Exercise 2 - Classes, Aggregation, Inheritance, Abstract Classes

S idefine thenecessary Java classesfor modellingthe entities, clusterand cluster set . For each class, appropriately define the visibility of the members

- *Integrating the classArraySet into the project, which models the abstract data set of integers and provides a vector-based realisation of Booleans. (Attached)*
- *Modify theDiscreteAttribute class class by adding the following method*

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

Input: reference to a object Date, reference toa ArraySet object (which maintains the set of row indicesof sometuples stored in data) , discrete value

Output: number of discrete value o ccurrences (integer)

Behaviour: Determines the number of times thev-value appears at the current attribute (column index) in the examples stored in date and indexed (by row) by idList

- *Define the **abstract** classItem that models a generic item (attribute-value pair, e.g. Outlook="Sunny").*

Attributes

Attribute attribute; attribute involved in the

item Object value; value assigned to the

attribute

Methods

Item(Attribute attribute, Object value)

Behaviour: initialise attribute member values

Attribute getAttribute()

Behaviour: Returns attribute;

Object getValue()

Behaviour: returns value ;

public String toStri ng()

Behaviour: returns value

***abstract double** distance(Object a)*

Implementation will be different for discrete item and continuous item

***void** update(Data data, ArraySetcluster edData)*

Input: reference to an object of class Data, set of i of the rows of the date matrix forming the cluster

Behaviour: Modify the membervalue , assigning it the value returned by data.computePrototype(cluster edData, attribute);

N.B. ComputePrototype(...) shall be defined at Data (see specifi that below)

- *Define the class **DiscreteItem** which extends the class **Item** class and represents a pair <Attribute discrete - value discrete> (e.g. Outlook="Sunny")*

Methods

DiscreteItem(DiscreteAttribute attribute, String value)

Behaviour: In voca the constructor of the parent class

***double** distance(Object a)*

Behaviour: Returns 0 if (getValue().equals(a)) , 1 *otherwise.*

- *Define the* ***classTuple*** *which represents a tuple as a sequence of attribute-value pairs.*

  Attributes

  Item [ ] tuples;

  Methods

  *Tuple(**int** size)*

  Input: number of items that will constitute the tuple

  Behaviour: constructs the object referred to by tuples

  ***int** getLength()*

  Behaviour: returns tuple.length

  Item get(int i)

  Behaviour: returns the temin position i

  ***void** add(Item c, **int** i)*

  Behaviour: store c in tuples[i]

  ***double** getDistance(Tuple obj)*

Behaviour: determines the distance between the tuple referred by obj and the current tuple (referred by this). The distance is obtained as the sum of the distances between items at equal positions in the two tuples. Use ***didouble*** *distance(Object a) of Item*

double *avgDistance(Data data,* ***int*** *cluster edData[ ])*

Behaviour: returns the average of the distances between the current tuple and those obtained from the rows of the indata matrix having an index in cluster edData .

```
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. See the specification of themethod getItemSet(***int*** *index) to be added to the Data class and specified immediately thereafter.*

- *Modify the Data class by adding the following methods Tuple*

  getI temSet(***int*** *index)*

  Input: line index

  Behaviour: Creates and returns aTuple object that models as a sequence of Attribute -value pairs the i -th row in date.

```
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: number of clusters to be generated

  Output: array, of k integers representing the row indices in date for the

tuples initially chosen as centroids (step 1 of 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 centroid
                    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: indices of two rows in the set in Data

Behaviour: returns true if the two didata lines contain the same values, false otherwise

Object computePrototype(ArraySet idList, Attribute attribute)

Input: set of row indices, attribute against which to calculate the prototype (centroid)

Output: centroid value with respect to attribute

Behaviour: returns computePrototype(idList, (DiscreteAttribute)attribute)

String computePrototype(ArraySet idList, DiscreteAttribute attribute)

Input: set of data rig he indices belonging asa cluster, discrete attribute against which to calculate the prototype (centroid)

Output: centroid with respect to attribute

Behaviour: Determines the most frequently needed value for attributes in the subset of attributes identified by idList (make use of the frequency(...) method of DiscretAttribute).

- *Add the class Cluster that models a cluster (see attachment).*
- *Define the classCluster Set which represents a set of diclusters (determined by k -means)*

Attributes

Cluster C[ ];

***int** i=0;valid position for storing a new clusterin C*

Methods

*Cluster Set(**int** k)*

Input: number of clusters to be generated (k -means)

Output:

Behaviour: I create the array object referenced by C

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

Behaviour: Assign c to C[i] and increase i.

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

Behaviour: returns C[i]

***void** initializeCentroids(Data data)*

Behaviour:chooses centroids, createsa clusterfor each centroid and stores it 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 nearest Cluster (Tuple tuple)

Input: reference to a Tuple object

Output: cluster 'closest' to the passed tuple

Behaviour: Calculates the distance between the tuple referred by tuple and the centroid of each cluster in C and returns the closest cluster (make use of the getDistance() method of the Tuple class).

*Current Cluster Cluster (**int** id)*

Input: index of a row in the matrix in Data

Behaviour: Identifies and returns the cluster to which the tuple represents the example identified byid . If the tuple is not included in any clusterreturns null (make use of the contain() method of the Cluster class).

***void** updateCentroids(Data data)*

Behaviour: calculates the new centroid foreach clusterin C (make use of the computeCentroid() method of theCluster class)

public *String t oString()*

Input:

Output:

Behaviour: Returns a string made from each centroid of the cluster set .

public *String toString(Data data)*

Input:

Output:

Behaviour: Returns a string describing the state of each 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;
}
```

■ *Define the classKMeansMiner which includes the implementation of the kmeans algorithm*

Attributes

Cluster Set C;

Methods

*KmeansMiner(**int** k)*

Input: number of clusters to be generated

Behaviour: Creates the array object referenced by C

Cluster Set getC()

Behaviour: returns C

int kmeans(Date data)

Output: number of iterations performed te

Behaviour: Carries out t h e k-means algorithm by executing the pseudo -code steps:

1. *Random choice of centroids for kcluster s*
2. *Assignment of each row of the matrix atto the clusterhaving the closest centroid to the example.*
3. *Calculation of new centre idi for each cluster*
4. *Repeat steps 2 and 3. until two consecutive iterations return equal centroids .*

```
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;
                  //remove the tuple from the old cluster
                  if(currentChange && oldCluster!=null)
                        //the node is to be removed from its old cluster
                        oldCluster.removeTuple(i);

            }
            //STEP 3
```

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

- *Import the classMainTest class into the project. A possible example odi execution is given below:*

```
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

Iteration Number: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=0.666666666666
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
```