

Esercitazione 2 – Classi, Aggregazione, Ereditarietà, Classi Astratte

Si definiscano le classi Java necessarie per modellare le entità tupla, cluster e insieme di cluster. *Per ogni classe, definire opportunamente la visibilità dei membri*

- Integrare nel progetto la classe *ArraySet* che modella il dato astratto *insieme di interi* e ne fornisce una realizzazione basata su vettore di booleani. (In Allegato)
- Modificare la classe *DiscreteAttribute* con la aggiunta del seguente metodo

int frequency(Data data, ArraySet idList, String v)

Input: riferimento ad un oggetto *Data*, riferimento ad un oggetto *ArraySet* (che mantiene l'insieme degli indici di riga di alcune tuple memorizzate in data), valore discreto

Output: numero di occorrenze del valore discreto (intero)

Comportamento: Determina il numero di volte che il valore *v* compare in corrispondenza dell'attributo corrente (indice di colonna) negli esempi memorizzati in *data* e indicizzate (per riga) da *idList*

- Definire la classe astratta *Item* che modella un generico item (coppia attributo-valore, per esempio Outlook="Sunny").

Attributi

Attribute attribute; attributo coinvolto nell'item

Object value; valore assegnato all'attributo

Metodi

Item(Attribute attribute, Object value)

Comportamento: inizializza i valori dei membri attributi

Attribute getAttribute()

Comportamento: restituisce attribute;

Object getValue()

Comportamento: restituisce value;

public String toString()

Comportamento: restituisce value

abstract double distance(Object a)

L'implementazione sarà diversa per item discreto e item continuo

void update(Data data, ArraySet clusteredData)

Input: riferimento ad un oggetto della classe Data, insieme di indici delle righe della matrice in data che formano il cluster

*Comportamento: Modifica il membro **value**, assegnandogli il valore restituito da data.computePrototype(clusteredData,**attribute**);*

N.B. ComputePrototype(...) andrà definito in Data (vedi specifiche nel seguito)

- Definire la classe **Discreteltem** che estende la classe **Item** e rappresenta una coppia <Attributo discreto- valore discreto> (per esempio Outlook="Sunny")

Metodi

DiscreteItem(DiscreteAttribute attribute, String value)

Comportamento: Invoca il costruttore della classe madre

double distance(Object a)

Comportamento: Restituisce 0 se (getValue().equals(a)) , 1 altrimenti.

- Definire la classe **Tuple** che rappresenta una tupla come sequenza di coppie attributo-valore.

Attributi

Item [] tuple;

Metodi

Tuple(int size)

Input: numero di item che costituirà la tupla

Comportamento: costruisce l'oggetto riferito da **tuple**

int getLength()

Comportamento: restituisce **tuple.length**

Item get(int i)

Comportamento: restituisce lo tem in posizione *i*

void add(Item c,int i)

Comportamento: memorizza *c* in **tuple[i]**

double getDistance(Tuple obj)

Comportamento: determina la distanza tra la tupla riferita da `obj` e la tupla corrente (riferita da `this`). La distanza è ottenuta come la somma delle distanze tra gli item in posizioni eguali nelle due tuple. Fare uso di **double distance(Object a)** di `Item`

double avgDistance(Data data, int clusteredData[])

Comportamento: restituisce la media delle distanze tra la tupla corrente e quelle ottenibili dalle righe della matrice in `data` aventi indice in `clusteredData`.

```
double avgDistance(Data data, int clusteredData[]){
    double p=0.0,sumD=0.0;
    for(int i=0;i<clusteredData.length;i++){
        double d= getDistance(data.getItemSet(clusteredData[i]));
        sumD+=d;
    }
    p=sumD/clusteredData.length;
    return p;
}
```

N.B. Vedere la specifica del metodo Tuple getItemSet(int index) da aggiungere alla classe Data e specificato subito dopo.

- Modificare la classe `Data` con l'aggiunta dei seguenti metodi

Tuple getItemSet(int index)

Input: indice di riga

Comportamento: Crea e restituisce un oggetto di `Tuple` che modella come sequenza di coppie Attributo-valore la *i*-esima riga in `data`.

```
Tuple getItemSet(int index){
    Tuple tuple=new Tuple(explanatorySet.length);
    for(int i=0;i<explanatorySet.length;i++){
        tuple.add(new DiscreteItem(explanatorySet[i],
            (String) data[index][i]),i);
    }
    return tuple;
}
```

int[] sampling(int k)

Input: numero di cluster da generare

Output: array, di *k* interi rappresentanti gli indici di riga in `data` per le

tuple inizialmente scelte come centroidi (passo 1 del k-means)

```
int[] sampling(int k){  
  
    int centroidIndexes[]=new int[k];  
    //choose k random different centroids in data.  
    Random rand=new Random();  
    rand.setSeed(System.currentTimeMillis());  
    for (int i=0;i<k;i++){  
        boolean found=false;  
        int c;  
        do  
        {  
            found=false;  
            c=rand.nextInt(getNumberOfExamples());  
            // verify that centroid[c] is not equal to a centroide  
            already stored in CentroidIndexes  
            for(int j=0;j<i;j++){  
                if(compare(centroidIndexes[j],c)){  
                    found=true;  
                    break;  
                }  
            }  
            while(found);  
            centroidIndexes[i]=c;  
        }  
    }  
    return centroidIndexes;  
}  
  
private boolean compare(int i,int j)
```

Input: indici di due righe nell'insieme in Data

*Comportamento: restituisce vero se le due righe di **data** contengono gli stessi valori, falso altrimenti*

Object computePrototype(ArraySet idList, Attribute attribute)

Input: insieme di indici di riga, attributo rispetto al quale calcolare il prototipo (centroide)

*Output: valore centroide rispetto ad **attribute***

Comportamento: restituisce computePrototype(idList, (DiscreteAttribute)attribute)

String computePrototype(ArraySet idList, DiscreteAttribute attribute)

Input: insieme degli indici delle righe di data appartenenti as un cluster, attributo discreto rispetto al quale calcolare il prototipo (centroide)

Output: centroide rispetto ad attribute

Comportamento: Determina il valore che occorre più frequentemente per attribute nel sottoinsieme di dati individuato da idList (fare uso del metodo frequency(...) di DiscretAttribute).

- Aggiungere la classe **Cluster** che modella un cluster (vedi allegato).
- Definire la classe **ClusterSet** che rappresenta un insieme di cluster (determinati da k-means)

Attributi

Cluster C[];

int i=0; posizione valida per la memorizzazione di un nuovo cluster in **C**

Metodi

ClusterSet(int k)

Input: numero di cluster da generare (k-means)

Output:

Comportamento: creo l'oggetto array riferito da C

void add(Cluster c)

Comportamento: assegna c a C[i] e incrementa i.

Cluster get(int i)

Comportamento: restituisce C[i]

void initializeCentroids(Data data)

Comportamento: sceglie i centroidi, crea un cluster per ogni centroide e lo memorizza in C

```
void initializeCentroids(Data data){  
  
    int centroidIndexes[]=data.sampling(C.length);  
    for(int i=0;i<centroidIndexes.length;i++)  
    {  
        Tuple centroidI=data.getItemSet(centroidIndexes[i]);  
        add(new Cluster(centroidI));  
    }  
}
```

Cluster nearestCluster(Tuple tuple)

Input: riferimento ad un oggetto Tuple

Output: cluster più “vicino” alla tupla passata

Comportamento: Calcola la distanza tra la tupla riferita da tuple ed il centroide di ciascun cluster in C e restituisce il cluster più vicino (fare uso del metodo getDistance() della classe Tuple).

Cluster currentCluster(int id)

Input: indice di una riga nella matrice in Data

Comportamento: Identifica e restituisce il cluster a cui la tupla rappresentate l'esempio identificato da id. Se la tupla non è inclusa in nessun cluster restituisce null (fare uso del metodo contain() della classe Cluster).

void updateCentroids(Data data)

Comportamento: calcola il nuovo centroide per ciascun cluster in C (fare uso del metodo computeCentroid() della classe Cluster)

public String toString()

Input:

Output:

Comportamento: Restituisce una stringa fatta da ciascun centroide dell'insieme dei cluster.

public String toString(Data data)

Input:

Output:

Comportamento: Restituisce una stringa che descriva lo stato di ciascun cluster in C.

```
public String toString(Data data ){
    String str="";
    for(int i=0;i<C.length;i++){
        if (C[i]!=null){
            str+=i+": "+C[i].toString(data)+"\n";
        }
    }
    return str;
}
```

■ Definire la classe **KMeansMiner** che include l'implementazione dell'algoritmo kmeans

Attributi

ClusterSet C;

Metodi

KmeansMiner(int k)

Input: numero di cluster da generare

Comportamento: Crea l'oggetto array riferito da **C**

ClusterSet getC()

Comportamento: restituisce **C**

int kmeans(Data data)

Output: numero di iterazioni eseguite

Comportamento: Esegue l'algoritmo k-means eseguendo i passi dello pseudo-codice:

1. Scelta casuale di centroidi per k clusters
2. Assegnazione di ciascuna riga della matrice in data al cluster avente centroide più vicino all'esempio.
3. Calcolo dei nuovi centroidi per ciascun cluster
4. Ripete i passi 2 e 3. finché due iterazioni consecutive non restituiscono centroidi uguali .

```
int kmeans(Data data){
    int numberOfIterations=0;
    //STEP 1
    C.initializeCentroids(data);
    boolean changedCluster=false;
    do{
        numberOfIterations++;
        //STEP 2
        changedCluster=false;
        for(int i=0;i<data.getNumberOfExamples();i++){
            Cluster nearestCluster = C.nearestCluster(
                data.getItemSet(i));
            Cluster oldCluster=C.currentCluster(i);
            boolean currentChange=nearestCluster.addData(i);
            if(currentChange)
                changedCluster=true;
            //rimuovo la tupla dal vecchio cluster
            if(currentChange && oldCluster!=null)
                //il nodo va rimosso dal suo vecchio cluster
                oldCluster.removeTuple(i);
        }
        //STEP 3
```

```

        C.updateCentroids(data);
    }
    while(changedCluster);
    return numberOfIterations;
}

```

■ *Importare la classe **MainTest** nel progetto. Un possibile esempi odi esecuzione è riportato nel seguito:*

```

0:sunny,hot,high,weak,no
1:sunny,hot,high,strong,no
2:overcast,hot,high,weak,yes
3:rain,mild,high,weak,yes
4:rain,cool,normal,weak,yes
5:rain,cool,normal,strong,no
6:overcast,cool,normal,strong,yes
7:sunny,mild,high,weak,no
8:sunny,cool,normal,weak,yes
9:rain,mild,normal,weak,yes
10:sunny,mild,normal,strong,yes
11:overcast,mild,high,strong,yes
12:overcast,hot,normal,weak,yes
13:rain,mild,high,strong,no

```

Numero di Iterazione:4

0:Centroid=(sunny hot high weak no)

Examples:

```

[sunny hot high weak no ] dist=0.0
[sunny hot high strong no ] dist=1.0
[sunny mild high weak no ] dist=1.0

```

AvgDistance=0.6666666666666666

1:Centroid=(overcast cool normal weak yes)

Examples:

```

[overcast hot high weak yes ] dist=2.0
[rain cool normal weak yes ] dist=1.0
[overcast cool normal strong yes ] dist=1.0
[sunny cool normal weak yes ] dist=1.0
[overcast hot normal weak yes ] dist=1.0

```

AvgDistance=1.2

2:Centroid=(rain mild high strong yes)

Examples:

```

[rain mild high weak yes ] dist=1.0
[rain cool normal strong no ] dist=3.0
[rain mild normal weak yes ] dist=2.0
[sunny mild normal strong yes ] dist=2.0
[overcast mild high strong yes ] dist=1.0
[rain mild high strong no ] dist=1.0

```

AvgDistance=1.6666666666666667