Integer.valueOf(String): Integer 클래스를 return하기 때문에 산술 연산을 할 수 없다.

Integer.parseInt(String): int 형을 return하기 때문에 산술 연산을 할 수 있다.

int 형은 자료형이고 null이 없으며, Integer는 객체이며 null 값을 가지고 있다.

extends: abstract class를 상속받아서 +@시키겠다. 선언된 것도 받아옴

implements: class라는 interface를 구현하겠다. 함수만 받아옴

Recommender

public

class Recommender

선언

TreeMap: support1 <Integer, Integer> MovieId (영화 set) -> Num

support2 <IntPair, Integer> MovieId x MovieId -> Num

support3 <IntTriple, Integer> MovieId x MovieId x MovieId -> Num

PropertiesConfiguration: config

int: min\_supports

min\_evidence\_3

double: threshold\_2

threshold\_3

Func

public

void train (MovieRatingData data)

선언: TreeMap: Baskets <Integer, HashSet<Integer>>(UserID -> Set<Movield>)

HashSet: Basket <Integer>

사용: TreeMap<>.getBaskets

updateSupport1, updateSupport2, updateSupport3

int predict (HashSet<Integer> profile, Integer q)

사용: predictPair, predictTriple

만약 predictPair가 1일 경우 return 1 아니면 predictTriple

void updateSupport1 (HashSet<Integer> Basket)

사용: Basket에서 item을 하나씩 가져온다.

가져온 아이템을 support1.get()을 사용해서 Integer c에 넣음.

만약 c가 null이면 c에 new Integer(1)

아니면 c에 new Integer (c.intValue() + 1)

support1에 item, c를 넣음.

void updateSupport2 (HashSet<Integer> Basket)

사용: Basket의 size가 2보다 같거나 클 때

Sets.combination(Basket,2)에서 Set<Integer> pire를 하나씩 가져와서 Integer c에 support2.get(new IntPair(pair))를 넣는다.

만약 c == null이면 c = new Integer(1)

아니면 c에 new Integer (c.intValue() + 1)

support2에 new IntPair(Pair), c를 넣음.

void updateSupport3(HashSet<Integer> Basket)

선언: HashSet: \_Basket <Integer>

사용: Basket에서 Integer elem을 하나씩 가져와서 support1.get(elem)이 min\_supports 보다 크거나 같으면 \_Basket.add(elem)

Basket에 \_Basket을 넣는다.

만약 Basket.size()이 3이상이면 Sets.combination(Basket, 3))에서 Set<Integer> triple을 하나씩 가져와서 support3.get(new IntTriple(triple))을 c에 넣는다.

만약 c == null이면 c = new Integer(1)

아니면 c에 new Integer (c.intValue() + 1)

support3에 new IntTriple(triple), c를 넣음.

private

int predictPair (HashSet <Integer> profile, Integer q)

/\* TODO: implement this method \*/ //DONE

for(Integer i : profile){

Integer den = support1.get(i);

Integer num = support2.get(new IntPair(i, q));

if (num == null)

continue;

if (num.intValue() < min\_supports)

continue;

if ((double)num/(double)den >= threshold\_2)

return 1;

}

return 0 ;

}

선언: Integer: den = support1.get(i)

num = support2.get(new IntPair(i, q))

사용: profile에서 Integer i를 하나씩 꺼내어 den에 support1.get(i)를 넣는다.

그리고 num에도 support2.get(new IntPair(i, q))를 넣는다.

만약 num이 null이라면 계속하고 num.intValue()< min\_supports가 작으면 계속 한다.

만약 ((double)num/(double)den >= threshold\_2이면 return 1을 한다.

return 0;

int predictTriple(HashSet <Integer> profile, Integer q)

선언: int: evidence

TreeSet: t <Integer>

IntTriple: item

Integer: num <- support3.get(item)

사용: profile.size()가 2보다 작은 경우 return 0

Sets.combinations(profile, 2)에서 Set<Integer> p을 하나씩 꺼내서 Integer den에 support2.get(new IntPair(p))을 넣는다.

if (den == null) continue;

if (num.intValue() < min\_supports) continue ;

if ((double)num / (double)den >= threshold\_3) evidence++ ;

if (evidence >= min\_evidence\_3) return 1 ;

return 0;

implement Comparable

class IntPair

선언

int: first

second

func

public

IntPair( int first, int second)

사용: first가 second보다 이하일 때 this.first = first, this.second = second; 예~버그잡음

아니면 this.first = second, this.second = first;

IntPair(Set <Integer> s)

선언: Integer [] elem = s.toArray(new Integer[2]);

사용: 만약 elem[0]이 elem[1]보다 작다면 this.first = elem[0], this.second = elem[1] ;

아니면 this.first = elem[1] , this.second = elem[0] ;

int compareTo(Object obj)

선언: IntPair: p

사용: this.first가 p.first보다 작으면 return -1, 크면 return 1

return (this.second – p.second);

class IntTriple

선언

//왜 그냥 넣는 애들이 없을까

만들자

int []: elem

func

IntTriple(Set <Integer> s)

TODO

public

int compareTo(Object obj)

TODO

///////////////////////////////////

int first ;

int second ;

int third;

int temp;

Integer [] elem ;

IntTriple(Set<Integer> s) {

/\* TODO: implement this method \*/

elem = s.toArray(new Integer[3]);

for(int i = 0; i < 3; i ++){

for(int j = 0; j < 2; j++ ){

if(elem[j+1] < elem[j]){

temp = elem[j];

elem[j] = elem[j+1];

elem[j+1] = temp;

}

}

}

this.first = elem[0];

this.second = elem[1];

this.third = elem[2];

}

public int compareTo(Object obj) {

/\* TODO: implement this method \*/

return 0 ;

}