRIVER RECOMENDACIONES

Este driver solo ofrece la funcionalidad pedir recomendación, pero dentro de esta funcionalidad hay diferentes opciones (según el algoritmo que escoja el usuario).

4.1 Pedir recomendación

Objetivos:

Obtener un conjunto de recomendaciones a partir de un usuario y sus ítems valorados.

Drivers construidos y integrados en la prueba:

Este test solo usará el driver de recomendaciones tanto para la impresión como para la inicialización.

Entrada:

En la entrada tendremos diferentes valores:

- Id del usuario
- Número de items a valorar
- Path de dónde leer
- Tipo de algoritmo (CF/CB/Hybrid)
- Si quiere evaluación el usuario

Salida:

Se verá el número de ítems que el usuario haya pedido que se le recomiende, de más a menos recomendado según el algoritmo que haya decidido.

Resultado de la prueba: ok

Operativa: Se muestra el conjunto de recomendaciones.