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)*

- *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

- *Definire la classe **DiscreteItem** 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 del seguente metodo*

*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;
}
```

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

Attributi

private Cluster C[]=**new** Cluster[0];

Metodi

ClusterSet()

***void** add(Cluster c)*

definito come segue

```
void add(Cluster c){
      Cluster tempC[]=new Cluster[C.length+1];
      for(int i=0;i<C.length;i++)
            tempC[i]=C[i];
      tempC[C.length]=c;
      C=tempC;
}
```

*Cluster get(**int** i)*

Comportamento: restituisce C[i]

***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 QTMiner che include l'implementazione dell'algoritmo QT*

Attributi

ClusterSet C;

double radius

Metodi

QTMiner(double radius)

Input: raggio dei cluster

Comportamento: Crea l'oggetto array riferito da C e inizializza radius con il parametro passato come input

ClusterSet getC()

Comportamento: restituisce C

***int** compute(Data data)*

Output: numero di cluster scoperti

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

1. *Costruisce un cluster per ciascuna tupla non ancora clusterizzata, includendo nel cluster i punti (non ancora clusterizzati in alcun altro cluster) che ricadano nel vicinato sferico della tuple avente raggio radius*
2. *Salva il candidato cluster più popoloso e rimuove tutti punti di tale cluster dall'elenco delle tuple ancora da clusterizzare*
3. *Ritorna al passo 1 finchè ci sono ancora tuple da assegnare ad un cluster*

```
4.      int compute(Data data){
5.            int numclusters=0;
6.            boolean isClustered[]=new boolean[data.getNumberOfExamples()];
7.            for(int i=0;i<isClustered.length;i++)
8.                  isClustered[i]=false;
9.
10.           int countClustered=0;
11.           while(countClustered!=data.getNumberOfExamples())
12.           {
13.                 //Ricerca cluster più popoloso
14.                 Cluster c=buildCandidateCluster(data, isClustered);
15.                 C.add(c);
16.                 numclusters++;
```

```
17.                     //Rimuovo tuple clusterizzate da dataset
18.
19.                     int clusteredTupleId[]=c.iterator();
20.                     for(int i=0;i<clusteredTupleId.length;i++){
21.                           isClustered[clusteredTupleId[i]]=true;
22.                     }
23.                     countClustered+=c.getSize();
24.
25.               }
26.
27.               return numclusters;
28.         }
29.                     }
```

buildCandidateCluster(Data data, boolean isClustered[])

Input: insiene di tuple da raggruppare in cluster, informazione booleana sullo stato di clusterizzazione di una tupla (per esempio isClustered[i]=false se la tupla i-esima di data non è ancora assegnata ad alcun cluster di C, true altrimenti)

Comportamento: costruisce un cluster per ciascuna tupla di data non ancora clusterizzata in un cluster di C e restituisce il cluster candidato più popoloso

- *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

Number of clusters:3
0:Centroid=(rain mild high weak yes )
Examples:
[overcast hot high weak yes ] dist=2.0
[rain mild high weak yes ] dist=0.0
[rain cool normal weak yes ] dist=2.0
[sunny mild high weak no ] dist=2.0
[rain mild normal weak yes ] dist=1.0
```

```
[overcast mild high strong yes ] dist=2.0
[rain mild high strong no ] dist=2.0

AvgDistance=1.5714285714285714
1:Centroid=(overcast cool normal strong yes )
Examples:
[rain cool normal strong no ] dist=2.0
[overcast cool normal strong yes ] dist=0.0
[sunny cool normal weak yes ] dist=2.0
[sunny mild normal strong yes ] dist=2.0
[overcast hot normal weak yes ] dist=2.0

AvgDistance=1.6
2:Centroid=(sunny hot high weak no )
Examples:
[sunny hot high weak no ] dist=0.0
[sunny hot high strong no ] dist=1.0

AvgDistance=0.5
```