STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **L**, Number 1, 2005

A PROGRAMMING INTERFACE FOR NON-HIERACHICAL CLUSTERING

GABRIELA ŞERBAN

Abstract. Clustering is one of the important techniques in Data Mining. Clustering methods aim at grouping objects into clusters, so that the objects within a cluster are more similar to each other than objects in different clusters. The similarity between two objects is defined by a distance function. Clustering techniques are used in a variety of domains like: Natural Language Processing, Databases, HealthCare. In this paper we present a new programming interface for non-hierarchical clustering. Using this interface, we can simply develop non-hierarchical clustering applications. Using the designed interface, we made an experiment for words clustering, using a Romanian corpus.

Keywords: Clustering, Programming, Interface.

1. Introduction

The purpose of this paper is to present a standard interface for programming clustering tasks. The interface is meant to facilitate the development of software for clustering in different domains. In particular, the interface should facilitate an approach in which objects to be clustered and attributes describing the objects can be designed and implemented separately and then interconnected relatively easily in a standard, uniform fashion.

The aim of the proposed approach is to abstract the clustering issue, assuring a general approach, independent of the concrete representations of the entities involved in the clustering process.

In various domains clearly appears the necessity of clustering different kind of objects, with respect to different kind of attributes.

For example, in the field of Natural Language Processing we often need to group words by the similarity of their meanings, using a given corpus of words. In this

Received by the editors: March 20, 2005.

2000 *Mathematics Subject Classification.* 68N19, 62H30.

1998 *CR Categories and Descriptors.* I.5.3[**Computing Methodologies**]: Pattern Recognition – *Clustering*; D.1.5[**Software**]: Programming Techniques – *Object-Oriented Programming*;

example, the objects to be clustered are words and the attributes characterizing the objects are also words (from the corpus) [2].

Another example in which clustering is needed is HealthCare. HealthCare rises the problem of grouping patients in classes (clusters) with respect to the values of a number of symptoms for a given disease. In this kind of problems, the objects to be clustered are patients, and the attributes are symptoms [9].

There are many other domains in which clustering is needed, but for different kinds of objects and attributes.

That is why, in this paper we propose an unitary approach for all the clustering applications, independent of the type of objects to be clustered and the type of attributes characterizing the clustering process.

2. Clustering

As it is well-known, clustering is a partition of data into groups of similar objects.

Let us consider the following issue: given n objects $O_1, O_2, ..., O_n$, and m attributes $A_1, A_2, ...A_m$ (a set of relevant characteristics of the analyzed objects) ,we intend to group the objects in a given number k of *clusters*, so that the objects within a cluster are more similar (related to the given attributes) to each other than objects in different clusters.

For computing the similarities between objects we use the vector-space model, which means that the vector $\vec{O_i} = ({O_i}^1, {O_i}^2, ..., {O_i}^m)$ is associated with an object O_i as following: ${O_i}^j$ is a real number that gives a classification of the object O_i from the point of view of attribute A_j.

For designing our interface, the computation method of ${O_i}^j$ is unimportant.

Similarity and dissimilarity between objects are calculated using metric or semi-metric functions, applied to the attribute values characterizing the objects.

There are several methods for computing the similarity between two objects represented by their associated vectors (as defined above).

(1) The similarity measure between two objects O_a and O_b is defined as the *normalised cosine* between the vectors $\vec{O_a}$ and $\vec{O_b}$ [7]:

$$sim(\vec{O_a}, \vec{O_b}) = cos(\vec{O_a}, \vec{O_b}) = \frac{\sum_{j=1}^{m} O_a^j \times O_b^j}{\sqrt{\sum_{j=1}^{m} O_a^{j^2}} \times \sqrt{\sum_{j=1}^{m} O_b^{j^2}}}.$$

(2) the similarity measure between two objects O_a and O_b is defined as

$$sim(\vec{O_a}, \vec{O_b}) = \frac{1}{\sum_{j=1}^{m} (O_a^j - O_b^j)^2}.$$

(3) the similarity measure between two objects O_a and O_b is defined as

$$sim(\vec{O_a}, \vec{O_b}) = \frac{1}{\sum_{j=1}^{m} |O_a^j - O_b^j|}.$$

The *distance* between two objects O_a and O_b is defined as

$$d(\vec{O_a}, \vec{O_b}) = \frac{1}{sim(\vec{O_a}, \vec{O_b})}.$$

A well-known class of clustering methods is the one of the partitioning methods, with representatives such as the *k-means* algorithm. Essentially, given a set of n objects and a number $k, k \leq n$, such a method divides the object set into k distinct and non-empty partitions. The partitioning process is iterative and heuristic; it stops when a “good” partitioning is achieved. A partitioning is “good”, as we said, when the intra-cluster similarities are high and inter-cluster similarities are low.

We give next the non-hierarchical clustering algorithm (k-means algorithm) [8].

```
Algorithm k-means is
Input: - The set $X = \{\vec{O_1}, \vec{O_2}, \cdots, \vec{O_n}\}$ of n vector objects to be
         clusterised,
       - the distance measure $d : R^m \times R^m \to R$, between objects in a
         multi-dimensional space,
       - k, the number of desired clusters,
       - a function for computing the mean of a cluster $C$, $\mu : C \to$
         $R$,
       - the coefficient $\sigma$ (the threshold).
Output: - the set of clusters $C = \{C_1, C_2, \cdots, C_k\}$.
Begin
   Select $k$ initial centroids $\{\vec{f_1}, \vec{f_2}, \cdots, \vec{f_k}\}$
   While the diameter of a cluster $\geq \sigma$ do
      For all clusters $C_j \in C$ do
         $C_j = \{\vec{O_i} \mid \forall \vec{f_l}\ d(\vec{O_i}, \vec{f_j}) \leq d(\vec{O_i}, \vec{f_l})\}$
      EndFor
      For all clusters $C_j \in C$ do
         $\vec{f_j} = \vec{\mu}(C_j)$
      EndFor
   EndWhile
End.
```

As distance measure we considered:

$$d(\vec{O_a}, \vec{O_b}) = \frac{1}{sim(\vec{O_a}, \vec{O_b})}$$

and as centroid the mean of the cluster:

$$\vec{\mu}(C_j) = \frac{1}{\mid C_j \mid} \sum_{\vec{O} \in C_j} \vec{)}$$

We define the diameter of a cluster as *the distance between the least similar elements in a cluster.*

We also mention that the algorithm stops when the diameter of each cluster is less then a fixed threshold.

3. The programming interface

In this section we propose a standard interface that allows a simple development of clustering applications, providing a uniform development for all kind of applications.

The programming interface provides a hierarchy of classes and interfaces that can be used in all clustering applications. The clustering mechanism will be the same for all types of objects and attributes.

The interface is realized in JDK 1.5, and is meant to facilitate software development for non-hierarchical clustering.

There are three basic entities (objects): *objects to be clustered*, *attributes* (that characterize the objects) and *clustering.*

For designing the interface, we made an abstraction of the clustering mechanism, in order to be used for any kind of data (objects and attributes). Much more, the objects to be clustered are completely separated from the attributes that characterize them (an object has to know nothing about an attribute). Thus, we can easily change the attributes characterizing the objects, without affecting the clustering process.

The *clustering* object is the main object of the interface that manages the clusterization of the given objects related to the given attributes. The *clustering* object provides the behavior specific to the clustering process.

A main characteristic is that the *clustering* object is completely separated from the objects to be clustered and the attributes characterizing the objects (the *clustering* object knows only the behavior provided by the methods from the interface of these entities).

The interface provides an object as a manager for the list of objects to be clustered that manages the creation of the list of objects, attributes and centroids from an external device (file, database).

For using the interface, the user has to define the specialized object classes `CConcreteObject` (the concrete object to be clustered), `CConcreteAttribute` (the concrete attribute) and `CConcreteManager` (the concrete manager for the list of objects and attributes), by creating instances for each. The list of objects, attributes and centroids given by the manager are then passed to a clustering object (`CClustering`), that will initialize the clustering process and will manage the results.

In the following we present the skeleton of a clustering application.

(1) First, the user defines the class corresponding to the concrete object to be clustered.

```
public class CConcreteObject extends CObjectToBeClusterized
{...}
```

(2) Second, the user defines the class corresponding to the concrete attribute that characterizes the objects.

```
public class CConcreteAttribute implements IAttribute
{...}
```

(3) The user defines the class corresponding to the concrete manager of objects to be clustered and attributes.

```
public class CConcreteManager implements IClusteringManager
{
public CListOfObjectsToBeClustered createListOfObjects(){...}
public CListOfAttributes createListOfAttributes(){...}
public CCluster createCentroids(){...}
{...} }
```

The application class which initializes the clusterization with the data provided by the concrete manager object is always the same (remains unchanged for all non-hierarchical clustering issues), and is described below.

```
   class Application {
   private Application(){
          CConcreteManager cm=new CConcreteManager();
   //the manager provides the list of objects to be clusterized
CListOfOjectsToBeClustered l=cm.createListOfObjects();

   //the manager provides the list of attributes corresponding to the objects
CListOfAttributes y=cm.createListOfAttributes();

   //the manager provides the initial centroids of the clusters
CCluster f=cm.createCentroids();

   //the manager initializes the clustering process
CClustering l=new CClustering(l, y, f); }
   public static void main(String args[]){
          Application apl=new Application();

   }}
```

Figure 1 shows a simplified UML diagram of the interface, illustrating the hierarchy of classes. It is important to mention that all the classes provided by the interface, except the concrete classes, remain unchanged for all kinds of clustering applications.

4. The Design of the Interface

The classes used for realizing the interface are the following:

- `IList` **INTERFACE**
  Defines the structure of a list of objects, having operations for managing the list: adding an element on a given position, removing an element from a given position, returning the number of elements from the list, returning an element from a given position.
- `IClusteringManager` **INTERFACE**
  Defines the structure of a manager for the clustering process. The manager provides methods for obtaining the elements that are needed in the clustering process: creating the list of objects, the list of attributes and the initial centroids.
- `CLine`
  Defines the structure for the vector $\vec{O}$ corresponding for an object O (as we had defined above). An element of this vector is a real number, representing a characteristic measure for the object related to an attribute (in this class, the type of an attribute is unimportant). The main methods of this class are for: adding, updating, removing elements, for calculating the similarity between two lines.
- `CCluster`
  In our design a cluster is represented as a list of `CLine`(a line identifies in fact an object). For a cluster, the type of the object is unimportant. The main methods of this class are for: adding, updating, removing elements, for calculating the centroid of a cluster, for testing the equality of two clusters.

OBJECTS

An object is the entity to be clustered.

- `CObjectToBeClustered` **ABSTRACT CLASS**
  Is the basic class for all the objects. The specific objects will be instances of subclasses derived from `CObjectToBeClustered`. An object is identified by its vector representation. The methods of this class

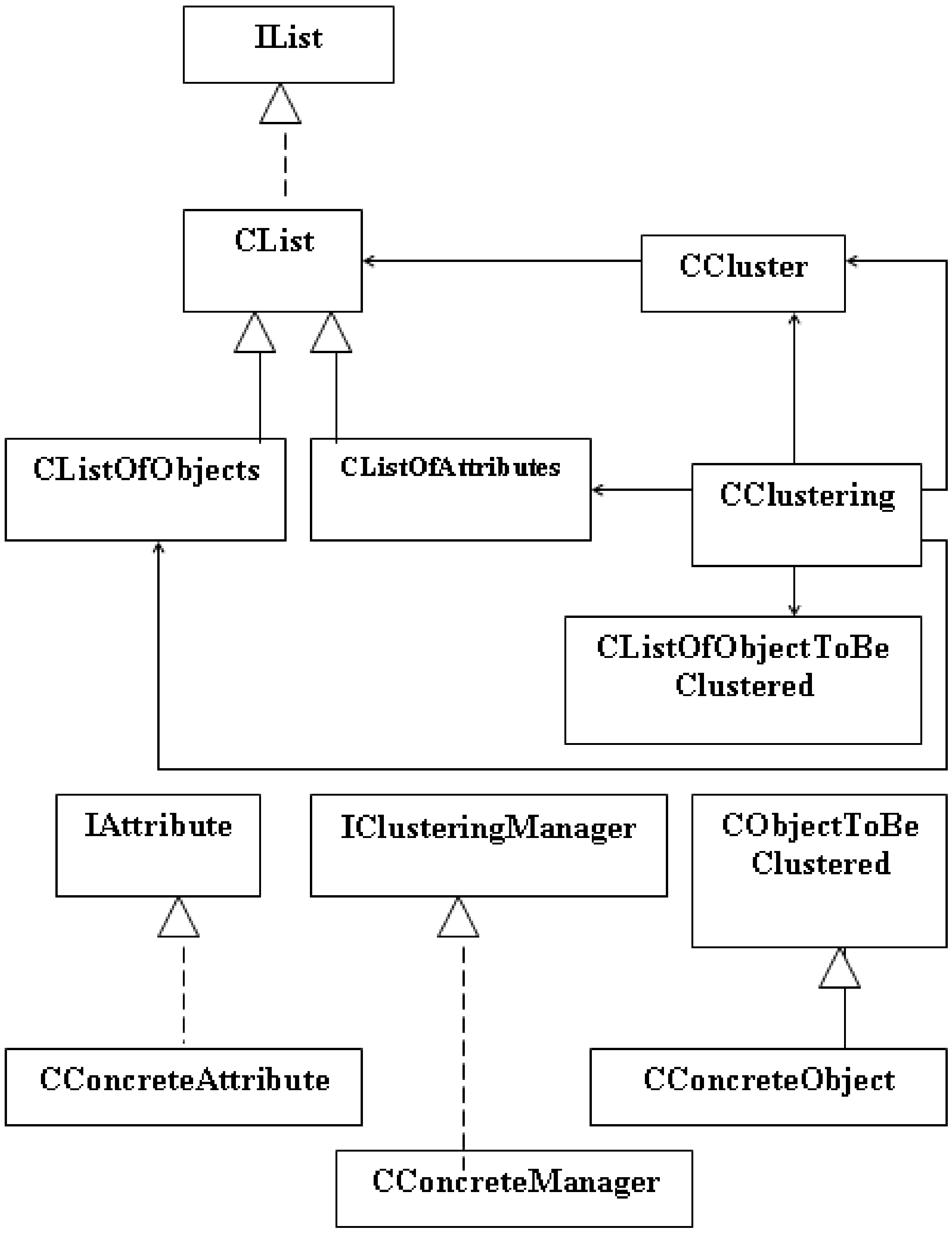

FIGURE 1. The diagram of the programming interface

are for: returning a String with the representation of an object, returning the value of an object, comparing two objects, returning the vector corresponding to an object.

- `CListOfObjects` **ABSTRACT CLASS**

  This class represents the list of objects to be clustered. The main methods are for managing the list.
- `CListOfObjectsToBeClustered` **ABSTRACT CLASS**

  This class maintains a list of objects `CListOfObjects` and the list of vector representations corresponding to each object from the list. The main methods are for managing the components.

ATTRIBUTES

- `IAttribute` **INTERFACE**

  Defines the structure of an attribute characterizing an object to be clustered. The methods of this class are for: returning a String with the representation of an attribute, returning the value of an attribute, comparing two attributes.

CLUSTERING

- `CClustering`

  Is the basic object of the interface, that manages the clustering process of the objects (related to the attributes). Defines the *heart* of the interface, the uniform usage that all objects and attributes are meant to conform to.

  An instance of the clustering class is associated with an instance of a list of objects and a list of attributes at the creation moment. This is made in the constructor of the class `CClustering`. The main method of this class is the method that manages the clusterization process (using the non-hierarchical clustering algorithm) and returns the clusters obtained.

```
    public class CClustering
{
    private CListOfAttributes y; //reference to the list of attributes
    private CListOfObjectsToBeClusterized l; //reference to the list of
objects to be clustered
    private CCluster f; //reference to the initial centroids
    ...
}
```

5. Experiments

In order to test the above defined interface, we considered a NLP experiment for the Romanian language. The aim was to clusterize a set of words (to group the words after the similarity of their meanings).

In our experiment, the attributes for the words to be clusterised were also words. Using a corpus for Romanian language, the vector words are computed based on the idea described in [2].

For testing the generality of our interface, we have also developed a clustering application for HealthCare. We mention that all data were taken from the website at [10].

The objects to be clusterized in this experiment were patients: each patient is identified by 9 attributes [9]. The attributes have been used to represent instances. Each instance has one of two possible classes: benign or malignant.

For each experiment, we have defined the classes that provide the current clustering focus, `CConcreteManager`, `CConcreteAttribute` and `CConcreteObject` and the applications for clustering were easily developed.

As a conclusion of our experiments, we have to mention, from a programmer point of view, the advantages of using the above proposed interface:

- is very simple to use;
- the effort for developing a clustering application is reduced – we need to define only three classes, the rest is provided by the interface;
- the user of the interface has to know nothing about the method for clustering the objects, because is provided by the interface;
- we can dynamically change the type of objects to be clustered or the type of the attributes that characterize the objects, and the interface remains unchanged.

6. Conclusions and Further Work

As a conclusion, we have developed a small framework that will help programmers to build, dynamically, their own clustering applications without dealing with the clustering mechanism (that remains unchanged and is provided by the interface). For a concrete application, the programmer has only to create three classes (derived from the classes defined by the interface): a class corresponding to the object to be clustered, a class corresponding to the attribute characterizing an object and, finally, a class corresponding to the entity that manages the objects and attributes.

After defining the concrete classes, the clustering will be made by creating an instance of the class `CClustering` provided by the interface. So, the programmer's effort for developing an application is small.

We mention that using the proposed interface we can simply develop clustering applications for different kind of data (objects). The objects can be words (in

NLP), databases, even patients (in HealthCare), or any other objects for which clustering techniques can be applied.

Further works can be done in the following directions:

- how can the interface be generalized in order to be used both for hierarchical [2] and non-hierarchical clustering;
- how can the interface be generalized for adaptive clustering (there are new Objects to be clustered or/and new Attributes that characterize the already clustered Objects).

References

[1] Jain, A., Dubes, R, "Algorithms for Clustering Data", Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[2] Tatar, D., Serban, G.: "Words Clustering in Question Answering Systems", Studia Universitatis "Babes-Bolyai", Informatica, XLVIII(1), 2003, pp.23–32.

[3] I. Dagan, L. Lee, F. C. N. Pereira: "Similarity-based models of Word Coocurences Probabilities", Machine Learning Journal 34(1–3), 1999, pp.1–26.

[4] C. Orasan, D. Tatar, G. Serban, D. Avram, A. Onet: "How to build a QA system in your back-garden: application to Romanian", EACL '03, Budapest, April 2003, 12-14, pp.139–142.

[5] P. Resnik: "Semantic Similarity in a Taxonomy: An information-Based Measure and its Application to Problems of Ambiguity in Natural language", Journal of AI Research, 1998, Center for the Study of Language and Information (CSLI), pp.1–28 .

[6] G. Serban, D. Tatar, "Word Sense Disambiguation for Untagged Corpus: Application to Romanian Language", Proceedings of CICLing 2003 (Intelligent Text Processing and Computational Linguistics), Mexico City, Mexic, Lecture Notes in Computer Science N 2588, Springer-Verlag, 2003, pp.270-275.

[7] D. Jurafsky, J. Martin: "Speech and language processing", Prentice Hall, 2000.

[8] C. Manning, H. Schutze: "Foundation of statistical natural language processing", MIT, 1999.

[9] Wolberg, W., Mangasarian, O.L.: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193–9196.

[10] http://www.cormactech.com/neunet, "Discover the Patterns in Your Data", CorMac Technologies Inc, Canada.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania

E-mail address: gabis@cs.ubbcluj.ro