Universitat Politècnica de València

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Master in Control and Industrial Informatics
Màster en Automàtica i Informàtica Industrial

Master Dissertation

New methods for discovering local behaviour in mixed databases

Author: Fátima Barceló Rico

Directors: José Luis Díez Ruano
Jorge Bondía Company

Department of Control and Systems Engineering
Departament d'Enginyeria de Sistemes i Automàtica

Valencia, July 20, 2009

Contents

Introduction

Nowadays, the number of data sets collected in numerous applications and with different purposes is quite large, making it very interesting to find a way of working with such a large number of data and find knowledge inherent to the database, whether it can be first sighted or not. This makes the *Knowledge Discovery in Databases* an increasingly important research area in engineering, being the *clustering* techniques among the most useful and applied mechanisms to find this embedded information most of the times hidden from first or *simple looks*. The aim of these techniques is to group data into different sets, called clusters, in which the data belonging to each one will be more similar to the elements of the same cluster than to elements of other clusters. A large list of applications can be written for the *KDD* techniques, like pattern finding, bio-informatics, modeling, data mining, etc, being clustering a method used widely to provide a way to interpret the set always for the particular application in which it is being analyzed.

Given that the applications are numerous and differ from one to another very much, the data bases to which the techniques are to be applied comprehend a wide range of cases. The complexity of the applications also requires them to be quite complete and this usually means that they will extend to a large number of attributes. These attributes can correspond either to numerical characteristics and then they are represented by a number or to categorical characteristics and then they are represented by a symbol or category. Therefore, data sets can be divided, when attending to their attributes, into three groups, when all attributes are numerical (numerical data sets), when all attributes are categorical (categorical data sets) and finally, when both types of attributes are present (mixed data sets). Several more subdivisions can be made if the applied criteria change, but all of them will be thoroughly described in a later chapter.

It is quite important to mark the differences between a database that contains some information about the output, i.e. desired classification, and the data set which does not contain any. In the first case, it will be said that the clustering is *supervised* while in the second one the clustering will be referred to as *unsupervised*. Sometimes, if information about the outputs is available then information about the number of clusters can be known too and that makes most of the algorithms perform in a much better way. However, when no information is available or included, finding patterns or the proper

clusters is always much more complicated than in the supervised case, as there is no knowledge about the number of clusters. Obviously, the ideal way of working is with information about the output, but most of the times this is either not available or very expensive to get, and then *unsupervised* clustering is the chosen option.

Before, it was mentioned that the elements in the same cluster were more similar among themselves than to elements in different clusters. This involves that a similarity measure has to be defined, according to which the elements will be clustered. This similarity measure can be easily defined as a distance metric in the case of numeric attributes, but it is not so easy to define it when the attributes are categorical or mixed. This measure is the base of all clustering algorithms, as it will induce the assignment of one element to a particular cluster or to a complete different one, so it is quite important to define it accurately according to the purposes of the application.

Ideally, the perfect clustering algorithm should comprehend the most general characteristics of all clustering algorithms, this means that this algorithm should work well either in supervised or unsupervised cases and either with numerical data or categorical or even mixed. However, the truth is that if the applications are very complex it is almost impossible to design an algorithm that works well in many cases.

In this work, a *methodology* has been proposed which performs well in a good number of applications, but always when the applications have modeling purposes. This last point implies that any application with a different goal might not be very well covered by the algorithm here proposed, given that the focus of it, as already mentioned, is to find a model which describes the behaviour of the data set to cluster.

To offer a better understanding of the techniques explained and the one proposed, they have been applied to a few databases, some of them very well know in literature and others not public but collected for testing the proposed algorithm. The criteria to select the data sets has been to try to show the performance of this algorithm and compare it with others.

Then the layout of this work is the following: in the next chapter there is a more detailed introduction to the problem treated through this document and a description of many topics related to it and necessary to understand the later explanations. In Chapter 2 it has been included a brief classification of the kinds of clustering algorithms and a review of the state of art in most of the described types. Chapter 3 presents an introduction to one of the problems treated here, treatment of the symbolic variables, and the review of how it is the state of the approaches found in literature up to now. In Chapter 4 is described the proposed approach to overcome the problems of treating categorical values. Later, in Chapter 5 the index built to perform the modeling of the data set is detailed, including examples where it has been applied. Finally, in Chapter 6 the future works are described and how the already described techniques are expected to be used.

1. Preliminaries

1.1 Introduction

Data mining is the name given to that group of techniques that work with sets of data and try to get some knowledge embedded within them and hidden from simple looks. *Clustering* is one of these techniques and one of the most popular and used ones. Clustering refers to the division of data into groups of similar objects, where each is group called *cluster*. Each cluster consists of objects that are similar to the objects in the same clusters and dissimilar to objects in other clusters. Clustering is widely used in several disciplines, such as statistics, biology, software engineering, web applications, text mining and so on. The data sets are constantly becoming larger and this high dimensionality prevents easy analysis or validation without using a computerized technique. It is important to note that when representing a large database with a small number of clusters we achieve some simplification at the price of losing some data details. However, this simplification will facilitate later work with the data set. Therefore, as far as the committed error can be tolerated, this simplification will be good for the posteriori work on the data set.

In this chapter, many things related to the clustering process will be described, like types of measures and some preprocessing steps applied to the objects.

1.1.1 Clustering Steps

Given that clustering techniques have to determine which objects are *similar* to which others, this suggest that clustering is an iterative process [4]. The following steps describe the process of applying this technique:

- ◇ **Data Collection**: Refers to the carefully extraction of relevant data objects from the underlaying data sources. Here, data objects are distinguished by their individual values for a set of parameters considered, called *attributes*.

- ◇ **Initial Screening**: Refers to the first treatment of the data after its extraction from the source, in order to see whether the data collected is suitable for clustering.
- ◇ **Representation**: Includes the proper preparation of the data in order to become suitable for the clustering algorithm. Here is where the similarity measure is chosen (or defined) and the characteristics, including the dimensionality of the data, are examined.
- ◇ **Clustering Tendency**: Checks whether the data in hand has a natural tendency to cluster or not. Most of the times this step is ignored.
- ◇ **Clustering Strategy**: Involves the careful choice of the clustering algorithm and initial parameters and its immediate application.
- ◇ **Validation**: This is one of the last steps. It is often based on manual examination and visual techniques. However, as the amount of data and its dimensionality grow, the manual and visual techniques become useless as designers do not have means to compare it with predefined ideas.
- ◇ **Interpretation**: The idea is to draw conclusions from the clustering results. Therefore, it is highly desired to have an algorithm that provides interpretable results that can be used for conclusions or further analysis.

1.1.2 Notation

In order to clarify those terms mentioned before as *object*, *attribute*, etc. and used all along this document we will offer a brief explanation on notation. Consider a data set $X = \{x_1, x_2, ..., x_N\}$ consisting of N data points which will be referred to as *objects* and may represent people, things, transactions, etc. Each object can be represented in the following way $x_i = (x_{i1}, ..., x_{iz})$, where each component from 1 to z will represent an attribute from the attribute space A, $x_{il} \in A_l$, $l = 1 \dots z$. Attributes may represent characteristics, variable, dimension, field, etc. This *object by attribute* data format corresponds to an $N \times z$ matrix and is used by most of the clustering algorithms.

All the clusters will be mainly defined by their prototypes or also called *centers*. To refer to them c_i will be used, where each of the centers will have z coordinates (as many as the column dimension of the input matrix).

1.1.3 Important Properties

The different classifications we can find for clustering and the most important algorithms that can be found in each of them will be described in later sections and chapters.

Nevertheless, no matter in which category the clustering algorithm lies, a good one must have most of this properties:

- ◇ Scalability to large data sets, i.e. a large quantity of objets.
- ◇ Ability to work with high-dimensional data, i.e. many attributes by object.
- ◇ Ability to work with all type of attributes.
- ◇ Ability to find clusters of irregular shape.
- ◇ Handling outliers.
- ◇ Low time complexity.
- ◇ Data order independency.
- ◇ Interpretability of results

However, it would not be realistic to expect any algorithm to perform like that. In a more realistic way, algorithms as will be shown later, only meet some of these properties. Therefore, it will be the designer who finally has to prioritize these desirable characteristics and select the ones that are important for the particular application the algorithm is being designed for, and then, design the algorithm based on them.

1.2 Similarity/Dissimilarity Measures

All clustering algorithms are based on the similarity/dissimilarity the objects of the database have among themselves. However, there does not exist an inherent way of measuring this, and that is why similarity/dissimilarity measures have to be defined [28]. These measures try to quantify how similar two elements are and they do it looking and comparing the attributes that define the two involved objects. Therefore, it is advisable to look first to the attributes and the types they can be, to define later the similarity/dissimilarity measure based on them.

1.2.1 Data Types: Numerical and Categorical

All the objects within the database are represented by their attributes which represent the characteristics or features either considered relevant or that were available through the data collection phase to do the clustering of these objects. These characteristics are

the key factors of the problem and their choice and type influences the results of the clustering algorithm in a highly considerable way.

In such way, two types of data can be differentiated: *numerical* and *categorical*, [28]. **Numerical** refers to when the attribute can be fully represented by a number. This leads to another subdivision: *continuous* and *discrete*. An attribute is continuous if its domain is uncountably infinite. This means that between any two values of the attributes there exists an infinite number of values. Examples of this can be *temperature*, *length*, etc. On the other hand, a *discrete* attribute is so if its domain corresponds to a finite set. This means that such attributes can adopt only some of the values of the interval and between any two values the number of values is finite. Examples of this can be *number of children*, *age*, etc. Numerical attributes have the inner property of being ordered by its magnitude, whether they are discrete or continuous.

Categorical attributes are those that can not be fully represented by a number, but they have to be represented by a category. These categories are most of the times nominal labels which are assigned to the attribute to define the particular characteristics it has. One of the main characteristics is that these categories can not be totally ordered, meaning this that there is not an absolute way of ordering them because these categories are independent from one another. Examples of this type are: *city where people live*, *favorite food*, etc.

Gradable attributes are a particular type of categorical attributes. Each category is a grade: bad, regular, good, etc. Although they are categorical, the different categories are dependant and are ordered. This means that, in this point, they are more similar to numerical attributes than to categorical and their treatment could be done as if they were discrete numerical values.

Attributes that are numerical but divided into intervals are also an exception, given that they correspond neither to continuous numbers nor to discrete number, but to a group of numbers which can be considered as categories which are dependant and can be ordered. Their treatment can be done in a similar way to gradable attributes, because the numerical approach is more similar to their real behaviour than the categorical approach.

One particular case is that of *binary* attributes, i.e., when they are discrete, either numerical or categorical, and can adopt only a pair of values. There a multiple examples of this case, like: *Yes/No*, *Female/Male*, *1/0*, etc. This type of attributes are more similar to categorical than to numeric, meaning presence/no presence of something, which is a symbolic representation.

1.2.2 Metrics and measures properties

Once the characteristics of the data have been determined, the problem of finding the proper way to measure how far or close the data objects are from each other is faced. As mentioned before, this can be done defining a *similarity/dissimilariy* measure [5]: the more two objects resemble each other the larger their similarity is and the smaller their dissimilarity. The most common way of measuring this is using a distance measure. It is important to note that this can be measured in different ways and it will be necessary to define it. It has already been said that all measures depend on the attribute type, therefore it will not be possible for categorical values to use geometry-based distance measures, as such data has no such orientation.

When the attributes are numerical, the most used distance measure is that called *metric*, although here it will be referred to as ***measure*** to maintain as much generality as possible. Given three objects x, y and z a distance metric d should satisfy the following properties:

1. $d(x, y) \geq 0$: non-negative value;
2. $d(x, y) = 0$ **if** and **only if** $x = y$: distance of any object to itself is 0;
3. $d(x, y) = d(y, x)$: symmetry;
4. $d(x, z) \geq d(x, y) + d(y, z)$: triangle inequality.

Here, it is presented a description of the most general and commonly used distance measures for each type of attribute.

1.2.3 Numerical measures

The most common similarity measures used for this type of attributes are the ones based in the Minkowsky Distance, also called the q-norm of the difference vector:

$$d(x, y) = \|x - y\|_q \tag{1.1}$$

◇ *Minkowski Distance* is another expression for (1.1), and is defined as:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^q \right)^{1/q} \tag{1.2}$$

where q is a positive integer ($q \geq 1$).

◇ *Euclidean Distance* is defined as:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1.3}$$

Note that this equation is a particularization of (1.2) with $q = 2$.

◇ *Manhattan Distance* is defined as:

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{1.4}$$

Note that this equation is a particularization of (1.2) with $q = 1$.

◇ *Maximum Distance* is defined as:

$$d(x,y) = \max_{i=1...n} |x_i - y_i| \tag{1.5}$$

Note that this equation is a particularization of (1.2) when $q \to \infty$.

◇ *Mahalanobis Distance* is defined as:

$$d(x_j, c_i) = (x_j - c_i)^T \Sigma_i^{-1} (x_j - c_i) \tag{1.6}$$

where Σ_i is the covariance matrix of the cluster. Note that this equation is like Euclidean distance, but the covariances of the objects have been included or, what is the same, the Euclidean distance corresponds to the Mahalanobis distance but with covariance=I (identity matrix).

Although the most common distance measures are these ones, they are not the only ones for numerical attributes. For example, it can also be found the *sample correlation coefficient* in some applications as similarity measure for quantitative (numerical) data. The sample correlation coefficient is a similarity measure used to quantify the lineal dependency between two characteristics or dimensions. Nevertheless, it is not widely used for clustering purposes.

Sometimes it is desirable or appropriate to use a more complex distance as it can be the one proposed by **Gowda and Diday** (1.10) [11]. This a *dissimilarity* measure that defines three different measures for two variables, A_k and B_k: *position*, *span* and *content*. Adding these three variables we will get the final distance. Some parameters have to be defined:

al = lower limit of A_k;
bl = lower limit of B_k;
$a\mu$ = upper limit of A_k;
$b\mu$ = upper limit of B_k;
$inters$ = length of the intersection between A_k and B_k;
$l_s = span = |\max(a\mu, b\mu) - \min(al, bl)|$;
U_k = difference between the larger and the smaller value of the characteristic k^{th} for all the objects;
$l_a = |a\mu - al|$;
$l_b = |b\mu - bl|$;

With those parameters the three necessary variables can be defined as shown in (1.7) for the position d_p, in (1.8) for the span d_s and in (1.9) for the content d_c.

$$d_p(A_k, B_k) = \frac{|al - bl|}{U_k} \tag{1.7}$$

$$d_s(A_k, B_k) = \frac{|l_a - l_b|}{l_s} \tag{1.8}$$

$$d_c(A_k, B_k) = \frac{|l_a + l_b - 2 \cdot inters|}{l_s} \tag{1.9}$$

And finally, adding the three equations, we will get the final distance:

$$D(A_k, B_k) = d_p + d_s + d_c \tag{1.10}$$

The description of the simlarity/dissimilarity measures might never end, because the distance chosen will depend on the application and its requirements. When there is some freedom to choose it, the simplest ones are the most used, given that the computational cost is lower and the interpretability is higher. However, it is not always possible to keep it simple and achieve the purposes of the application. Then complex measures as the last described one are needed to perform in a better way the clustering process.

1.2.4 Binary measures

For this type of attributes the matching coefficients can be defined, which are based on the number of common elements between the two objects that are being compared. As they

are binary attributes their characteristics are only defined by two values. Let's use 1 to describe one of the options and 0 to describe the other one whichever their real categories are, in other to simplify the description of the measures. Now four parameters are defined to express matching and not matching:

- α is the number of attribute values that are equal to 1 in both objects.
- β is the number of attribute values that are equal to 1 in x but 0 in y.
- γ is the number of attribute values that are equal to 0 in x but 1 in y.
- δ is the number of attribute values that are equal to 0 in both objects.
- τ is the total number of attributes and then, $\tau = \alpha + \beta + \gamma + \delta$

From this definition we can express the two most used distance measures for binary attributes:

- *Simple Matching Coefficient* is defined as:

$$d(x, y) = \frac{\alpha + \delta}{\tau} \tag{1.11}$$

  This index accounts for the number of attributes that are equal in both objects. This coefficient is used in most of the situations when there are binary attributes.

- *Jaccard Coefficient* is defined as:

$$d(x, y) = \frac{\alpha}{\alpha + \beta + \gamma} \tag{1.12}$$

  This index disregards the number of 0-0 matches. This is used when the state described as 1 has a higher weight or influence than the other (described as 0). This one is not commonly used and only applied in singular cases.

1.2.5 Categorical measures

Although in the case of numerical and binary attributes the distance measures are very well-known and widely used, in the case of categorical attributes there is no a distance measure that has become known for its great performance and variety of possible applications. The most common one might be the one:

$$d(x,y) = \sum_{i=1}^{z} \delta_{xy}^{i} \tag{1.13}$$

where for each attribute i in x and y, the parameter δ_{xy}^{i} is defined as:

$$\delta_{xy}^{i} = \begin{cases} 0 & if \quad x_i = y_i \\ 1 & if \quad x_i \neq y_i \end{cases} \tag{1.14}$$

In this way when both attributes are equal they do not affect to the total distance and when they are different they contribute by one unity. This means that (1.13) measures dissimilarity between a pair of objects.

This distance is easy to understand and apply, but sometimes it is required a more complex distance measure. The following one is based on the **Gowda and Diday** distance but adapted to categorical attributes. Now the variables defined are only two: *span* and *content*. Respectively they are:

$$D_s\left(A_k, B_k\right) = \frac{|l_a - l_b|}{l_s} \tag{1.15}$$

$$D_c\left(A_k, B_k\right) = \frac{|l_a + l_b - 2 \cdot inters|}{l_s} \tag{1.16}$$

The parameters to be defined in these equations are:

$inters$ = number of elements in the intersection of A_k and B_k;
l_s = number of elements in the union of A_k and B_k;
l_a = number of elements in A_k;
l_b = number of elements in B_k;

Some other distances can be found in the literature for categorical data. Among them we can find those based on probabilities such as the *entropy* measure which indicates the diversity or heterogeneity of a data set and also the *Kullback-Leibler* measure which indicates the divergency between two probability distributions. However, these measures are not as widely used as (1.13).

Alike the numerical case, the measure depends on the application and it will be defined for it, always trying to make it as general and simple as far as it meets the requirements.

1.3 Normalization

The attributes considered in the database can be of a very different nature, not only for the type of data they belong to, but also for the unities they are expressed in, their magnitudes and other characteristics. So, it is not the same to express a distance in meters or in kilometers and it is also different if an attribute has values around 0.1 and other around 1000. Therefore, to make all attributes have the same importance it is advisable to normalize all them before the clustering algorithm is applied [5], [29]. This will do that the units with which the attributes are measured do not affect, nor the magnitude of them.

This normalization could be done in different ways. One of them is converting the range of the variable into a pre-fixed interval of values, i.e $x_{norm} \in [-1, 1]$. Another way is to make the normalized variable to have some properties, like statistical properties, i.e. null mean and unitary variance. The most important property of normalization is that after it, all variables have to be equally important for the clustering algorithm.

Normalization step is essential for a proper clustering process, much more if the clustering algorithm will be a numerical algorithm, where the distances computed are numerical and so it is the cost function. Therefore, the non-normalized attributes can make the clustering process consider different importance for attributes that are not real. With the normalization, the units with which the attributes are measured do not affect to the clustering.

1.3.1 Normalization by interval

In this type of normalization, the attributes will be normalized so as to all them, after normalization, are defined in the same interval. This interval can be defined by the user and then all variables will be transformed, or by any of the variables and then the rest will be transformed. This last option is the most used one, because in that case the range of transformation is already defined by some variable or variables which will be considered the standard ones and will not need to be normalized.

This normalization is based on a linear transformation of the data in the defined range [y_{min}, y_{max}]. Figure (1.1), shows an example of this when the normalized range is between [-1,1], although it could be any other. It will have to be applied to each attribute independently of the rest, because the limits of definition for each one is usually different from the others. The equation for the new normalized values is:

$$y_{norm} = \frac{\Delta y}{\Delta x}(x - x_{\min}) + y_{\min} \tag{1.17}$$

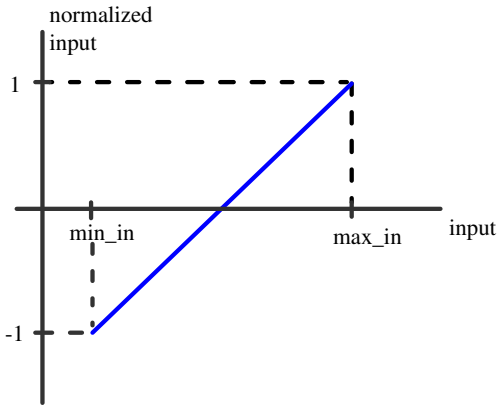

Figure 1.1: *General representation of the normalization of a numerical attribute*

In eq. (1.17), y refers to the normalized variable while x to the non-normalized one. Δy is the length of the interval for the normalized attribute and Δx the length of the interval of the attribute before normalization.

With this normalization all attributes will be within the same interval and will have same importance in the clustering process.

1.3.2 Normalization by properties

In this other type of normalization, the variables will be normalized in order to meet some properties equal for all of them. The most common type is the normalization to have statistical properties in the variables, like null mean and unitary variance. In this way, the interval where each attribute lies will be different from the rest, but they all will follow a normal distribution, which, from the statistical point of view, makes them all equally balanced.

The equation to normalize the variable in this way is:

$$y_{norm2} = \frac{(x - \bar{x})}{\sigma_x} \tag{1.18}$$

where $\bar{x}$ is the mean of the non-normalized variable and σ_x the variance of this.

With this type of normalization all attributes will be, in the statistical sense, balanced and will have the same effects in the clustering process.

1.3.3 Denormalization

Normalization is extremely important, as it has already been mentioned, for the clustering process. If attributes are not normalized the clustering algorithms will give them different importance only depending on their magnitude and not in their real importance.

However, once the clustering is done and the results are obtained, it is important to remember that those results are in terms of normalized attributes, which means that are not valid for the non-normalized variables. It is necessary, therefore, to denormalize the parameters obtained from the algorithm, always using the same method used for the normalization. Then, all the parameters will be able to be used in the non-normalized data or new *raw* objects.

1.4 Weighting Factors

Now all attributes have the same importance and this is desirable when there is no knowledge about them or about the system. On the contrary, if there is some knowledge about the importance of some particular attributes it is desirable to add it to the clustering algorithm, not by means of the attribute magnitude but by means of a sort of importance factor introduced by the expert. This can be called *weighting factors*, that are factors introduced in the similarity measure to penalize the directions when we do not want large variations in the cluster objects or, what is the same, to give larger importance to those attributes that have more relevance in the clustering process. A weight will be given to each attribute, being the default weight 1 (no weighted). The introductions of weights in the case of the Euclidean distance expressed previously in (1.3) can be seen in:

$$d(x,y) = \sqrt{\sum_{i=1}^{z} \omega_i \cdot (x_i - y_i)^2} \tag{1.19}$$

While the introduction of weights in the Euclidean distance expressed in (1.13) is expressed by:

$$d(x,y) = \sum_{i=1}^{z} \omega_i \cdot \delta_{xy}^i \tag{1.20}$$

where ω_i refers to the weight of the attribute i and multiplies directly the amount that this attribute contributes to the global distance.

2. Clustering Algorithms

2.1 Basic Clustering Classifications

The classification of clustering algorithms is not straightforward as there exist different criteria. These criteria do not consider the same factors. In fact, the considered factors are completely different. This yields to an overlap between the several classes, being it possible that one algorithm belongs to two groups if the classification criteria is different in both. Currently, clustering techniques are divided into *hierarchical* and *partitional*. Hierarchical techniques are further divided into *agglomerative* and *divisive*. While hierarchical algorithms assemble (if they are agglomerative) or disassemble (if they are divisive) points into clusters, partitional algorithms learn clusters directly. Usually, this last type of algorithms tries to discover clusters by iteratively relocating points between clusters, but some times the technique used is different, like finding highly populated regions, dividing the space into segments or many others.

Another criteria to make this division can be the membership function of the objects to each cluster, which can be $exclusive$ or $non-exclusive$. Exclusive membership means that the objects belong or not to a particular cluster. This is the most traditional way of clustering attributes and many times it is denoted as *hard* clustering. However, if the membership is non-exclusive then the objects belong to more than one cluster with different membership degrees to each one. This is denoted as *fuzzy* clustering and it is quite interesting in the cases where the database is not very uniform. Another case is the *possibilistic* clustering, which is in many cases included in the fuzzy group. Here, it will be included too, but special attention will be given to it.

The last classification depends on the type of data the algorithm has been designed to work with. As the types of data have been defined in section 1.2.1 the classification of the algorithm according to this criteria is immediate. The algorithms can be: numerical, categorical or mixed. Last case is, obviously, when the algorithm works with both types of attributes both in similar or very different proportions.

In this chapter, the most important algorithms for the first and second classification will be described, leaving the third classification for the next chapter because it deserves special attention as it is part of the focus of this document to improve the current performance of mixed data algorithms.

2.2 Partitional and Hierarchical clustering

Before describing the most relevant algorithms within each of these groups, a more detailed and formal description of this classification is presented next.

Hierarchical clustering. As the same name indicates, these algorithms create a hierarchical decomposition of the objects. The agglomerative ones are also called *bottom-up* while the divisive ones are also known as *top-down*. Agglomerative algorithms start with each object being in a separate cluster itself and in each step they merge two or more groups according to a distance measure. These algorithms stop when they reach a prefixed stop condition or when all data form a single and unique cluster. On the other hand, divisive algorithms start with one group that contains all objets and progressively divides this one into two or several. This keeps going until a prefixed stop condition is reached or all objets are in different clusters (clusters contains only one object). The main problem with hierarchical algorithms is that once some clusters are merged or split they are not revised in future steps for improvement. Another drawback is that these algorithms have a high computational cost.

Partitional clustering. Given a set of data of n objects, partitional clustering conducts k partitions of the data, where each cluster optimizes a clustering criteria, such as minimizing the distance from each object to the center of the cluster. Therefore, the improvements are done locally and that makes them not good for many applications. These are high complexity algorithms, as some of them exhaustively and iteratively go over all objects to achieve the fixed goal. Most of the times they start with an initial configuration which will be refined. Usually, this initialization is random if no information is available a priori and this can lead to a local minima in some cases. This is one of the drawbacks of this type of algorithms, although generally, they work quite well.

2.2.1 Hierarchical Algorithms

Both agglomerative and divisive algorithms create different levels of clustering which can be represented by a tree, known as *dendrogram*. This allows exploring data with different levels of granularity. The advantages (A) and disadvantages (D) include:

- ◇ **A**: Flexibility regarding the level of granularity.
- ◇ **A**: Ease of handling any form of similarity measure.
- ◇ **A**: Application to any type of attribute.
- ◇ **D**: It is difficult to choose proper stop criteria.
- ◇ **D**: They do not revise clusters once done.

In hierarchical algorithms the distances between points are calculated using the selected distance measure. However, because some clusters are merged or split and these are subsets of points the same measure can not be used and therefore has to be adapted to sets of data to perform the merging or the splitting. Such derived measure is called *linkage metric* and it reflects concepts of *closeness* and *connectivity*. The linking metrics are applied over every pair of objects of the two clusters to be merged or of the same cluster to be split and a specific operation is applied, such as maximum, minimum or average. The methods using this intercluster distances are related to the connectivity graphs $G(X, E)$ where X, the vertices, are data points and E, the edges, are connections and weights.

The most used hierarchical clustering algorithms are: **BIRCH**, **CURE**, **CHAMELEON** and **COBWEB**, and most of them attempt to solve the problem of high computational cost.

- ◇ **BIRCH**: Balance Iterative Reduction and Clustering using Hierarchies [51].

  It is a hierarchical-divisive algorithm. This technique is based on the idea that we do not need to keep all the data base in main memory, but only enough statistical information about it. For each cluster the algorithm summarizes data by storing three values: n, number of points in that cluster, LS linear sum of the attribute values of the objects in that cluster and SS, the sum of squares of the attribute values. These three values are called Cluster Feature (**CF**) and with them it can be represented the CF-tree, where each node is a CF and represents a cluster. All the CF are kept in memory.

  CF-trees are characterized by two parameters: *the branching factor, B*, which is the number of children for a non-leaf node and the *threshold, T*, which is the maximum distance between any pair of points in a cluster. A new data point descends along the tree to the closest CF leaf. If it fits the leaf well, and if the leaf is not overcrowded, CF statistics are incremented for all nodes from the leaf to the root. Otherwise, a new CF is constructed. The resulting tree is balanced which is the main advantage of this algorithm and allows a efficient search.

BIRCH depends on the two described parameters to control the CF-tree construction and also data ordering. When the tree is constructed (first data pass), it can be condensed for an optional second data pass. The third phase of the algorithm includes a global clustering of CF. In the final optional fourth phase the irregularities that appear can be resolved.

The main advantage is that the algorithm only requires one go over the data. The disadvantage is that it only works well on well distributed numerical data. The order of the data base is important given the parameter T which limits the number of elements in the cluster by its distances. Finally, BIRCH has not been set in high dimensional data and then its performance in this respect is questionable.

◇ **CURE**: Clustering Using REpresentatives [41].

The most important characteristic of this agglomerative algorithm is that it represents a cluster by a fixed number, c, of points scattered around it. The distance between two clusters used in the merging process is the minimum of the distances between two representatives. Selecting representatives scattered around a cluster makes it possible to cover non-spherical shapes. CURE uses an additional step to make it non-dependant of outliers: original selected representatives are shrunk to the geometric centroid of the cluster by a parameter, α, specified by the user.

CURE uses two techniques to achieve scalability: data sampling and data partitioning. The algorithm creates randomly p partitions, and finds k clusters in each of the partitions, always starting from the whole subgroup and performing a divisive clustering. The computational complexity depends on the sampling size, the shrinking factor α (used to create new clusters), the number of representatives c, and the number of partitions p, so it is no direct to calculate its value, although it is high.

◇ **CHAMELEON**

This is another agglomerative algorithm [31] that uses the connectivity graph G corresponding to the K-nearest neighbors model: the edges of K most similar neighbours to any given point are kept, and the rest are cut. CHAMELEON has two stages. In the first one, small clusters are built and they will be the input to the second stage in which the agglomerative process is performed using measures of relative interconnectivity and relative closeness. Both measures are relatives because they are later normalized by the global cluster measure. Therefore the modeling is dynamic and depends on data locally. Merging steps depend on thresholds that have to be set by the user. It has been shown to find different shapes and sizes clusters.

◇ **COBWEB**

Although this algorithm [16] is a hierarchical one, it can be classified neither as agglomerative nor as divisive, because the applied technique is far away from these two. As it is a categorical algorithm, its explanation will be left for the following chapter where this type of algorithms will be described.

Other hierarchical algorithms are: CLASSIT [18], SLINK [43] and some others, but their implementations are not as widely used as the ones previously explained.

2.2.2 Partitional Algorithms

This family of algorithms includes the first ones that appeared in the Data Mining clustering applications. They divide data into k independents subsets, but since it is impossible to check all possible subsets due to computational cost, the improvement is done by performing a global **iterative** optimization. This makes reference to different relocations done iteratively to reassign points among the k clusters. Unlike hierarchical methods, the clusters are revised after they are built, improving them by relocation. Usually, these algorithms lead to high quality clusters. The optimization of these algorithms is done by the minimization of a global objective function.

The most used partitional algorithms are: **k-means**, **PAM** and **CLARA**. k-means works with the mean value of the elements belonging to each cluster, also known as **centroid**. On the other hand, PAM and CLARA belong to the so called **k-medoids** methods. In this type of methods a cluster is represented by one of its points. This allows the algorithm to cover all attribute types, which is a very good characteristic. Another advantage is that medoids are no sensitive to outliers, because these represent exceptions in the cluster. Once medoids are selected, clusters are defined as subsets of points close to their respective medoids. The objective function has to be a distance measure that can work with the points and the correspondent medoid. Here, there is a description of these three algorithms:

◇ **k-means**

This algorithm [21] is by far the most popular clustering tool used widely in the past and nowadays. The goal in k-means is to produce c clusters from a set of n objects, so that the objects of all clusters minimize the squared error objective function, i.e., the addition of all Euclidean distances from all the objects to the center of the cluster where they have been assigned to:

$$J_{index} = \sum_{i=1}^{c} \left(\sum_{k=1}^{n} d\left(x_k, c_i\right) \right) \tag{2.1}$$

where n is the number of objects, c the number of clusters, c_i the center of cluster i, $d(x, c_i)$ is the distance of the element x to the center of the cluster it has been assigned i.

This equation can be expanded as:

$$J_{index} = \sum_{i=1}^{c} \left(\sum_{k=1}^{n} \sqrt{\sum_{v=1}^{z} \left(x_{k,v} - c_{i,v}\right)^2} \right) \tag{2.2}$$

where now z is the number of attributes. So, the total distance will be the root square of the addition of the squares of the difference of each attribute.

The k-means algorithm has as input parameter c, the number of clusters, and as an output the algorithm returns the centers or means of all clusters c_i. The algorithm procedure is as follows:

1. Select c objects as initial centers,
2. Assign each data object to the closest center,
3. Recalculate the centers of each clusters,
4. Repeat steps 2 and 3 until centers do not change.

This algorithm works very well if and only if the parameter k chosen is the proper one. Otherwise, the results will not be interpretable and then this algorithm will lose its great advantage. The main disadvantage is that because the similarity measure is a distance metric, it can not be used for categorical attributes, only for numerical.

It also happens for the k-means algorithm that it is able to detect well distributed and spherical-shaped groups of data. When the clusters have different sizes or their shapes are not spherical-like, then this algorithm does not provide efficient results.

◇ **PAM**: Partitional Around Medoids.

This is an early k-medoids method. PAM [32] uses an iterative optimization that combines relocations of points between clusters with renominating the points as potential medoids. In PAM the objective function is similar to (2.1) but using the medoid instead of the mean of each cluster. It works well for small or not very large data sets. As all medoid-based algorithms it handles well outliers and extreme values.

◇ **CLARA**: Clustering LARge Applications [32].

This is another k-medoids methods which was born to overcome the computational cost issue existent when PAM is applied to large databases. CLARA uses several samples (five usually works fine), each with several points of the database, which are subjected to PAM. The whole database is assigned to resulting medoids, the objective function is computed and the best system of medoids is retained. Note that there is a quality issue when using sampling techniques in clustering: the result might not represent the initial data set, but rather a locally optimal solution.

This is why it was introduced the algorithm CLARANS (Clustering Large Applications based on RANdomized Search) [37]. This algorithm uses random search to generate neighbours by starting with an arbitrary node and randomly checking neighbours with maximum number of neighbours. If a neighbour represents a better partition, the process continues with this new node, improving the index to optimize. Otherwise, a local minimum is found and the algorithm restarts until a prefixed number of local minima is found.

Many others partitional algorithms can be found in literature. Most of them are based on the k-means algorithm and try to improve its drawbacks, like the necessity of choosing k beforehand, the refinement procedure which is dependant on the initial points selected (many times randomly), etc. Some other partitional algorithms are: *max-min* [35], *chin-map* [38], *ISODATA* [35], *histogram representation* [35] and others. They can be found in the most common books about clustering, but their use is not very widely spread nowadays.

2.3 Exclusive or Non-exclusive membership

This criteria to classify the clustering algorithms differs completely from the previous one. Here, it is not considered how the algorithm groups the set of objects, but it is only considered the membership functions assigned to each object. Therefore, there are no changes in the way the clusters are represented by centers, medoids, prototypes or other variant of representatives. Nor in the way the distances are computed. However, there will be changes in the definition of the cost function, given than now some other parameters will have to be included.

This classification leads to two new branches: algorithms with exclusive membership value for each one of their objects and algorithms with non-exclusive

membership. The first ones are also called *hard* clustering algorithms while the second ones are also known as *fuzzy* clustering algorithms. [1]

2.3.1 Hard Algorithms

These algorithms give every object an exclusive membership value for the cluster they have been assigned to. This means that every object belongs or not to a particular cluster and if it belongs to that cluster it can not belong to any other. The resulting clusters have the following properties:

$$D_i \cap D_j = \emptyset, \quad 1 \leq i \neq j \leq c \tag{2.3}$$

$$\emptyset \subset D_i \subset Z, \quad 1 \leq i \leq c \tag{2.4}$$

$$\bigcup_{i=1}^{c} D_i = Z. \tag{2.5}$$

where D_i and D_j are two subgroups (clusters) of objects of the total group Z, and c is the total number of clusters.

Equation (2.3) means that there are no elements in common between any two clusters, equation (2.4) means that no cluster is the empty space and finally, equation (2.5) indicates that the union of the objects from all clusters gives the whole database. The algorithms explained so far belong to this class, whose most representative is **k-means**.

2.3.2 Fuzzy Algorithms

The idea of non-exclusive membership algorithms comes up immediately if it is presented a situation like the following one. Let's have a database where a hard clustering is performed, and when the clustering is performed there is an object which is equally distanced from the centers of two clusters. Then, which cluster must be this object assigned to? What is more, sometimes clustering techniques are applied to a database

[1] *Possibilistic* clustering will here be included in this last group because the membership functions are not hard. However, in reality, fuzzy sets are a special type of possibilistic sets, because a restriction has been applied to them.

with the target of representing the data included in it with fewer objects or with a model. It is then when the objects of the clusters become represented by their center, medoid or the representative adopted for that particular case. This situation makes that those points equally distanced to more than one cluster are not represented properly if they are only assigned to any of those clusters.

Fuzzy clustering relaxes the requirement that data points have to be assigned to one and only one cluster. In these algorithms data points can belong to more than one cluster and even with different levels of membership, not just half and half. These non-exclusive cluster assignments can represent the database structure in a more natural way, specially when clusters do not have a perfect boundary, or what is the same, when clusters overlap. At these overlapping boundaries the fuzzy membership can indicate the ambiguity of the cluster assignment.

There are two major approaches to this gradual cluster assignment [1]: the first one are probabilistic methods, where we can find the *fuzzy c-means* (FCM) algorithm among others. The second one are the possibilistic methods, where the *possibilistic fuzzy c-means* (PCM) is placed.

◇ **FCM**: Fuzzy C-Means.

This algorithm allows gradual membership which will be measured as degrees in [0,1]. This makes the data model much more detailed and allows the total model to express how ambiguous or definite the database is. Given that now memberships are fuzzy, they can not be expressed with only one value or label. Now they have to be a vector for each point $x_i \in X$ the length of which is c the number of clusters:

$$u_j = (u_{1j}, ..., u_{cj})^T \tag{2.6}$$

where u_{ij} represents the membership to each cluster.

U (membership) matrix will be $c \times n$ and is called the fuzzy partition matrix. Because this algorithm is probabilistic, it will be called the probabilistic cluster partition of X. It has to meet two properties, reflected in the following equations:

$$\sum_{j=1}^{n} u_{ij} > 0 \quad \forall i \in \{1, ..., c\} \tag{2.7}$$

$$\sum_{j=1}^{c} u_{ij} = 1 \quad \forall j \in \{1, ..., n\} \tag{2.8}$$

where equation (2.7) indicates that no cluster is empty and equation (2.8) that the sum of the membership degrees for each object equals 1. This means that all data

are equally included and receives the same weight as all other data, although the distribution of this weight among the clusters differs from one object to another. As consequence, no cluster can contain all data and the membership values have to be normalized for each object.

Obviously, the closer a data point lies to the center of a cluster, the higher its degree of membership should be to this cluster. Now the problem of finding the best partition of the data set not only relies in the optimization of a cost index where computed the sum of all distances from the point to their cluster center is computed, but it is also desired to maximize the degrees of a membership. Now the general cost index will be:

$$J(X, U, C) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} \cdot d_{ij} \tag{2.9}$$

where u_{ij}^{m} is the element of the fuzzy partition matrix that is related to the element x_j and all the clusters i from 1 to c and d_{ij} is the distance of the point x_j to the center of the i-th cluster. In k-means algorithm the Euclidean distance is used. The parameter m ($m > 1$) is called *fuzzyfier* or weighting exponent. With higher values for m, the boundaries between clusters become softer. Usually $m = 2$ is chosen.

Because the cost function now depends on two parameters it is iteratively optimized. This means that first the membership degrees are optimized for fixed cluster centers, then the cluster prototypes are optimized for fixed membership degrees, which are the *optimum* values obtained in the previous iteration. Equations (2.10) and (2.11) show respectively the membership and the center updates, where i refers the actual clusters, j covers all the clusters and, finally, k is the number of objects.

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}^2}{d_{jk}^2} \right)^{\frac{2}{m-1}}} \tag{2.10}$$

$$c_i = \frac{\sum_{k=1}^{N} (u_{ik})^m \cdot x_k}{\sum_{k=1}^{N} (u_{ik})^m} \tag{2.11}$$

The choice of the optimal cluster center for fixed membership of the data is the same case as the k-means cost index, and that is way this algorithm is called *fuzzy c-means*. Fuzzy c-means is a stable and robust classification method, it is quite insensitive to the initialization and is not likely to get stuck in an undesired local minima.

◇ **PCM**: Possibilistic fuzzy C-Means.

Often it is desirable to have the property of the probabilistic membership degrees, although some times it can be misleading. High values for the membership of a datum in more than one cluster mean that the point is at the same distance to those clusters. If now there is another point with similar characteristics this can suggest that both points are close, but this might not be true, as it can be seen in figure (2.1). Both points are equally distanced to the two clusters, but they are not close.

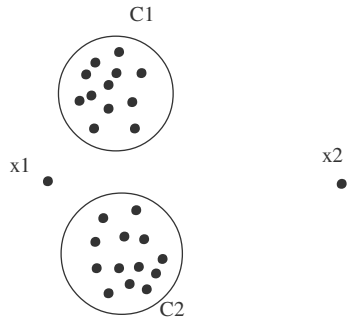

Figure 2.1: *Case when the same normalized membership value correspond to two different points*

The normalization of membership values can lead further to undesired effects in the presence of noise and outliers. The membership values affect the clustering results, since data point weights influence on cluster prototypes. A more intuitive assignment of degrees of membership can be achieve by dropping the normalization constraint, avoiding undesirable normalization effects. This last point can be highly desirable if the clusters are considered completely independent from one another.

Now the U matrix is a possibilistic cluster partition of X and still meets the equation (2.7). Now the membership degrees of each object resemble the possibility of being a member of the corresponding cluster instead of pointing how is the proportion of their belonging to each cluster. Dropping the normalization constraint, (2.8), leads to the problem that the cost index could be minimized with all $u_{ij} \to 0$. In order to avoid this trivial solution a penalty term is introduced, which forces the membership degrees away from 0, eq. (2.12). In this way the desire for strong assignments is expressed and included in the cost index.

$$J\left(X, U, C\right) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} \cdot d_{ij} + \sum_{i=1}^{c} \eta_i \sum_{j=1}^{n} \left(1 - u_{ij}\right)^{m} \tag{2.12}$$

In the possibilistic approaches, the clustering methods have to learn the weights of the data points, while in probabilistic all weights are 1, understanding by this *weight* the total membership value of one point. The formula for updating the membership degrees is:

$$u_{ij} = \frac{1}{1 + \left(\frac{d_{ij}^2}{\eta_i}\right)^{\frac{1}{m-1}}} \tag{2.13}$$

where we can see that the new value will only depend on the distance d_{ij}, and the rest of the distances do not modify this membership.

Depending on the cluster's shape the parameters η_i have different interpretations. If some knowledge of the data set is known a priori, then η_i can be set a priori. However, this does not usually happen and these parameters have to be estimated. Good estimations can be found using a probabilistic clustering model of the given data set. Then η_i can be estimated using the probabilistic fuzzy partition matrix, as shown in (2.14).

$$\eta_i = \frac{\sum_{j=1}^{n} u_{ij}^m \cdot d_{ij}}{\sum_{j=1}^{n} u_{ij}^m} \tag{2.14}$$

The best property of this algorithm is that it is able to leave some points which are far from cluster (outliers) outside and do not assign them to any cluster.

The main difference between these two approaches is that probabilistic algorithms are forced to partition the data exhaustively while the corresponding possibilistic methods are not compelled to do so. Another difference is that probabilistic methods distribute the total membership of the data points while the possibilistic methods are required to determine the data point weights themselves. Probabilistic algorithms attempt to cover all data points with clusters, which can be an advantage when the real clusters have this property. In the possibilistic case, there is no interaction between clusters. Given that the initialization of the *PCM* is much more complex than the initialization of *FCM* but it performs better, sometimes *FCM* is used to initialize η_i and after that the possibilistic algorithm is applied. However, this has a high computational cost and some applications make it impractical.

As the distance measure used is the Euclidean distance only spherical shapes can be detected. Some algorithms have been designed to overcome this problem, by modifying the distance computed. All of them can work with both probabilistic or possibilistic methods, [1]. A presentation of these methods is done next.

◇ **GK**: Gustafson-Kessel [1].

This algorithm replaces the Euclidean distance used in *FCM* and *PCM* by the Mahalanobis distance, equation (1.6), in order to adopt various shapes and sizes of the clusters. In *FCM* and *PCM*, as the distance function used is the Euclidean distance the shapes of the clusters to be identified are only hyper-spherical.

GK models each cluster D_i by both its center, c_i, and its covariance matrix, Σ_i. Both parameters have to be learned. The eigenvalues of the matrix Σ_i represent the shape of the cluster D_i. Specific constraints can be set depending on the requirements of the application. For instance, restricting to axis-parallel cluster shapes by considering only diagonal matrices. This case is preferred when clustering is used for the generation of fuzzy rule systems. The distance function is now defined as:

$$d_{ij}^2 = (x_j - c_i)^T \Sigma_i^{-1} (x_j - c_i) \tag{2.15}$$

Apart from that, the cost function and the update equation for the cluster centers and the update equation for the membership degrees are identical to the *FCM* or *PCM*, depending which approach is being used. The update of the covariance matrices is given by:

$$\Sigma_i = \frac{\Sigma_i^*}{\sqrt[p]{\det(\Sigma_i^*)}} \tag{2.16}$$

where Σ_i^* is:

$$\Sigma_i^* = \frac{\sum_{j=1}^{n} u_{ij} (x_j - c_i)^T (x_j - c_i)}{\sum_{j=1}^{n} u_{ij}} \tag{2.17}$$

They are defined as the covariance of the data assigned to cluster i, modified to incorporate fuzzy assignment information u_{ij}.

GK extracts more information about the data than *FCM* and *PCM*, but it is more sensitive to its initialization. Then, a good recommendation could be the use of *FCM* or *PCM* for its initialization and afterwards, applying GK.

◇ **FSC**: Fuzzy Shell Clustering, [1].

All the algorithms described so far in this section detect shapes like solid objects, and thus they are called *solid* algorithms. Variants of *FCM* and *PCM* have been proposed to detect some other shapes, like lines, circles or ellipses, called *shell*

algorithms. They extract prototypes that have a different nature from the data points and for that they need to modify the definition of the distance function. Some of these algorithms are:

- **FCV**: Fuzzy C-Varieties, [1].

  It detects lines, planes or hyperplanes. Each cluster is a subspace defined by a point and a set of orthogonal unit vectors $D_i = (c_i, e_{i1}, ..., e_{iq})$ where q is the dimension of the subspace. The distance function is now defined as:

  $$d(x_j, D_i) = \|x_j - c_i\|^2 - \sum_{l=1}^{q} (x_j - c_i)^T e_{il}. \tag{2.18}$$

  This algorithm can also be used for construction of locally linear models of data with underlying interrelations.

- **FCQS**: Fuzzy C-Quadratic Shells, [1].

  This algorithm is able to recognize ellipses, hyperbolas, parabolas or linear clusters. Since sometimes the projections of the circle-shaped clusters form an ellipse this algorithm is very useful.

Other fuzzy shell algorithms are *AFCE* to recognize ellipses, *FCRS* to recognize non-smooth shapes, such as rectangles, *FC2RS* to recognize rectangles and other non-smooth polygonal shapes. A description of all them and further references can be found in [1]

◇ **KFC**: Kernel-based Fuzzy Clustering, [1].

Kernel learning methods constitute a set of machine learning algorithms that make it possible to extend classic linear algorithms. The kernel variants of fuzzy clustering algorithms further modify the distance function to handle non-vectorial data, such as sequences, trees or graphs, without needing to modify the algorithms themselves. The aim is two-fold: first, they make possible to treat tasks that require a more complex algorithm than a linear one and, second, they make it possible to apply algorithms to data that are not described in a vectorial form. More generally, kernel methods can be applied **independently** of the data nature.

Kernel methods are based on an implicit data representation transformation, with which the normal space is transformed to *feature space*. The second principle of kernel methods is that data are not handled directly in the feature space, they are only handle through their scalar products that are computed using the initial representation. Here, algorithms are written only in terms of scalar products between data. Then, the data representation improvement comes from using scalar products based on an implicit transformation of the data.

Applying this to clustering, it aims at extracting prototypes that have a different nature from the data points, and thus it modifies the concept of distance between points. In the kernel approach, the similarity is computed between pairs of data points and does not involve cluster centers. On the other hand, kernel methods do not have an explicit representative of the cluster and can not be seen as prototype-based clustering methods.

The application of kernel methods needs to select the kernel and its parameters and this may be difficult, but they are able to cluster non-vectorial defined objets, which is one of their advantages.

◇ **FCRM**: Fuzzy C-Regression Models, [22].

Some algorithms are designed, instead to form different groups, to construct a model based on the inputs in order to predict the output from a new input. These algorithms are also called *model-based* clustering algorithms.

FCRM is one of this type, which will be better described in section (2.4). Its cost function basically tries to find a set of fuzzy models to represent the output with a linear combination of the inputs.

Given a data set where each independent input observation x_k has a correspondent output observation y_k, it is assumed that several lineal models c describe the relation between the input and the output: $\hat{y} = f_i(\mathbf{x}; \beta_i) + \epsilon_i$ where $1 \leq i \leq c$. This is known as a *switching regression model*.

If each object is described by a combination of all or some models, then the problem to solve can be divided into two: finding a good estimations of parameters β_i which define $\hat{y}$ (predicted output) and finding the membership of each object to each one of those models.

In this algorithm these two problems are solved simultaneously, by an iterative approach. The membership of each object meets equations (2.7) and (2.8). The solution will be one that minimizes a cost index defined by the summation of all errors between the real output and the output of the model weighed by the membership of that object:

$$J(x, U, c, \beta) = \sum_{k=1}^{n} \sum_{i=1}^{c} u_{ik}^{m} \cdot E_{ik}(\beta_i) \tag{2.19}$$

where E_{ik} is the square error between the output of the object k and the output given the the local model i.

Basically, the steps to find the solution are:

1. Initialization of the variables: m, $U^{(0)}$, ϵ, etc. Choice of the error measure E_{ik}.

2. Calculate the values for all β_i. This will be done with least squares.
3. Update the membership matrix.
4. Compare the new matrix with the last one, if the sum of changes between the two matrices is smaller than ϵ then iteration will stop. Otherwise, go back to 2).

This algorithm is tested to work very well when the correct number of clusters, c, is chosen and when data is distributed following lineal models.

◇ **AFCR**: Adaptive Fuzzy Clustering and Fuzzy Prediction Models, [40]

This algorithm, because it is based on the FCRM, could also be classified as a model-based algorithm. It addresses the problem of the shapes of the clusters. Here they are changed dynamically and adaptively in the clustering process.

It is based on a cost index where distances to centers and modeling error are weighed, equation (2.20) and the weight of each term changes dynamically:

$$J\left(x, U, c, \beta\right) = \sum_{k=1}^{n} \sum_{i=1}^{c} u_{ik}^{m} \cdot Z_{ik} \tag{2.20}$$

where: $Z_{ik} = \left(1 - \alpha_i\right) D_{ik} + \alpha_i E_{ik}\left(\beta_i\right)$ and $\alpha_i = 1 - \frac{\min_j\{\lambda_{ij}\}}{\max_j\{\lambda_{ij}\}}$. D_{ik} refers to the distance between object k and i-th center of cluster. α_i corresponds to eigenvalues of the variance-covariance matrix, which will determine the shape of the clusters.

In that way, the index at first will have hyper-sphere clusters because the distance term will be more influential, and therefore α_i will be near to 0: the index will take into account more the distances than the output error. As the clustering algorithm progresses, the clusters will be more similar to hyper-ellipsoids and α_i will be near to 1: the index will take more into account the output error and therefore the regression parameters β_i estimation will be more accurate.

The steps to find the solution are very similar to the case of the FCRM:

1. Initialization of the variables: m, $U^{(0)}$, ϵ, etc. Choice of the error measure E_{ik} and distance measure D_{ik}.
2. Calculate the values for all β_i. This will be done applying least squares using the membership matrix.
3. Update the membership matrix. Compute α_i.
4. Compare the new matrix with the last one, if the sum of changes between the two matrices is smaller than ϵ then iteration will stop. Otherwise, go back to 2.

2.4 Other Classifications and their algorithms

2.4.1 Other classifications

The three main classifications introduced section 2.1 are not the only existent criteria to classify the clustering algorithms. Some other methods have come up focused on solving specific problems. Among them we can find:

- ◇ **Density-based Clustering**: these algorithms group objects according to a specific density objective function. Density is usually defined as the number of objects in a particular neighbourhood of a data object. In these approaches a given cluster continues growing as long as the number of objects in the neighbourhood exceeds some parameter (threshold).
- ◇ **Grid-based Clustering**: The objective of these algorithms is to quantize the data set into a number of cells and then work with objects belonging to these cells. They do not relocate points, but rather build several hierarchical levels of groups of objects.
- ◇ **Model-based Clustering**: These algorithms find good approximations of model parameters that best fit the data. They are closer to density-based algorithms in that they grow particular clusters so that the preconceived model is improved, but most of the times they do not use the concept of density. They can be both hierarchical or partitional.
- ◇ **Spatial Clustering**: Those algorithms try to find geometric shapes within the data set and then construct clusters with the elements that do form the geometrical shape. They use, most of the times, density measures.

We can see as most of the new classifications overlap between them as much as the traditional classifications. This is normal as fixing one criteria for one characteristic the other characteristics are to be chosen among the possible ones characterizing the other criteria.

2.4.2 Algorithms

Here, the most popular algorithms within each new classification will be described:

◇ **Density-based Clustering**:

Clusters can be thought of as regions of high density, separated by regions of low density. *Density* refers to the number of data objects in the neighbourhood. There are two approaches for these methods: the first one assigns density to a training data point while the second one assigns density to a point in the attribute space. Below there is a description of the main algorithms in this classification.

- **DBSCAN**: Density-Based Spatial Clustering of Applications with Noise [42]. This algorithm finds, for each object, the neighbourhood that contains a minimum number of objects. A cluster is defined as a set of all points transitively connected by their neighbours. Therefore, a *closeness* measure has to be defined. Advantages of this algorithm is that it finds clusters with different shapes and it is not sensitive to the input order. Besides, it is incremental, given that a new introduced object only affects a certain neighbourhood. Disadvantages are that the user has to define two parameters: the radius of the neighbourhood, ϵ, and the minimum number of objects it should contain, $MinPts$. There is no way to define these parameters easily. In fact, most of the applications require different parameter values for different parts of the database.
- **OPTICS**: Ordering Points To Identify the Clustering Structure [6].
  OPTICS is an extension of the previous algorithm that relaxes the strict requirement of input parameters. OPTICS computes an augmented cluster ordering, in order to automatically and interactively cluster the data. What this algorithm does is that it keeps ϵ and $MinPts$ and it covers an spectrum of all different $\epsilon' \geq \epsilon$. With each point, OPTICS stores only two additional fields, the *core* and the reachability-distance, where the *core* distance is the distance to $MinPts'$ nearest neighbour when it dos not exceed ϵ. Otherwise, it is undefined. OPTICS is an extension of DBSCAN in the direction of different local densities.
- **DENCLUE**: DENsity-based CLUstEring [24].
  It uses a *density function* to point the influence of an object to its neighbourhood and overall density is modeled as the sum of the influence functions of all objects. The influence function can be anyone, as long as it is determined by the distance. Usually, this function adopts two values, one for the activation (1) and other for the non-activation (0). The activation is produced when the distance is larger than a prefixed threshold, σ:

$$f_{Square}(x, y) = \begin{cases} 1 & if \quad d(x, y) > \sigma \\ 0 & otherwise \end{cases} \tag{2.21}$$

DENCLUE concentrates on local maxima of density functions called *density attractors*. Then a *center-defined* cluster for an attractor is a subset C, where the density function at the attractor is no less than the noise threshold defined, ξ.

The main disadvantages is that DENCLUE depends on the noise threshold ξ and the density parameter ω. On the other hand, the advantages are quite numerous: it handles outliers and exceptions, can find arbitrary shapes, keeps information about the cells that contain objects, etc.

Both DBSCAN and OPTICS are algorithms which apply the first approach while DENCLUE follows the second approach methods, mentioned at the beginning of this subsection. But these are not the only existent algorithms. We can also find in this classification algorithms like: *GDBSCAN* [42] and *GDBCLASD*, which are variations to improve the DBSCAN algorithm.

◇ **Grid-based Clustering**:

These algorithms are based on limiting the amount of computations by considering multirectangular segments. The procedure is as follows: 1) Get Data, 2) Grid Data, 3) Space Partitioning, 4) Data partitioning. This indirect handling of data has one advantage and this is that accumulation of cells of the grid makes grid-based clustering techniques independent of data ordering. The main drawback is that divisions are in almost all cases unbalanced. The main algorithms are:

- **STING**: STatistical INformation Grid [45].

  It works with numerical attributes and is designed to facilitate "region oriented" queries. STING assembles summary statistics in a hierarchical tree of nodes that are grid cells. Each cell has four default *children*. Each cell stores a point counter and information about the attributes of the objects in that cell, like mean, minimum, maximum etc. Measures are accumulated starting from bottom-level cells. When the cell tree is built, certain cells are identified and connected in clusters, similar to DBSCAN. STING is parellelizable and allows multiresolution. However, the definition of the right level of granularity (size of the cells) is not straightforward.

- **CLIQUE**: CLustering In QUEst [2].

  This is a subspace clustering algorithm. CLIQUE partitions data into hyper-rectangles and finds the dense ones, i.e. the ones with a certain number of objects in a given neighbourhood and then, unions of such rectangles constitute clusters. CLIQUE finds finds 1-dimensional dense cells, then 2-dimensional dense rectangles and so on, until the k-hype-rectangles are found.

Other algorithms are: *WaveCluster*, *MAFIA*, *BANG* and many others.

◇ **Model-based Clustering**:

The goal of these algorithms is finding the parameters of a model, based on the input data set, to describe the behaviour of the output. So, these models try to find the coefficients which will weigh up each of the inputs to get the output for each object. As they are part of the clustering techniques, they will find the parameters of several models, which will have to be integrated to describe the behaviour of the full output of the data set. Linear models are the simplest models and many algorithms try to get the output through combination of these linear models. Some examples of this type are:

- **EM**: Expectation-Maximization [10].

  EM is an extension of the k-means algorithm. EM assigns each object to a dedicated cluster according to the probability of membership for that object. The probability distribution function is the *multivariate Gaussian* and the main goal is the iterative discover of good values for its parameters. To this end the objective function includes how well the probabilistic model fits it. The algorithm can handle various shapes of clusters but its computational cost is quite high.

- **FCRM** and **AFCR**: described before.

◇ **Spatial Clustering**:

This classification refers to algorithms that do not look for high dense areas in the input space, but for zones where the objects are in a geometric form: circle, ellipsoid, etc. In this case, areas are no highly populated, so there is the need to define a specific cost function to detect the forms of the objects.

- **GRAVIclust**:

  This is an algorithm that tries to find clusters defined by the area, the center and the radius of the area and the density, [27]. Therefore, the clusters will have more or less a circular shape. The regions with a larger number of objects will have higher probability of constructing a cluster. Usually the distance measure used is the Euclidean distance. The search is improved iteratively, looking for the centers of the clusters by calculating the mass center (or gravity center) of all objects that are in each cluster.

2.5 Summary

Many algorithms have been described through this chapter. It has been seen that there are many classification criteria and that the same algorithm can always be classified in

more than one group because different properties are present in all algorithms. However, an algorithm is classified mainly attending to the purpose for which it was created, and therefore, there is always one group which characterizes it more than others.

Table 2.1 shows concisely the different classifications and classes,as well as algorithms belonging to each one.

Table 2.1: *Summary of clustering algorithms by classifications*

Classific.	Hierarchial	Partitional	Hard	Fuzzy	Density-based	Grid-based	Model-based	Spatial
Algorithms	BIRCH CURE CHAMELEON COBWEB CLASSIT SLINK	k-means PAM CLARA CLARANS max-min chin-map ISODATA	k-means	FCM GK Possibilistic	DBSCAN DENCLUE GDBSCAN	STING CLIQUE WaveCluster BANG	EM FCRM AFCR	GRAVIclust

3. Quantitative, Qualitative and Mixed Clustering

3.1 Importance

The majority of classical algorithms described in the previous chapter were designed to work with numerical databases with only few exceptions that can both work with numerical and categorical attributes. They worked well in the origins of clustering, when it was used to classify data sets where the objects were described with numerical characteristics. Nevertheless, at present, the clustering algorithms have grown in importance and number of applications and each day they are required to work with more complex and complete databases. This means that the length of the objects is usually longer (greater number of attributes that characterize every object) and the type of objects is not only numerical but can also be categorical. In fact, these days most of the data sets where clustering is desired to be applied do not only have numerical or categorical data, but these databases contain both, i.e it is *mixed data set*. This is the reason why it has been given special attention to this type of classification and is treated in a chapter on its own. In fact, although this work reviews all clustering techniques and describes the most important and used ones, it aims at proposing a new approach to work with mixed data, given that there is neither an algorithm nor a technique that can perform with a high level of accuracy yet, as it will be seen through this chapter.

Next section reviews some clustering algorithms developed for categorical data and the following one, different algorithms and techniques used to work with both type of data, the so-called mixed databases.

3.2 Categorical Algorithms

The description of this type of data was done in Section 1.2.1, and here, there is a list of the main characteristics [4]:

- ◇ Categorical Data have *no single ordering*: there are several ways in which they can be ordered, but there is no an absolute way, i.e. no way is more logical or proper than any other.
- ◇ This type of data is discrete and independent.
- ◇ Categorical Data can be visualized only if one order is adopted and then the visualization depends on this order.
- ◇ Categorical Data define, therefore, no a priori structure to work with.

It is obvious that different methods to discover natural grouping of numerical data are needed. These methods will differ from the numerical methods explained so far mainly in the distance measure, that can not be like those used up to now. The most popular and used algorithms are:

- ◇ **K-modes**

  This algorithm [26] was the first one oriented to categorical data. It is based on the *k-means* algorithm and uses the same structure but with a different similarity/dissimilarity measure where the main differences are that the *means* are replaced by *modes* and that the way of updating them is a *frequency based method*. The dissimilarity measure used is like equations (1.13) and (1.14). This expression counts the number of mismatches the two objects have on their attributes. All attributes are considered to have the same weight. If the frequencies of the values in a set are to be included to give the attributes different importance, the equation would be:

$$d(x,y) = \sum_{i=1}^{z} \frac{n_{x_i} + n_{y_i}}{n_{x_i} \cdot n_{y_i}} \cdot \delta_{xy}^{i} \tag{3.1}$$

  where n_{x_i} and n_{y_i} are the numbers of objects in the database with attributes values x_i and y_i for attribute i respectively. The **mode** of a set is the value that appears in the majority of elements of the set. Every cluster c, $1 \leq c \leq k$ will have a mode, defined by a vector $Q^c = (x_1^c, ..., x_z^c)$, where z is the dimension of the attribute vector. The vector Q^c that minimizes the cost index, eq. (3.2), is the desired output of the algorithm.

$$E = \sum_{c=1}^{k} \sum_{x \in c} d\left(x, Q^c\right) \tag{3.2}$$

Given that the k-modes algorithm is based on the k-means structure, it has the same advantages and disadvantages as this one.

◇ **ROCK**: RObust Clustering Using LinKs, [20].

ROCK is a hierarchical algorithm. An approach based on the concept of *link* between data objects which helps to overcome the problems that there exist when using the Euclidean distance.

Some definitions follows:

- Two objects x and y are called *neighbours* if their similarity exceeds a certain threshold θ set by the user: $sim(x, y) \geq \theta$.
- For two data objects, we define: $link(x, y)$, which is the number of common neighbours between the two objects, i.e. the number of objects that are similar to both.
- The *interconnectivity* between two clusters D_1 and D_2 is given by the number of *cross-links* between them, which is the sum of all of their links.
- The expected number of links in a cluster D_i is given by eq. (3.3) where eq. (3.4) is set for all experiments:

$$n_i^{1+2 \cdot f(\theta)} \tag{3.3}$$

$$f\left(\theta\right) = \frac{1-\theta}{1+\theta} \tag{3.4}$$

So, ROCK measures the similarity of two clusters by comparing the aggregate *interconnectivity* of two clusters with a user-specified static *interconnectivity model*. After that, this algorithm's cost function (3.5) has to be maximized.

$$E = \sum_{i=1}^{k} n_i \sum_{x_q, x_r \in D_i} \frac{link(x_q, x_r)}{n_i^{1+f(\theta)}} \tag{3.5}$$

ROCK can be described like this: firstly a random sample is drawn, and secondly a clustering hierarchical algorithm is involved to merge cluster. Therefore, we need a measure to identify clusters that should merge in every step. Finally, the clusters are labeled. In order to perform the merging, the *goodness measure*:

$$g\left(D_i, D_j\right) = \frac{link\left[D_i, D_j\right]}{\left(n_i + n_j\right)^{1+2\cdot f(\theta)} - n_i^{1+2\cdot f(\theta)} - n_j^{1+2\cdot f(\theta)}} \tag{3.6}$$

is used, where eq. (3.7) is the cross-link between clusters. The pair of clusters for which this measure is maximum are the best ones to be merged.

$$link\left[D_i, D_j\right] = \sum_{x_q \in D_i, x_r \in D_j} link(x_q, x_r) \tag{3.7}$$

ROCK algorithm is complex and depends on the user-parameter θ, but its performance can reach high levels.

◇ **STIRR**: Sieving Through Iterated Relational Reinforcement [19].

This is one of the most influential algorithms for categorical clustering. It uses an iterative approach where two data objects are considered to be similar if their attributes have a large overlap. The key features are:

- There is no a priory quantization. This means that the similarity only considers the co-occurrence of two objects.
- The similarity measure is defined so that it can apply even to items that do not occur together. It only looks at attributes that do overlap.

STIRR uses spectral partitioning on hypergraph clustering using non-linear dynamical systems. It proposes a weight propagation method which works as follows:

1. It first seed a particular item of interest, A, with a small amount of weight, although this is not essentially required as all weights could be initialized to 1.
2. The weight propagates to items with which A co-occurs frequently.
3. These items, having acquired the weight, propagate it further.
4. The process iterates until it converges.

Some concepts must be explained before the technique is applied:

- *Representation*: Each possible value for each possible attribute is represented by a node.
- *Configuration*: the assignment of a weight ω_i to each node v; the whole configuration will be referred to as ω.
- *Normalization*: re-scales weights so that their squares add up to 1 and ensure orthonormality.

Table 3.1: *Database with 3 categorical attributes*

Object	At_1	At_2	At_3
1	A	W	1
2	A	X	1
3	A	X	2
4	B	W	2
5	B	X	2
6	C	Y	3
7	C	Z	3

- *Combining Operator* $\oplus$: this can be 1) a product operator, 2) addition operator, 3) a combining rule or 4) a limiting version of the combining rule.
- *Dynamical System*: repeated application of a function f on some set of values.
- *Fixed Points*: points such that $f(u) = u$, for all nodes u.
- *Function* f: maps one configuration to another.

One of the key components of this approach is the choice of an initial configuration. Two ways of choosing it:

- If the user does not want to focus on a particular weight, then the initialization can be done uniformly, and all weights will be equal to 1.
- If the user does want to focus on a particular weight, we give this weight a higher value than the others.

Table 3.1 and Figure 3.1 show a representation of this algorithm.

STIRR converges and experiments show good results. However experiments containing no more than two clusters have been done.

◇ **CACTUS**: Clustering Categorical data Using Summaries, [17].

The main idea is that summary information constructed from the database is enough for discovering well-defined clusters, similar to BIRCH in the numerical branch of clustering algorithms. CACTUS looks for hyperrectangular clusters, called *intervalar regions*. Some measures are defined, like *support*, σ:

$$\sigma\left(a_i, a_j\right) = \left|\left\{\ t \in Dom : t.A_i = a_i \quad and \quad t.A_j = a_j\ \right\}\right| \tag{3.8}$$

which is the number of objects where a_i and a_j co-occur, Dom indicates the domain of the attribute. The support of the region , $\sigma(S)$, which is the number of objects in the data set that belong to S. $E[\sigma(S)]$:

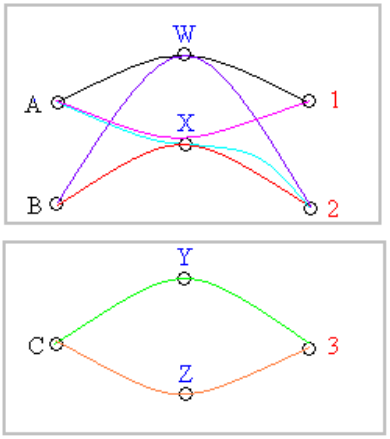

Figure 3.1: *Representation of the database in table 3.1 in STIRR*

$$E\left[\sigma\left(S\right)\right] = |Dom| \cdot \frac{|S_1| \times ... \times |S_n|}{|Dom_1| \times ... \times |Dom_n|} \tag{3.9}$$

is the expected support of S and $E[\sigma(a_i, a_j)]$ is the expected support of (a_i, a_j):

$$E\left[\sigma\left(a_i, a_j\right)\right] = |Dom| \cdot \frac{1}{|Dom_i| \times |Dom_j|} \tag{3.10}$$

Finally, it is said that values a_i and a_j are *strongly connected* if

$$\sigma\left(a_i, a_j\right) > \alpha \cdot E\left[\sigma\left(a_i, a_j\right)\right] \tag{3.11}$$

is true, i.e. if the number of data points having both a_i and a_j is larger than the frequency expected under an independence assumption by the user-defined margin α.

A cluster can be so, when:

- for all i, j, D_i and D_j are strongly connected.
- D_i is maximal for all i
- $support(D)$ is α times the expected one.

The procedure steps are:

1. *Summarization*: summaries of data are computed. In this phase two types of summaries can be computed:

- *inter-attribute* summaries: counts of all strongly connected attribute value pairs from different attributes.
- *intra-attribute* summaries: computation of similarities between attribute values of the same attribute.

2. *Clustering*: using the summaries, candidate clusters are computed. Here, the algorithm analyzes every attribute to compute all cluster projections. A projection is a subset of the attribute that is strongly connected to the attribute values of every other attribute.
3. *Validation*: the set of candidate clusters are validated. Here, to validate candidates, the algorithms checks if its support is greater than the threshold and then detects false candidates.

The graphic representation of Table 3.1 is the same as when applying STIRR. However, now the detected clusters depend on other parameters.

For instance, if the co-occurrence is set to 2 and the support to 3, only one cluster will be detected, Figure 3.2. It can be seen how in the upper part there are 3 co-occurrences with support 2 (A-1, X-2 and B-2). While in the lower part there is only one (C-3).

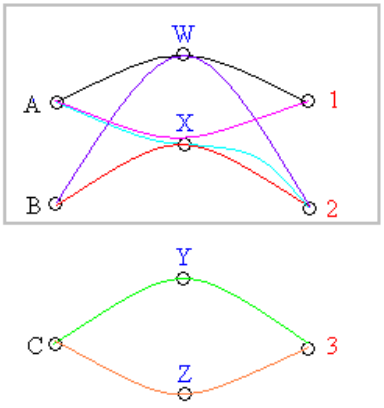

Figure 3.2: *Example using CACTUS*

It can be seen how this method builds clusters based on rectangular shapes. The steps are the mentioned before and are quite dependent on the representation of the data.

Although CACTUS is a quite complicated algorithm, an advantage of it is that it is scalable, because it only requires a pass over the data set.

◇ **CLICKS**: Name that comes from the Subspace CLustering of Categorical data via maximal K-partite cliques [49].

It is based on the same philosophy as CACTUS, that finds clusters in different n-dimensional data and tries to overcome the issue that such algorithm only mines a limited class of subspace clusters. The idea, same as CACTUS, is similar to the algorithm CLIQUE [39], although this one is for numerical data.

This algorithm uses the same concepts for the attributes and the attributes' domains described for CACTUS like: *support* σ, *expected support* $E[\sigma(S)]$, *user-defined margin* α, *strongly connected*, eq. (3.11). And adds some new ones as: *undirected graph*, $\Gamma_D = (V, E)$, where $((v_i, v_j) \in E \Leftrightarrow S_\alpha(v_i \times v_j) = 1)$ is called the *k-partite graph*. A subset $C \subseteq V$ is a *k-partite clique* in Γ_D if and only if every pair of vertices $v_i \in C \cap D_i$ and $v_j \in C \cap D_j$ with $(i \neq j)$ are connected by any edge in Γ_D. In these equations S_α refer to the density indicator function, C and D are (subspace) clusters and v_n refer to the vertex on the graph:

$$V = \bigcup_{i=1}^{n} D_i \tag{3.12}$$

Then, in the graph every element of the domain will be represented as a vertex and a union between vertices, edge or *clique* will be represented if the connectivity between the two vertices is over the user-defined value α. A cluster C will be considered so if the k-subspace is *dense* and *maximal* which means that the number of subspaces contained on the set of subspaces given can not be larger.

This algorithm is divided into three steps:

- *Pre-processing*: In this step all the attributes are evaluated and the *k-partite graph* is created giving Γ_D. The vertices of the graph are placed.
- *Clique detection*: The co-occurrences are evaluated and all the *k-partite cliques* are added to the graph Γ_D.
- *Post-processing*: The *support* of the candidate cliques are verified within the original data set to form the final clusters. This does not lead to a complete evaluation of the original data set. If that is desired an extra step has to be performed. This is the *selective vertical expansion* which aims to discover those clusters in subspaces that are not dense (as the density property is not *downward closed*) but the support exceeds the margin α.

The two first steps are very similar to the CACTUS proceeding, while the third step is the one that makes this algorithm overcome the issue of limited subspace cluster detection and makes it have a good performance.

◇ **MVFCM**: Mixed-type Variable FCM [36].

This is an extension and modification of the FCM algorithm to work with data sets that contains mixed features of symbolic and fuzzy characteristics. *Fuzzy* data are another type of data like imprecise or with a source of uncertainty. This algorithm is based on modifications done in the FCM dissimilarity function and also on changes in the Hathaway's parametric approach for fuzzy data. Once those changes and modifications are done, the clustering algorithm is capable to classify clusters when mixed data, including fuzzy numbers, occur.

The metrics used for each type of attribute in this algorithms as described here.

Quantitative feature components.

The Gowda and Diday's dissimilarity function, eq. (1.7)-(1.10), is the one that has been modified here to be able to handle symbolic data. The modifications have been performed in order to make the new dissimilarity function behave as much as possible as the Hausdorff metric, given than this last one can handle much better the situations of different numbers in the features between a pair of objects. Then, the *position*, *span* and *content* equations will be like, respectively:

$$d_p\left(A_k, B_k\right) = \frac{\left|\left(al + a\mu\right)/2 - \left(bl + b\mu\right)/2\right|}{U_k} \tag{3.13}$$

$$d_s\left(A_k, B_k\right) = \frac{\left|l_a - l_b\right|}{U_k + l_a + l_b - inter} \tag{3.14}$$

$$d_c\left(A_k, B_k\right) = \frac{\left|l_a + l_b - 2 \cdot inter\right|}{U_k + l_a + l_b - inter} \tag{3.15}$$

The final distance will be the addition of these three parameters. All the parameters of these equations have already been described in the subsection 1.2.2 in the description of the Gowda and Diday's dissimilarity measure.

Qualitative feature components.

The problems that appeared in the Gowda and Diday's dissimilarity measure for quantitative data, were not present in the same measure for qualitative data, so that the measure defined as in equations (1.15) and (1.16), with all the parameters described in the same section, will be valid here and, therefore, used.

Fuzzy feature components.

Hathaway et. al. in [23] proposed FCM clustering for symmetric TFN (trapezoidal fuzzy numbers). However, this approach did not consider the left and right shapes

of fuzzy numbers. Then, in this algorithm this issue has been overcome by taking another dissimilarity measure for fuzzy numbers that do consider these cases.

A trapezoidal fuzzy shape can be parameterized with four parameters $A = m(a_1, a_2, a_3, a_4))$, as can be seen in Figure 3.3

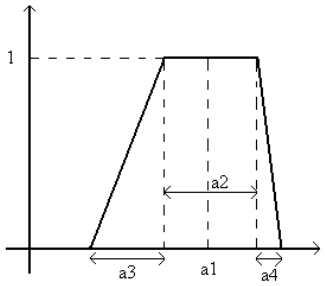

Figure 3.3: *Parametrization of TFN*

With this four parameters up to four different shapes can be fully described: numbers, intervals, triangular and trapezoidal shapes, as shown in Figure 3.4.

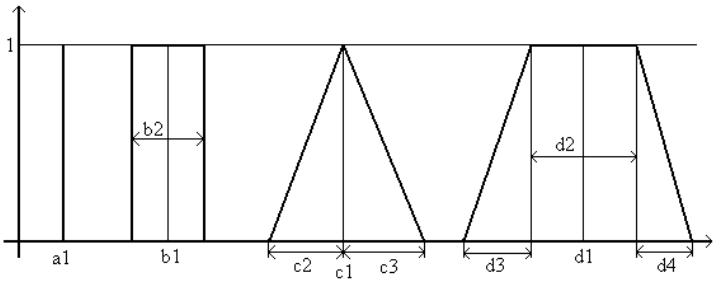

Figure 3.4: *Four shapes fully described with TFN parameters*

If instead of this parameterization, the one described in equation (3.16), is used with parameters $A = (m_1, m_2, \alpha, \beta)$, then the LR-shape fuzzy number could also be described and taken into account:

$$\mu_x(x) = \begin{cases} L\left(\frac{m_1 - x}{\alpha}\right) & for \quad x \leq m_1 \\ 1 & for \quad m_1 \leq x \leq m_2 \\ R\left(\frac{x - m_2}{\beta}\right) & for \quad x \geq m_2 \end{cases} \tag{3.16}$$

Parameters α and β ($\alpha > 0$ and $\beta > 0$) are called the left and right spreads, respectively. If now the Yang's distance is used with the parameters of this new approach, this new distance includes some information about the shape of the membership function, while the Hathaway's measure did not. With this new parametrization the problem of the LR-shapes is overcome and at the same time, a better result for the distance is obtained.

The fuzzy clustering algorithm.

The algorithm, based on FCM, includes the already explained modifications, the new definitions of the Gowda and Diday's dissimilarity function and the Yang measure with the new parameterization of the TFN, where the original L-R has been modified, in the FCM index cost to be minimized. Now two new update equations have to be defined for the iterative process, one for the symbolic data, and other for the fuzzy type data. The first one is based on the modification of the membership function to achieve a lower value of cost index, while the second one is based on the modification of the four parameters that define the shape of the membership function to make the index smaller. So, the steps of the algorithm can be described as follows:

- *Step 1*: Set the initial conditions. Including the initialization of all the membership values.
- *Step 2*: Compute the initial centers for the clusters.
- *Step 3*: Update the membership values.
- *Step 4*: Compare the values of this membership functions with the previous ones. If the difference is lower than a user-defined parameter ϵ, then stop the iteration process. If it is larger, go back to step 2.

This algorithm is one of the few ones that can handle fuzzy type data, being this one of the best properties of the algorithm.

◇ **COOLCAT**: An entropy based algorithm for categorical clustering, [7].

This algorithm was designed with the purpose of clustering large data sets in the way of *data streams* (data that are continuously arriving to be clustered). It uses, contrary to most of the categorical clustering algorithms in the literature, the notion of *entropy* to measure the similarity/dissimilarity of the elements. Therefore, the aim of this algorithm is to minimize the expected entropy of the clusters. Besides, as COOLCAT acts incrementally, it does not need, for a new arriving object, to re-process the whole set to place it in a cluster.

It assumes independence of the attributes of the data set. In that way, the total entropy function, which is dependent on probabilities, can be computed as the

addition of all the entropies, making it possible and easy to compute incrementally the data. The function which has to be minimized is:

$$\overline{E}\left(\breve{C}\right) = \sum_{k} \left(\frac{|C_k|}{|D|} \left(E\left(C_k\right)\right)\right) \tag{3.17}$$

where $\breve{C}$ represents the clustering, C_k the cluster k, D the data set and $E(C_k)$ the entropy of the cluster k.

The algorithm consists of two steps: *initialization* and *incremental step*. In the first step the algorithm "defines" the N clusters (N is a user-defined parameter) where elements will be placed. This is done by selecting the elements that have the largest entropy, or what is the same, the N elements of the set that maximize the entropy function and therefore, that are most different. In the second step, once the clusters have been chosen, the rest of elements are placed in them incrementally. This means that the placement is done element by element and, consequently, the order of processing the points has some impact on the quality of the clusters obtained. This is why after the clustering has been performed, a final phase remains to be done. The m (user-defined parameter) elements that have the largest entropy are removed from their respective clusters and are placed again following the same criteria of minimizing the entropy.

The main advantage of this algorithm is that it avoids the usage of a metric measure which is not well defined for categorical attributes, and replaces it for the entropy function which is much more convenient. Nevertheless, the data order dependency makes the clustering results no highly reliable, and therefore not applicable in many cases.

Other algorithms are: *LIMBO* [5], *KEROUAC* [30], *Snob* [44], *SQUEEZER* [50] and many others can be found in literature, given that in the last years research about categorical clustering has been growing in importance.

3.3 Mixed databases: Approaches to work with them

This section is divided into two subsections. In the first one there is a description of the algorithms that have been developed to work with both type of data simultaneously, whether they are genuine algorithms or mere fusion of a numerical and a categorical algorithm. In the second section, it will be described a common approach which is not based on any particular algorithm, but it is focused on the data transformation in order to make all data be of the same type and then apply only one algorithm that works well

for that type, either numerical or categorical, depending which type of data have been transformed.

3.3.1 Mixed Data algorithms

Some of the developed algorithms to work with mixed data are:

◇ **k-prototypes**

This algorithm [26] is an integration of $k-means$ and $k-modes$, adding both dissimilarity measures:

$$s^r + \gamma s^c \tag{3.18}$$

where s^r is the dissimilarity on numeric attributes, s^c the dissimilarity on categorical attributes, and γ is a weight to balance the two parts and avoid favoring either type of attribute. This last one is a user-defined parameter, and depends on the particular application.

◇ **dSQUEEZER** and **usmSQUEEZER**: **d**iscrete SQUEEZER and **u**nified **s**imilarity **m**easure SQUEEZER [48]

Both algorithms are extensions of SQUEEZER [50], designed to handle, besides categorical data, numerical and mixed data. In the first variation, *dSQUEEZER*, the numbers are converted into categories by taking them by intervals. So then, any numerical attribute is discretized. The algorithm has few variations with regards to the original categorical algorithm.

The second variation, *usmSQUEEZER*, is a bit more complex. Some changes have to be done in the *Summary* definition for each cluster. Before, the summary gathered relevant information of each cluster: the number of attributes, and pairs indicating the values of those attributes and their support. For this mixed data version, the changes will include the consideration that now, the first elements of the data are categorical while the rest are numerical. For the categorical values, the information stored is the same as in the original one and for the numerical attributes, their mean values are kept. The new measure, the Unified Similarity Measure, has a term for the categorical attributes, same as SQUEEZER and a new term based on the same *support* concept, for numerical attributes. Then, the numerical attributes must be normalized, to make their contributions be between 0 and 1, the same as any categorical contribution is. Using this new measure, the SQUEEZER algorithm can be directly applied to mixed data sets, always with the consideration that the

data set attributes have to be reordered, to have first the categorical ones followed by the numerical part.

The advantage of these two variations is that they can be used for large data sets, sharing this property with the SQUEEZER algorithm where they are developed from. Disadvantages are not many, but when the data set is not large, other algorithms offer much better results.

◇ **Probability algorithm based on k-means**, [3].

This algorithm is based on k-prototypes [26], which is the mixed version of k-means [21]. It modifies that cost function to overcome some issues related to the clustering of mixed data. Like in k-prototypes, in the new cost function, the computation of the distances for numerical and categorical data are separated into two different terms. So is the prototype of each cluster. The differences with respect to the original algorithm are mainly three:

- The distance between two categorical attributes is not binary (0 or 1) depending if the two values match or not. It now depends on the $co - occurrence$ of the attribute within a single class and in the database.
- The weights which define the importance of each attribute are not user-assigned. They are computed in the categorical case by the co-occurrence and in the numerical case by the distribution.
- The prototype of each cluster is now: the mean value of the numerical attributes and for the categorical ones, instead of the mode, it will be a proportional distribution of all their values in the cluster.

Given that the categorical distance now defined includes the distribution of all the data set, this measure has embedded the significance of each categorical attribute. On the numerical side, the significances is computed dividing the range into several intervals (discretization) and computing the co-occurrences of those numerical attributes.

The steps in this algorithm are:

1. Data elements are assigned to one of the k clusters randomly.
2. Find the center of each cluster.
3. For each object, find the center which is closest. Assign it to that cluster.
4. Re-compute cluster centers with new distribution.
5. Repeat 3) and 4) until cluster assignments do not change for a fixed number of times.

Table 3.2: *Database with 4 objects and ordinal codification*

Object	Description
A	{1,2,3,5}
B	{2,3,4,5}
C	{1,4}
D	{6}

This algorithm has been tested on conventional benchmarks, only numerical, only categorical and mixed and its performance has been proved to be good. The main disadvantage is that it is quite complicated to apply and a good statistics basis is required.

3.3.2 Data Conversion

As introduced before, there is a technique to work with mixed data which is not based in any mixed algorithm. It is based on data transformations, either numerical data are transformed to categorical in order to use a categorical algorithm or categorical data are transformed to numerical in order to use a numerical algorithm.

The second case is the most popular one, mainly for two reasons: first, the numerical algorithms have been better developed and, at present, they provide more accurate results. Second, numerical algorithms are much easier to be interpreted than categorical. While the Euclidean distance is a *physical* distance and is easily interpretable, it is quite difficult to understand and interpret the categorical similarity measure, not to mention the difficulty to represent them.

Then, in most of these applications, what will be found is a transformation of the categories into numerical values, although here both approaches will be described. Several ways of converting categories or symbolic data to numbers can be found in the literature, one of which consists of representing each object by a vector in which 0 means the category is not present for that object while 1 means the category is present in that object. But this way of codification can have very serious drawbacks. Let's explain it with an example.

Consider a database with 4 objects like in Table 3.2. The numbers indicate the categories that are present in each of them, no matter now what these categories symbolize. Assigning 1s and 0s in the way explained before, Table 3.3 shows the transformed objects.

If now the database is to be clustered into two clusters, the Euclidean distance will be used as similarity distance. It can be obeserved that the pairs of objects with

Table 3.3: *Transformed database*

Object	Transformation
A	{1,1,1,0,1,0}
B	{0,1,1,1,1,0}
C	{1,0,0,1,0,0}
D	{0,0,0,0,0,1}

Table 3.4: *Categorical attribute with 4 categories*

Color	Transformation
green	1
blue	2
red	3
yellow	4

smaller distance are A and B, where $d(A, B) = \sqrt{2}$ and C and D where $d(C, D) = \sqrt{3}$. However, it can be seen that C and D do **not** share any element. Then, we can reach the conclusion that using this binary (there is/there is not) transformation, some objects that in any natural way would form a cluster, are grouped together. It has to be said that this example is one of the most difficult ones, given that the data set is quite complex. The databases used for clustering purposes do not follow this pattern. Usually, all the instances have the same number of attributes, even though the values adopted in each case can differ quite a lot. Therefore, the data base in Table 3.2 is not very representative of the format used in clustering. However, this could be an example to show a case of presence/absence of objects and it has been seen how badly classified it is, using the normal metric.

For binary attributes, this codification would be perfect, as in that particular case it is true that objects either are or are not (*Yes,No*), or either are X or are Y (*Female,Male*) and in any case there are no more possibilities.

There are other ways of codifying categories. Some of them try to overcome this false grouping. One is, given that the categories are by themselves discrete values, to assign an entire number to each one. This means that if a categorical attribute has a domain of 4 categories, the conversion will be done assigning {1,2,3,4} to these categories [8]. In Table 3.4, there is an example.

The advantage of this transformation is that is straightforward and easy to apply. The main problem of this method is that independent and with no specific order values are transformed in dependent and ordered values, i.e. in categorical values all them are

Table 3.5: *Gray transformation for an attribute with 4 categories*

Original	$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
Green	1	0	0	0
Blue	0	1	0	0
Red	0	0	1	0
Yellow	0	0	0	1

equally different to the others, however here 3 (red in the example) will be closer to 4 (yellow) than to 1 (green), introducing a closeness and a relation that are not true in real data.

Another transformation, similar to the first one explained is that based on the gray code, [25]. The transformation will be done with 1s and 0s as the first case, but instead of replacing attributes, they will replace categories, i.e, in the new data base, each attribute will be transformed to N attributes where N is the length of the domain of $Attr_i$ and each one of these new attributes will be 1 or 0 to indicate that it is so. Let's take the example of the attribute in Table 3.4 with 4 categories that could represent, for example, the car colors people prefer. The new attribute converted and codified would be like shown in Table 3.5.

This method assures the independency between the data and does not introduce any type of ordering. Nevertheless, the main problem can be the dimensionality of the transformed matrix, given that attributes with large number of categories would result in large number of new transformed attributes. Let's consider this case. If the database had 5 attributes each one with 10 categories, the database would be transformed to a 50 input database, which is not easy to handle by any algorithm. Besides, the transformed data will not have null mean and unitary variance, good statistical property.

Another codification for categorical data can be done using the binary code. Table 3.6 shows how an attribute with 4 categories could be codified using this code. Here the advantages and drawbacks of this code, place it in an intermediate point between the ordinal codification and the gray code. The advantage, as in the ordinal code, is that the number of transformed attributes is not as high as when using gray code. However, a similar drawback is present, an attribute is closer to some and further from some others, when this distance should be the same in all cases. It is true, that the relation is not as direct as in the ordinal case, but it does exist in some cases.

Given that the important property of the categorical attributes is the independency, which involves no fix way of ordering them, this is what the numerical codification applied must keep. Then, thinking of Euclidean distance, which is the one used by most of the numerical algorithms, the proposed spatial points must have the

Table 3.6: *Binary transformation for an attribute with 4 categories*

Color	Transformation
green	00
blue	01
red	10
yellow	11

property of being *equidistant*. This is easy to meet in the two dimensional case, as shows Figure 3.5, where it can be seen an isosceles triangle inside the unity circle. Although these points have the same distance between them, it is easy to see that in the two-dimensional case only three points can be represented.

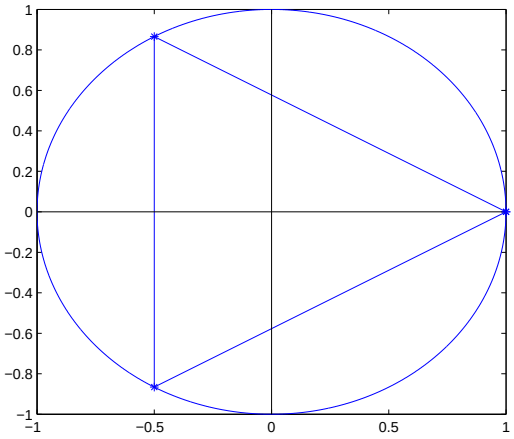

Figure 3.5: *2D case of equally distanced points*

The 3D case, a pyramid with triangular base is represented in Figure 3.6, where the base is still within the unity circle. Now attributes with four categories can be represented. However, it is obvious that the dimension is always $N-1$ for an attribute with N possible values, which means that the same problem of a large number of transformed attributes is appears here. Besides, it is not straightforward the calculation of the points in any T-dimensional space if T>3.

So far, different techniques to apply the categorical to numerical codification have been described. Although they are not as common as the ones mentioned before, there are also techniques to codify the numerical values to categorical and they basically depend on the nature of the number. If the number is discrete, it is quite easy to assign to each of them a category, that could be represented by a letter (A,B,C,D,...). The problem of this

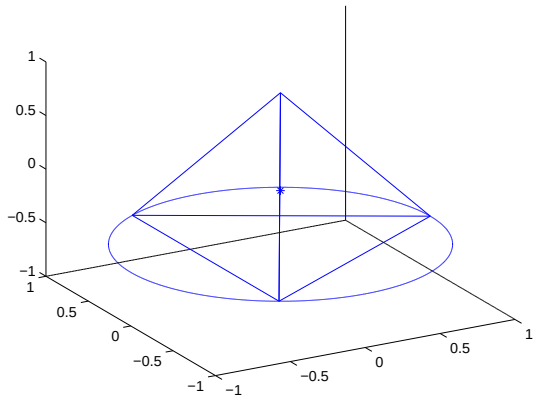

Figure 3.6: *3D case of equally distanced points*

Table 3.7: *Temperature codification to categorical*

Temperature interval (C)	$Attr_1$
$[-30, -10[$	A
$[-10, 10[$	B
$[10, 30[$	C
$[30, 50]$	D

is that if the attribute can adopt a large list of discrete numbers (for instance, people's age) the number of categories will be extremely large, and even if the dimensionality of the database does not increase, a large number of different attributes is always difficult to work with.

Nevertheless, if the number is continuous this technique is impossible to be used and then the most common one is the consideration of intervals as categories, [48]. These interval adoptions can be also useful when the number of categories, even if finite, is very large. The example of Table 3.7 represents how to codify as categorical the room temperature.

The main drawback of this approach is that, it is easy to see, the transformed numerical data lose information. Here, organized numbers are considered independent and accuracy is being obviated. Just the opposite to the categorical transformation, where information, false information, was being added. Another problem, which was mentioned before, is that categorical algorithms have not been as thoroughly developed as numerical ones. In addition to this, interpretability is much more difficult in symbolic representation.

All these drawbacks are the reasons why, when codification is taken as the option to work with mixed data, categorical attributes are codified to numerical values and a numerical algorithm is applied.

3.4 Summary

Nowadays, the data sets are not composed by either only numerical or only categorical attributes, so all the algorithms that only work with one type of data have become obsolete and the need of mixed data-type clustering techniques is obvious. The revision of the existent algorithms that work with both data has pointed that their development has not reached yet an optimal point and therefore their performance could be improved.

It has also been reviewed another technique based on the codification of the attributes of one of the data-type to the other in order to use only one-type algorithm. Several codifications have been described with their advantages and drawbacks.

Therefore, there exist a need to find a new approach which overcomes the problems found until now or, at least, that works better that the ones explained in the previous section. This is one of the goals of this work, and the proposed approach will be described in next chapter.

4. Proposed Approach for Mixed Data Clustering

This chapter aims at presenting a new approach to work in an efficient way when clustering mixed data, given that neither the algorithms nor the approaches found so far in the literature offer a high performance in these cases. The new approach is based on a new codification of the categorical data to allow them to be treated by any numerical algorithm. Here, the numerical algorithm used will be the well-known *k-means*, because the goal of this chapter is not to design an algorithm, but to test how the performance of the new codification approach is. Thus, the usage of one of the better-known and understood algorithms is quite convenient.

This document will describe the proposed codification and the reasons why this has been adopted. Later, there will be two more sections. First, the proposed codification will be applied and compared with other codifications, all them described at the end of last chapter. Next, an introduction of a different weight for each attribute to give them relative importance in the clustering process is done. With this, it will be seen how a simple algorithm with a good codification can have a very good performance.

4.1 Methodology

The steps into which we can divide the proposed approach application are, in a general case, **four**. The first two of them correspond to the preprocessing of the data, which is very important in order to get a database of quality where the clustering process can be applied. The third one corresponds to the modification of the algorithm and the introduction of weights to each attribute. The last one corresponds to the final integration of all the steps and the application of the algorithm.

4.1.1 Codification

As mentioned before, the numerical clustering algorithms have been more thoroughly developed than the categorical or mixed ones, so it seems a good approach to use them. If they are to work well on mixed data, categorical attributes must be codified in such a way that they have, after codification, the desired characteristics (*independence* is the most important one).

Basic approach: POLAR CODE

The proposed approach is based on assigning a pair of polar coordinates, within the unity circle, to each category. Then, the circle will have to be divided into as many *equally distanced* points as the number of categories a particular attribute has. Here *equally distanced* points means that the distance between two neighbour points is always the same, property that regular polygons have. The advantages of this new codifications are mainly three:

- ◇ The first is that, no matter how large the number of categories is, the codified attribute will be a vector of two elements. So, in the worst case when all data are categorical, the database will double its original dimensions.
- ◇ The second is that placing all categories around the unity circle, the data mean will be null, which is a good statistical property when working with numbers. Besides, adjusting the radius of the circle R, it can be achieved an unitary variance, also desired from the statistical point of view.
- ◇ Finally, the third advantage is that all the categories will have the same characteristics respect to the others, i.e. all attributes are in equal conditions with regards to the whole group, getting in this way one characteristic needed in categorical values.

The disposition of the points in a circle makes them not be all equally spaced, but all them have the same properties and they will be balanced. This can be better understood with an example. Looking at Figure 4.1, it can be seen a graphical representation when an attribute can adopt six different categories. In that case the circle is divided into six points equally spaced. It is obvious that the distances from one point to another in the circle are not all the same. However, it can be seen that all the points have the same characteristics, they have two next neighbours, two following neighbours and so on, and they all follow the same pattern. This is what was meant by *balanced* transformation. Besides, if the points are assigned randomly to each category,

all them will be placed in an arbitrary place with regards to the others, helping this to the dependence of each.

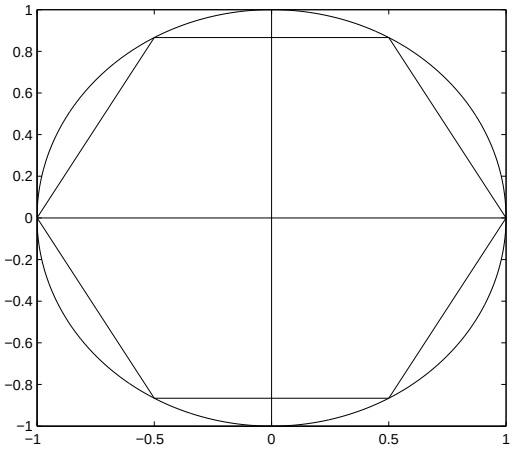

Figure 4.1: *Case when 6 categories are possible and the circle is divided into 6 points*

Mathematically, the new values can be represented as:

$$Cat_i = \left[\cos\left(\frac{(i-1)\cdot 2\cdot \pi}{N}\right) \quad \sin\left(\frac{(i-1)\cdot 2\cdot \pi}{N}\right) \right] \tag{4.1}$$

where N is the number of different categories that the attribute can take and i is the category index.

This way is as easy as the ones explained before, see Section 3.3.2, for converting categorical data to numerical and, even though it is not perfect, experiments done and shown in Section 4.3 will show the good performance of it.

An exception for the conversion of one categorical input into two numerical ones is when the categorical input is **binary**. In that case there is no need for two new coordinates but for one, given that the division of the circle into two points will have both of them projected onto the same axis, Figure 4.2. So, then the other coordinate is in both cases null and can be removed.

Codification Error

It can be observed that the larger the number of categories an attribute has, the larger the number of segments to divide the circle into. This will cause an increment in the

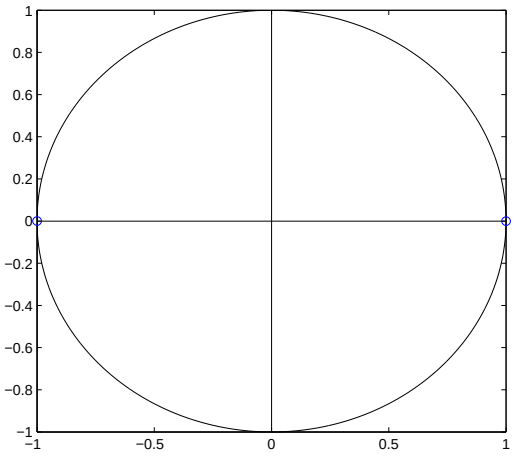

Figure 4.2: *Case when the attribute is binary*

codification error, meaning by *error* the difference between the largest distance and the shortest one. For example, in the case of Figure 4.1, largest distance is the diameter $2R$ while the shortest is the the side, which is R. Therefore, the error will be $error = 2R - R = R$.

In a general case, it can be considered that the error will be the diameter of the circle ($2R$) minus the side of the polygon:

$$l = 2R \sin\left(\frac{\pi}{N}\right) \tag{4.2}$$

This is true, when the polygon has an even number of points, because then and only then the maximum distance will be the diameter. If the number of points is odd, this will not be true, but in any case, a good estimation of the maximum error carried out is:

$$error = R - 2R \sin\left(\frac{\pi}{N}\right) \tag{4.3}$$

The dependence of the error on the radius can be noticed. Obviously, if $R = 0$ the error will be zero, but this is the trivial solution. For any other fixed radius, the error committed will increase with N.

For two or three categories, this codification will have no error, because all the points will be equidistant from one to another. If $N = 2$, the result of eq. (4.3) is

Table 4.1: *Coordinates of the platonic solids*

Tetrahedron	Octahedron	Hexahedron	Icosahedron
$(1, 1, 1)/\sqrt{3}$	$(\pm1, 0, 0)$	$(\pm1, \pm1, \pm1)/\sqrt{3}$	$(0, \pm1, \pm\varphi)/r_{ico}$
$(-1, -1, 1)/\sqrt{3}$	$(0, \pm1, 0)$		$(\pm1, \pm\varphi, 0)/r_{ico}$
$(-1, 1, -1)/\sqrt{3}$	$(0, 0, \pm1)$		$(\pm\varphi, 0, \pm1)/r_{ico}$
$(1, -1, -1)/\sqrt{3}$			$r_{ico} = \sqrt{1+\varphi^2}$

$error = 0$. If $N = 3$, eq. (4.3) does not show the real error, because three is an odd number and eq. (4.3) is, as said, an approximation.

However, when the number of categories is more than three, some points will start being closer or further to others. If the number of points is not large, the error will be quite small. However, for a large number of categories, the error will be quite large and therefore this codification might not be appropriate. It is the user by their knowledge of the application who has to decide which error is admitted in each case.

Extended approach: SPHERICAL CODE

It has already been said that the larger the number of points to divide the circle into, the larger the committed error will be. It is clear that the proposed codification is not very accurate to codify attributes with a large number of categories. In the case the committed error is too large, and it will depend on the user and the application to decide when it is so, an option is to change the polar codification to the spherical one, passing the 2D domain conversion to 3D, reducing thus the error. In the 3D case, the number of points placed with no error increases to four, Figure 3.6 shows this, a pyramid with triangular base.

In order to distribute the points in the sphere, the procedure is not as straightforward as in the circle. Now, to divide the sphere into a certain number of points N, this N can not be any number. The regular polyhedra in 3D correspond to the name of *platonic solids* and they and their number of vertices are: the tetrahedron ($V = 4$), the octahedron ($V = 6$), the hexahedron ($V = 8$), the icosahedron ($V = 12$), the dodecahedron ($V = 20$) and the truncated icosahedron ($V = 60$). The coordinates for each solid can be seen in Tables 4.1 and 4.2, and have been normalized for a sphere with unity radius.

In these two tables it is seen how the coordinates of the vertices of each figure depend on the polyhedra itself and there is not a general equation to describe all placements. The coordinates for each of the vertices have determined values for each polyhedra and most of them now depend on the *golden ratio* (φ):

Table 4.2: *Coordinates of the platonic solids II*

Dodecahedron	Truncated-Icosahedron
$(\pm1, \pm1, \pm1)/\sqrt{3}$	$(0, \pm1, \pm3\varphi)/r_{tico}$
$(0, \pm1/\varphi, \pm\varphi)/\sqrt{3}$	$(\pm1, \pm3\varphi, 0)/r_{tico}$
$(\pm1/\varphi, \pm\varphi, 0)/\sqrt{3}$	$(\pm3\varphi, 0, \pm1)/r_{tico}$
$(\pm\varphi, 0, \pm1/\varphi)/\sqrt{3}$	$(\pm2, \pm(1+2\varphi), \pm\varphi)/r_{tico}$
	$(\pm(1+2\varphi), \pm\varphi, \pm2)/r_{tico}$
	$(\pm\varphi, \pm2, \pm(1+2\varphi))/r_{tico}$
	$(\pm1, \pm(2+\varphi), \pm2\varphi)/r_{tico}$
	$(\pm(2+\varphi), \pm2\varphi, \pm1)/r_{tico}$
	$(\pm2\varphi, \pm1, \pm(2+\varphi))/r_{tico}$
	$r_{tico} = \sqrt{9\varphi + 10}$

$$\varphi = \frac{1+\sqrt{5}}{2} \tag{4.4}$$

As the number of vertices can not be any number, for codifying an attribute with N categories the regular polyhedra with the number of vertices equal or larger than this N will have to be used . Then, it can happen that the number of points exceed the number of categories. In this case what should be tried is to place the points in a way that the mean is null. This will be in such way if, and only if, the number of categories is even, given that all polyhedra have symmetric vertices. If the number of categories is odd, there will be one vertex which imbalances the total mean, but the effect will not be big, as the values of the vertices are all quite small.

The spherical code is a 3D extension of the 2D polar code so that, even when not all the vertices of the figure are taken for the codification, the error committed will be smaller, meaning by *error* again the difference between the largest and the shortest distance. This should help the clustering process when the number of categories is large, and in many cases it does. It will be seen in this same chapter in a later section how this extension always has similar or better results than those of the polar coordinates.

The error in this case has not a determined equation. However, the conclusions obtained for the 2D case, can be extended to this: the error will increase with the increment of the number of vertices in the polyhedra. For each polyhedra its error could be computed, just finding the distance between two neighbours vertices and the maximum distance which will be in many cases the diameter of the sphere.

Table 4.3: *Comparison of the several codification methods*

	Gray	**Binary**	**Ordinal**	**Polar/Spher**	**Spatial**
N	Large	Small	Small	Small	Large
Mean	Non null	Non null	Non null	Null	Null
Variance	Not unity	Not unity	Not unity	Unity ($f(R)$)	Unity
Relative error	0	$\sqrt{N}$	Large	$2 - 2\sin\left(\frac{\pi}{N}\right)$ / ??	0
Control of error	No	No	No	Yes	Yes
Comput. load	High	Low	Low	Low	High
Calculations	Easy	Easy	Easy	Easy/Medium	Difficult

Higher order coordinates

If the error in spherical coordinates is considered to be too large, and this is again a user and application-based decision, then higher order coordinates could be used to reduce it. Nevertheless, finding the vertices when $Dim > 3$ is not easy at all. The regular distributions of points in higher dimensions are called **regular-polytopes**. This is something that goes far away from the aim of this work and will not be considered here.

Besides, as it will be seen in next sections, the application of the polar and spherical code work quite well for many applications and therefore finding a higher order codification is an extra work that might not be necessary since the already explained codes have such a good performance.

Characteristics of the codifications approaches

Many different approaches for codification were described in the previous Chapter, Section 3.3.2. It seems a good idea, therefore, to make a comparison of all of them in terms of the most important characteristics that any codification method should meet. Table 4.3 shows the main characteristics of the codification methods mentioned before, which are the main methods found in literature, and the new proposed approach.

It can be seen that the approach here suggested is far from being perfect, but it represents the best trade-off between error, statistical properties of the codified variables and low computational cost. That is the reason why it is considered a good choice for the data conversion for clustering purposes.

4.1.2 Normalization

Once the categorical attributes have been codified, all of them are within the unity circle/sphere interval, which means that all coordinates are within the [-1,1] interval. Numerical attributes, however, will have different ranges and will be measured in different unities and both issues can influence quite a lot in the clustering process, explained in Section 1.3.

In this case, given that the categorical attributes will be codified using the explained codification and their transformed values will be in a pre-defined interval, it seems more logic to use the normalization by interval and choose the extremes of the unity circle/sphere as the the limits of the interval. Figure 1.1 shows graphically how the linear normalization will be, while equation (1.17) is the relation to apply.

Obviously, the categorical attributes codified to numerical values using the codification explained will have all their values within this interval and therefore will not have to be normalized.

Once categorical attributes have been codified and the rest have been normalized, the data set is ready for the clustering process.

4.1.3 Introduction of weights in the cost function

Once codified, all the attributes have the same importance *a priori* for the clustering algorithm. However, in most of the applications not all the characteristics have the same relevance for the clustering process The relevance does not come from the unities and most of the times it is embedded in the data and hidden. In those cases it is the expert of the system the one to help to get this information out of the database. Another possible way, if the expert is not available or the information proportioned is not enough is to consider a loop of several weights in several attributes and see which combination of them gives better results. This is also useful to discover importance in some attributes that were unknown, even if the expert provided information. If no information is available, the weights can initialized to the same value, usually 1 for all them.

The weights are directly introduced in the cost function, in fact, directly in the distance computation, to make the distances with larger weights influence more in the global function and then the algorithm will put bigger effort in minimizing them. Equation (1.19) shows how is the application of weights to the distance between two objects. There is a weight ω_v, $v = 1, .., z$, for each of the z attributes of the database.

These weights will be introduced, as an example, in the cost function of the k-means algorithm, which will be the one used to show the performance of the proposed codification approach. Equation (4.5) shows the cost function of the weighed algorithm, where $\vec{\omega}$ is the vector of weights the user will set or define based on knowledge of the system or initialized to 1:

$$J_{weighed} = \sum_{i=1}^{c} \left(\sum_{k=1}^{n} \vec{\omega} \cdot d\left(x_k, c_i\right) \right) \tag{4.5}$$

Including the weights in eq. (2.2), the final cost function for the k-means algorithm is obtained:

$$J_{weighed} = \sum_{i=1}^{c} \left(\sum_{k=1}^{n} \sqrt{\sum_{v=1}^{z} \omega_v \cdot \left(x_{k,i,v} - c_{i,v}\right)^2} \right) \tag{4.6}$$

Therefore the weights are defined in a vectorial form ($\vec{\omega}$) with the same length as the number of inputs, after the codification, where larger weights make the algorithm try to minimize the distance within each cluster for that particular attribute more than for the rest with lower weights. It is quite important to apply the same weight to all the codified inputs that come from the transformation of a categorical input into polar/spherical coordinates, because just one of the inputs of that pair has no meaning on its own.

4.2 Comparative study of codification approaches

In order to check the performance of the new proposed codification several benchmarks have been chosen. The data sets were provided by *UCI Repository of machine learning databases* [34], web site http://archive.ics.uci.edu/ml/. From all the possible data sets in the repository, those that were considered appropriate for this application, were selected. Many of selected databases present both types of attributes in different proportions, and two of them only categorical attributes.

Before showing the experiments done and their results, there is an explanation of the considerations taken to perform the experiments, necessary to get quality results.

In the following subsections the databases will be briefly described and details about the number of objects, the number of attributes and their types and the number of clusters at the output will be given. Results of the performance of different codifications

and with the proposed one will be presented, always using the algorithm k-means. Finally, the results will be analyzed over the whole comparative.

4.2.1 Considerations

Before doing the experiments, some pre-processing of the data should be done, to make the benchmarks have certain properties needed for clustering. First of all codification and normalization must be applied.

All codifications have been applied in the way they were described. Only the spherical code has suffered slight modifications, because when the attributes had two or three categories, there was no need for a 3D conversion and 1D and 2D conversion where taken respectively: line and isosceles triangle.

It is also important to define a way to treat the missing values, given that many of the found benchmarks are not complete. Here, it has been decided than any object that has one or more missing values will be removed from the data set. Another possibility could be to replace the missing value by the mean of the attribute. However, this can lead to very unrealistic objects and, therefore, to inaccurate results. The most accurate options is to omit these values as nobody is sure of them.

The similarity measure does not have to be defined, as the algorithm for the clustering is weighed k-means and it already has a similarity measure defined, equation (4.6). However, as no information is included about the importance of the characteristics in this first study, the vector of weights was initialized to 1, $\vec{\omega} = \vec{1}$, resulting then the normal k-means cost function, eq (2.2).

Given that the k-means algorithm starts randomly, it does not always offer the same result. If a experiment is run many times, there is a result which appears more often, this result corresponds to the minimum index, considering this as expressed before, and this is the value that has been taken as the *right* value of the clustering process using this algorithm, as far as it appears more than the 60% of the times, which has been considered the minimum level of times for a result to be representative of the solution.

For these reasons, to be sure the obtained result is the one corresponding to the minimum addition of distances, all experiments have been run 200 times (a large enough number of times). The results have been taken of the most repeated value and of the computational time to perform all repetitions. As the two last database are large, the number of repetitions is just 100, to make them not so slow. Results, however, are as conclusive as in the other cases.

Table 4.4: *Comparison of some codifications for the Vote data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	232 × 16	89.66%	2.3s
Binary	232 × 16	89.66%	2.4s
Gray	232 × 32	89.66%	2.8s
Polar	232 × 16	89.66%	2.3s

In order to compare solutions in a reliable way, not only the number of well clustered objects is considered for each case (heading: **Results (%)**). The computing time the whole set of repetitions last in each case and the size of the codified matrix are also taken into account (heading: **Time (s)**), being factors of less importance than the first, but also important to be considered.

4.2.2 Vote Database

The Vote database is a fully categorical one which is composed by 435 objects with 16 binary attributes each. Only 232 objects are complete. The output is divided into 2 clusters corresponding to either the Democrat (124) or the Republican (108) voters. The proposed codification will have to be applied to all the attributes. As seen in the previous section the new codified inputs will not have two coordinates, but only one which will be either $\{\cos(0),\cos(\pi)\} \equiv \{1,-1\}$. Results of all the experiments using the different codifications are shown in Table 4.4.

It can be seen that all experiments give same results in terms of well clustered objects and very similar ones in terms of computation time. This is due to the fact that all attributes are binary and every codification but the gray code assign just two values ($\{1,2\}$ when ordinal, $\{0,1\}$ when binary and $\{-1,1\}$ in polars) and then, after normalization, all codified matrices result in the same normalized values. When the gray code is used the codified values are $\{[0\ 1], [1\ 0]\}$, this means that the resulting codified matrix doubles the size of the original matrix. However, as the database is not large, the computing time is similar enough and the results are exactly the same as in the other cases.

It can be concluded that the polar codification does not offer any advantages in this case, but no codification is clearly superior to the others as this is a very simple case. On the other hand, it is also true that the proposed approach can not be said to be worse than any of the others.

Table 4.5: *Comparison of some codifications for the Heart Disease data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	296 × 13	71.2%	2.3s
Binary	296 × 17	80.7%	3.2s
Gray	296 × 25	81.8%	3.4s
Polar	296 × 17	82.4%	3.2s
Spherical	296 × 18	81.1%	3.0s

4.2.3 Heart Disease Database

The previous data set was fully categorical. The heart disease database is, however, a mixed database, where there are a total of 13 attributes, 7 of which are categorical and the rest, 6, are numerical. That means that the mixture of types of data is quite balanced. There are 303 objects, 296 of which do not have missing values and are good for clustering. Of those, 136 correspond to sick patients (with heart disease) and the rest, 160, to normal people. The number of possible categories is not the same in all the categorical attributes but each of them will be codified applying equation (4.1) adopting the proper N for each case. Therefore, the length of the objects will increase after using the codification and the final length will be 17, because only 4 of the 7 categorical values have more than two possible categories and they double their size. Results can be seen in Table 4.5.

It can be seen now how the results using the different codification differ from one another more than in the Vote database case. This makes sense because now, after codified, the matrix and its size are different in each case. The gray code, the binary code and the polar coordinates offer similar results. The ordinal code, however, has a worse performance than the others, although is the smallest codified matrix and then the fastest to compute.

Comparing the four codifications that have similar results it can be observed that the performance of the polar approach is slightly better than the other three and it is even faster than the one using the gray code, second best. Here spherical code, does not improve the results, but they are around the same values. All computing times are small because all resulting data sets are not very large, but even with these few instances it can already be seen some differences with the timing, being the polar codification approach in the middle.

So, in this case, the proposed approach offers a result as good as the best case and a computing time in middle terms.

Table 4.6: *Comparison of some codifications for the Credit Approval data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	653 × 15	86.4%	3.8s
Binary	653 × 25	54.2%	6.6s
Gray	653 × 46	54.2%	6.4s
Polar	653 × 20	86.4%	4.7s
Spherical	653 × 23	86.4%	5.1s

4.2.4 Credit approval Database

This database is also a mixed one with a total of 690 objects. After removing the objects with missing values, there are 653 objects available. The database has 15 attributes, where 9 are categorical and 6 are numerical. Categorical attributes now can adopt from 2 to 14 categories, but the number of new attributes will be two for each categorical attribute, but when the number of categories is 2. So the only drawback of the large number of categories of some attributes is that the codification error will be larger. The output reflects whether the credit card was approved for each person or not and, in 307 cases it was and in 383 it was not. After the codification the length has become 20 instead of the initial 15, as 5 of the 9 categorical attributes can adopt more than 2 categories. Results can be seen in Table 4.6.

These results differ very much from the ones of the heart disease database. Here, in opposition to the other experiment, the ordinal codification is the one that offers the best clustering results. The other two, binary and gray codes, have a very poor performance in this case. The polar codification works as well as the ordinal one, offering, again, the best results of all codifications. Extension to 3D, spherical coordinates, does not improve the results, offering exactly the same. Again computing time is not large in any of the cases, being the ordinal codification the fastest and the polar in an intermediate term.

4.2.5 Cylinder Band Database

This database is also a mixed one with a total of 512 objects. However, it has many missing values and instances with unknown attributes have been deleted. The resulting database has a total of 277 instances. They are not many, but the quantity of attributes is large (40). Some of them have been removed because they do not contribute with any extra information, leaving 34 attributes to be considered. Finally, the database has 21 numerical attributes and 13 categorical. As in the credit approval database the number of

Table 4.7: *Comparison of some codifications for the Type of Cylinder data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	277 × 34	62.82%	3.6s
Binary	277 × 42	60.3%	3.7s
Gray	277 × 63	62.45%	5.6s
Polar	277 × 40	63.18%	3.4s
Spherical	277 × 47	64.26%	4.0s

categories which each categorical attribute can adopt varies between 2 and 8, being only one attribute with 8 different categories. The others have between 2 and 5 being then the committed error not very large using polar coordinates for codification. The output can adopt three different categories, adding this a bit of difficulty to the clustering process. Results using different codifications can be seen in Table 4.7.

None of the codification approaches offers a very good result and this might be due to the existence of three clusters in the output. Nevertheless, all them have a very similar behaviour, in percentage of well clustered instances and in computation time. Again, as in the previous experiments done, the *polar* codification offers a slightly better result than the others and the computation time is not large due to the medium size of the resulting database. In this case the spherical codification does offer a slightly better result than the polar approach and, although it is no very significant the increment of performance, it is noticeable. These results confirm that the the proposed approach works slightly better than the others.

4.2.6 Mushrooms Database

This database, as the Vote one, is also a fully categorical. Nevertheless, not all attributes are binary and then the different codifications offer different values for the objects. It has only missing values in one of the attributes which has been fully removed. Finally, the database has 20 categorical attributes and the output is divided into two groups: edible and poisonous mushrooms. The number of objects is much larger than in the previous cases, being 8416 the total of objects to be classified. The number of different categories for each of the attributes varies between 2 and 12, having many of them 9 different options. The error is not huge but can be considered relatively large. This is the reason why it has been considered appropriate to apply to spherical coordinates too. Results of the experiments are shown in Table 4.8.

Now the results are not as similar to each other as in the previous case, but they are better, being many of them quite good. Here the *ordinal* codification differs

Table 4.8: *Comparison of some codifications for the Mushrooms data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	8416 × 20	73%	100s
Binary	8416 × 51	88.91%	160s
Gray	8416 × 111	89.5%	265s
Polar	8416 × 35	88.78%	180s
Spherical	8416 × 49	88.81%	140s

quite a bit from the others, which have all similar results. Now, the computing times are also very different between these experiments. The ordinal codification is the fastest one, but also the worst clustering result, therefore, time is not relevant in this case. Now the codification with better results is the gray code. However, it can be seen that the computing time is much larger than in the other cases, given that the data set is large and the resulting matrix is more than twice any of the others codified matrices. The results with the polar codification is not exactly as good as when the gray code is used, but it is very similar (less than 0.8% different) and the computing time is significantly smaller.

As in many cases, the application of the spherical codification offers better results but does not improves much the performance of the polar code. This small increment of the performance level goes contrary to expectation and can be due to the importance of the attributes. Nevertheless, both the polar and the spherical codifications have a good performance at a low computation time, being their results very good, attending to the three properties.

4.2.7 Salary Database

This is large database with mixed attributes. After removing the objects with missing values the total of instances to be clustered is 30161, a number that makes this database much bigger than the others. It has 14 attributes, 6 of which are numerical and 8 are categorical. It is a quite balance mixed set, therefore. The number of categories of the several categorical characteristics varies between 2 categories and 41, although there is just one attribute with so many possibilities. The normal number of categories are 6 and 14. It is straightforward that the error when applying polar coordinates will be very large and that spherical coordinates will reduce the committed error of the codification. However, the attribute with 41 categories will have a large error with both codifications. There are two clusters, people who has a salary bigger than 50K dollars per year and people whose salary does not exceed this quantity. Table 4.9 shows all results of these experiments.

Table 4.9: *Comparison of some codifications for the Salary limits data set*

Codification	**Database size**	**Results (%)**	**Time (s)**
Ordinal	30161 × 14	50.05%	284s
Binary	30161 × 33	71.58%	450s
Gray	30161 × 105	71.2%	495s
Polar	30161 × 21	50.05%	290s
Spherical	30161 × 28	75.36%	440s

After a quick look it is seen that none of the databases offers an extraordinary performance, although some of them reach a relatively medium-good result. The ordinal and the polar approaches offer the worst results of all, differing quite a lot from the others. The binary and the gray codes offer a much better result and very similar to one another. As mentioned before, the error of some codified attributes with polar codification is large due to the number of categories that can take, especially in that case of 14 possible values. Then, spherical coordinates can be used in order to improve the performance. It can be seen in the results summary how the spherical coordinates offer a better result than any of the other codifications and much better than the polar and ordinal codes. In opposition to the mushrooms database, it seems that in this case the attributes with a large number of categories do have a big importance in the clustering process and this is the reason why polar code does not work very well here and spherical coordinates do improve the results.

The computation time is large, due to the size of the database Differences can be observed in the magnitude of this time, being the fastest approaches also the worst ones in terms of results. The ones with larger time also offer a much better result in the clustering matching, being the spherical coordinates approach the best with the smallest time of the group of the codifications with good results.

4.2.8 Analysis of the results

In all experiments done it has been showed the results of many codifications approaches. It can be concluded that no one of them works clearly much better than the others, as the results are very similar in most of the cases and the one that works better in each of the cases is not by far. Besides, the best solution in one experiment can also be the worst in another.

It has been proved that the proposed approach based on polar coordinates works well in most of the cases and it is true that other codifications are comparable to ours in each experiment but the proposed one is the one that works as well as the best one in all the cases. There is only one case, the last one of Salary database, where the polar

codification does not work well. This is due to the large error committed when using it. Using the spherical coordinates the so large error is avoided and the results are improved. In many cases it has been observed that the usage of spherical coordinates does not improve the performance of the experiment, but in those cases the results are not worse and if they are, only slightly, of those obtained with polar coordinates. This is due to the importance of such attributes.

Then, it can be said that the polar codification and the spherical one when the committed error is large, offer a very good approach to the codification problem for clustering purposes, given that the results obtained with this codification are very good and always the best ones when using data transformation.

As the resulting converted matrices do not increase their sizes much regarding the original database, the computing time is one of the smallest one in all experiments using codifications. This is another advantage of the proposed codification.

Definitively, this section has shown how the proposed approach is reliable in performance term and compared to others codifications is the best one to adopt. In next section, it will be shown how the adoption of this codification and the modified k-means algorithm can lead to results better than or as good as complex algorithms, being much easier to apply than those ones.

4.3 Comparison with other algorithms

This section is a further analysis of the codification approach. In last section the reasons to check this codification method and not another was already shown. Now the question could be, if the application of a numerical algorithm in a converted mixed data set is a good approach to the mixed data treatment problem in clustering.

In order to do it, it has been used the polar codification (basic proposed approach) and the modified k-means algorithm (k-means with introduction of weights). This is a really simple algorithm, given that it is just a simple modification of the well known one. With this, it will be shown how the obtained results are very good and therefore, proved that codification of the categorical variables and the use of a numerical algorithm, even very simple, leads to good results.

In order to compare the results, it has been taken as reference results shown in [3]. These results corresponds to the algorithm developed there, which is one of the best found so far. The experiments in [3] were developed with the particularity that the unknown values of the databases were substituted by the mean value in numerical cases and by the modes in the categorical cases. Here, it has been consider that a much accurate

approach is to omit these unknown values, as considering the mean or the mode for them can not correspond to the reality and then the database is being corrupted. Both types of experiments, however have been done, and the observed differences in the results are less than 1%, so that the final results shown will be those of the analysis with no unknown data as they are thought to be more realistic. There is an extra table in each experiment which shows such comparatives, where the proposed approach results indicated are the weighted-unsupervised results.

Four databases have been taken to do this, because those were the experiments run in [3]. Some of the databases have already been explained here: Vote, Heart-Disease and Credit Approval. The other one is the well-known Iris data set. All them available in the *UCI Repository*.

A bit more information has been added about all data sets. This is about the attributes that form the set of characteristics. It is important because in the weight introduction they will be studied.

Two experiments have been done over each set. The first is with all weights initialized to 1, so the basic k-means algorithm is applied. The second is with the selected weights vector. Results of both cases are shown and compared with [3] and other algorithms.

4.3.1 Iris database

This database contains 150 objects which are divided into three different types of flowers: *Iris Setosa*, *Iris Versicolour*, *Iris Virginica*. This is a fully numerical database with four attributes where the objects are equally divided into the three classes. The attributes are: *sepal length*, *sepal width*, *petal length* and finally the *petal width*.

Given that all the attributes are numerical, the purpose of studying the results of this case is to see how the modified k-means works, because no codification is needed here. So, the only action to be taken before the clustering algorithm is applied is to normalize them, again in the interval [-1,1].

Finally, all the results can be seen in table in Table 4.10 and the posteriori weights that make performance increase are shown in Table 4.11. It can be seen that the attribute which contributes the most to the clustering is the fourth one, which is the *petal width*.

Given that no expert was present but information of the systems was available the process to discover the weight has been to introduce different weights to different inputs and see which one offered the best grouping for the data set. The information

Table 4.10: *Clustering performance for the Iris data set*

Algorithm	**Performance**
K-means	88.7 %
Proposed Algorithm	96 %

Table 4.11: *Weights for the Iris data set*

Parameter	**P1**	**P2**	**P3**	**P4**
Value	1	1	1	2

about the set has not been included for the clustering process, it has only been used to validate the results comparing the ones offered by the algorithm to the real clusters.

The comparison with other algorithms can be seen in Table 4.12.

4.3.2 Vote database

The Vote database has already been described in the previous section. The 16 attributes of this database are: *handicapped-infants*; *water-project-cost-sharing*; *adoption-of-the-budget-resolution*; *physician-fee-freeze*; *el-salvador-aid*; *religious-groups-in-schools*; *anti-satellite-test-ban*; *aid-to-nicaraguan-contras*; *mx-missile*; *immigration*; *synfuels-corporation-cutback*; *education-spending*; *superfund-right-to-sue*; *crime*; *duty-free-exports*; and finally *export-administration-act-south-Africa*.

In the same way as for the Iris set the results and the weights are shown in Tables 4.13 and 4.14 respectively. The most influent attribute is the fourth one, which is the *adoption of the budget resolution*.

The comparison with other algorithms can be seen in Table 4.15.

Table 4.12: *Comparison with other algorithms for the Iris data set*

Approach	**Results** (%)
K-means	88.7%
Ahmand et. al. algorithm	94.7%
Proposed Approach	96%

Table 4.13: *Clustering performance for the Vote data set*

Algorithm	**Performance**
K-means	89.7 %
Proposed Algorithm	97 %

Table 4.14: *Weights for the Vote data set*

Parameter	**P1**	**...**	**P4**	**...**	**P16**
Value	1	...	7	...	1

4.3.3 Heart disease database

This mixed data set has also been described in Section 4.2. The attributes are: *Age, sex, chest pain type, resting blood pres, cholesterol, fasting blood sugar < 120?, resting ecg, max heart rate, exercise induced angina?, oldpeak, slope, number of vessels colored, thal.*

Results and weights can be seen at Tables 4.16 and 4.17 respectively. Most significant attribute is the fifteenth after the codification, which is the *number of vessels colored.*

The comparison with other algorithms can be seen in Table 4.18.

4.3.4 Credit card approval database

Another mixed set described in the previous section. Attributes information: *A1: binary; A2: continuous; A3: continuous; A4: categorical; A5: categorical: A6; categorical; A7: categorical; A8: continuous; A9: binary; A10: binary; A11: continuous; A12: binary; A13: categorical; A14: continuous; A15: continuous.*

Table 4.15: *Comparison with other algorithms for the Vote data set*

Approach	**Results** (%)
K-modes	84%
Hierarchical clustering algorithm	86%
Ahmand et. al. algorithm	86.7%
Proposed Approach	97%

Table 4.16: *Clustering performance for the Heart Disease data set*

Algorithm	**Performance**
K-means	82.4 %
Proposed Algorithm	85.1 %

Table 4.17: *Weights for the Heart Disease data set*

Parameter	**P1**	**...**	**P15**	**...**	**P17**
Value	1	...	3	...	1

Results and weights can be seen in Tables 4.19 and 4.20. It is important to remark that the weighing here has been much harder and, even so, the improvement has hardly been noticeable.

The comparisons with other algorithms can be seen in Table 4.21.

4.3.5 Analysis of the results

Looking at Tables 4.10, 4.13, 4.16 and 4.19 we can see the summary of the results for the four cases the codification has been applied. It is quite straight forward that the levels of accuracy are quite good in all cases, being the percentage of well clustered objects bigger than 85% in all cases.

k-means by itself does not offer bad results, which indicates that the codification by itself offer quite enough information, thing that was already shown in the previous section. Nevertheless, in most of the cases there is a significant improvement of the results when the weighing factors are included in the process.

Looking at the Tables 4.12, 4.15, 4.18 and 4.21 that compare this proposal with other algorithms, we can see that this proposal has better results in most of the cases.

Table 4.18: *Comparison with other algorithms for the Heart-Disease data set*

Approach	**Results** (%)
K-prototypes	66%
ECOWEB	74%
SBAC	75%
Ahmand et. al. algorithm	84.8%
Proposed Approach	85.15%

Table 4.19: *Clustering performance for the Credit Card data set*

Algorithm	**Performance**
K-means	86.4 %
Proposed Algorithm	87.4 %

Table 4.20: *Weights for the Credit Card data set*

Parameter	**P1**	**...**	**P6**	**...**	**P16**	**...**	**P20**
Value	1	...	1.8	...	0.4	...	1

Only in the credit card analysis its performance is a bit worse than the Ahmad et. al algorithm, but not by much (less than 1% difference).

Therefore, it has been shown how the proposed codification approach not only works better that the rest of codifications found in literature but, besides, combined with a very easy algorithm, works equally than some very good algorithms. Moreover, not only this approach works in acceptable term, it is easy to understand and apply, which are characteristics highly desirable in every application.

Table 4.21: *Comparison with other algorithms for the Credit Approval data set*

Approach	**Results** (%)
Modha and Spangler's algorithm	83%
Ahmand et. al. algorithm	88.3%
Proposed Approach	87.4%

5. Possibilistic and Normalized C-Regression Models (PNCRM)

5.1 Motivation

In Chapter 1.4 some algorithms designed for clustering purposes have been presented. Independently where an algorithm is placed in the general classification, it is based on the *minimization of a cost index*. Basically, the characteristics of this cost index is what defines the nature of the algorithm. Although many indexes already exist, when there is an application where the already-defined indices do not meet the requirements a new one is developed.

The new cost index proposed here has been designed in order to find a set of lineal combinations of the inputs of the system, i.e. *linear local models*, such that the weighed combination of these local models gives the global output of the system. The validity of each model is defined by the membership function (MF) of each cluster. The total model will be built by the integration of all the local models weighed by the MF. So far, the definition is like the definition of a general index like FCRM and AFCR described in Section 2.3.2. Nevertheless, the particularities of this new index are mainly two:

- ◇ Given that most of the non-exclusive indexes found in the literature are fuzzy and this involves that all clusters are dependent from one another, it seems a good approach to design an index where the **clusters** can be **independent**. Therefore an attribute can belong in different proportions to them, being this level of belonging to each of them not dependent of how much they belong to the others.

  Here the concept of an object *belonging* to a cluster means that the prototype or output that defines the cluster also describes this object. The value of the membership function defines how much the object belong to each cluster.

- ◇ On the other hand, most of the designed algorithms define the membership of each object in a way the final function has a very irregular shape. This makes them not

very easy to understand or interpret. In order to avoid this, a particular shape for them will be imposed. With this introduction, the models will lose generality but will win **interpretability**.

These two characteristics are based on the idea that this cost function will be used to cluster data sets where the membership value of each object to a particular cluster is not a function of the membership values to the others. What is more, one object can have a high membership value (MV) to more than one cluster while another belongs to none. This might happen when the clusters and their local model describe the output attending to different criteria and an object can meet more than one criteria with different or the same level of importance. This is what is called *possibilistic* clustering.

The computed output will be a combination of the outputs of the local models. This is the same as in the fuzzy clustering. However, now the addition of weights of each cluster (membership) will not have the restriction to add 1 necessarily.

The construction of the output by the integration of several linear local models does not mean that each local model is valid by itself, because it can be the total integration what is a good model while none of them are good local models. Imposing a particular shape for the membership bounds the region where the model is valid to just one particular region of the input space. Then, it is achieved that each local model refers to that region and is valid only there, having a meaning by itself.

Thus, this new cost index aims at offering a way to model those data sets where the clusters are independent and also offer a model with low error but high interpretability.

5.2 Characteristics of the new cost function

The desired characteristics already mentioned will be described in this section in more detail.

5.2.1 Possibilistic clustering

In Section 2.3.2 the differences between fuzzy and possibilistic clustering were described. Obviously, the differences in the way they are defined make both techniques capable of working with different purposes. Dropping down the restriction of the normalization of the total membership value of each object means that a degree of freedom is introduced.

This freedom is related to the assignment of the object to the cluster. Now, it might happen that an object belongs to more than one cluster with high membership

while other, does not belong highly to any. From the fuzzy point of view this does not make sense, because all clusters are related and therefore the only way of describing an object is with the combination with several weights of all the prototypes of the identified clusters.

But if now each cluster, instead of representing a set of characteristics of the group where these characteristics are related, represents a set of independent values, where some can be related and others not, the possibilistic version of the clustering process makes more sense.

Let's consider a case where the height of the objects is taken. Three clusters are considered: short, normal and tall. Obviously every person will be in one of those groups or in between two of them. In that case, fuzzy division makes perfect sense.

Now, let's consider this other case: it is collected for each object, how well it plays the piano, the guitar and the saxophone. In this case, one group does not exclude the orders nor is a combination of them. So an object can have high membership in the three of them or even low in all.

It is in the cases like this one, that a possibilistic clustering algorithm seems a good approach. Besides, it also indicates when one object is an outlier because it is not defined by any of the prototypes.

5.2.2 Gaussian membership function

All the fuzzy algorithms described in this work have the membership functions computed according to the characteristics of the index. These can be: distance to center (FCM), output error (FCRM), a combination of both (AFCR), and so on. This means that an object can have certain degrees of its membership for each cluster and the object next to it can have quite different MV. Therefore, the membership functions could have very irregular shapes.

Although this high flexibility makes the error of the clustering process small, the interpretability of the models and the validity region of clusters are quite complicated to understand. The meaning of this is that finding a real interpretation of the functions when they are so irregular is very difficult, because the models or the spaces the user can understand are simpler (geometrical models or bounded regions).

In order to add interpretability to the division realized by the algorithm and the shapes of the MFs it has been adopted a predefined form for the MFs which will be introduced in the cost index as a part of it.

The form of the functions could be any, but a good approach might seem to consider them as normal distributions, i.e:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{\left(-\frac{(x-m)^2}{2\sigma^2}\right)} \tag{5.1}$$

which is one of the most general and well know distributions. This equation shows the unidimensional case, Figure 5.1, where x is the new datum to compute the value of the function, m and σ are the mean and the variance of the distribution respectively.

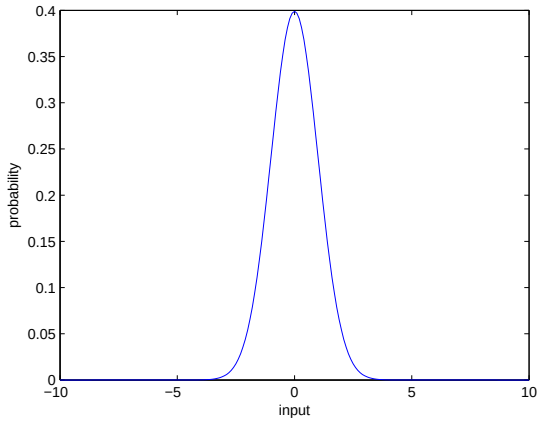

Figure 5.1: *Example of normal distribution with null mean and variance unitary*

The reason to choose this distribution are many:

- ◇ The function has different regions, mainly two, one where it is has high value and another where it has low value. This could represent the levels of membership an object has. Transition between two regions can represent the few elements that have a middle status.
- ◇ It only needs two parameters σ and m to be defined.
- ◇ Its shape is fixed but flexible: changing σ and m different regions can be covered.
- ◇ The transitive zone is smooth, which insinuates that elements close to the high membership will belong to that cluster very similarly to the next objects. This makes sense in real applications where a *close* input has a *close* output.
- ◇ It is very easy to extend to the multidimensional case:

$$f_m(x_1, ..., x_D) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} e^{\left(-\frac{1}{2}(x-m)^T \Sigma^{-1}(x-m)\right)} \quad (5.2)$$

where D is the number of dimensions, Σ is the matrix of variances, where each element of the diagonal (Σ_{ii}) corresponds to σ_i^2 and the rest are zeros, $(x - m)$ is a vector of length D, and $|\Sigma|^{1/2}$ is the root squared of the determinant of the matrix of variances.

- ◇ It is continuous but can be defined only for certain points, so it can take a certain value for each object of the database.

Once the membership functions are like this gaussian distribution they will lose some detail and accuracy, but they will describe compact regions where a models is valid, regions where the model does not describe the objects and in between the smooth transition in level of description.

Gaussian distribution, both in its unidimensional and multidimensional versions, has a multiplying factor for the exponential term. This factor depends on the variance and makes that the total area covered by the function is equal to the unity. This is in that way because this distribution is usually used in statistical analysis and in those cases it is important that the total probability is equal to one. So, the wider the function (larger σ) is the shorter the peak will be too. Figure 5.2 shows how the peak changes when σ of Figure 5.1 is modified.

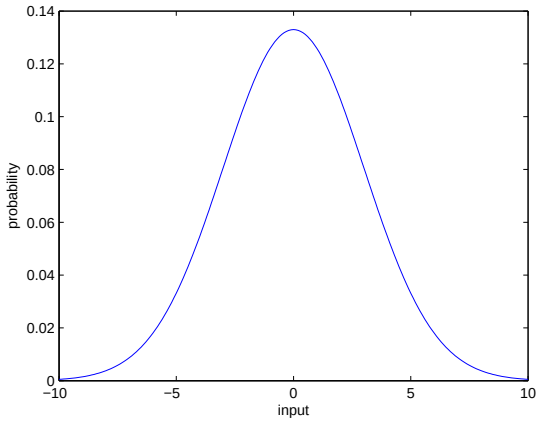

Figure 5.2: *Normal distribution with null mean and* $\sigma = 3$

In the considered case, however, given that this type of functions will represent the membership of a cluster, it will be needed that their value covers from zero (object does not belong to that cluster) to one (objects meets perfectly the characteristics described

by that group). So that, the normal distribution will be all normalized, which means the multiplying factor is removed from the original equations. The final equations for the unidimensional and the multidimensional cases are then:

$$f(x) = \mathrm{e}^{\left(-\frac{(x-m)^2}{2\sigma^2}\right)} \tag{5.3}$$

$$f_m(x_1, ..., x_D) = \mathrm{e}^{\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right)} \tag{5.4}$$

Figure 5.3 shows the normalized version of the MF in Figure 5.1.

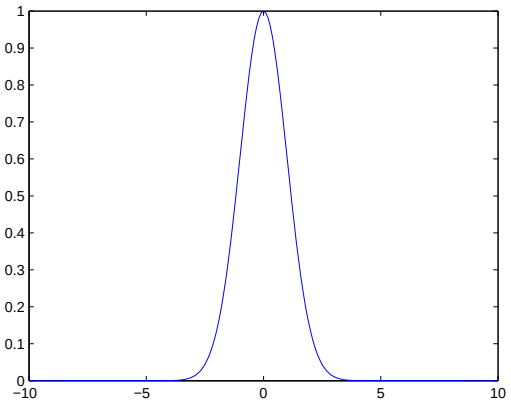

Figure 5.3: *Normalized gaussian distribution will* $m = 0$ *and* $\sigma = 1$

TRANSITION ZONE

In Figures 5.1 and 5.2 it can be seen how, depending on the variance, the transition zone is sharper or softer. This can affect the membership function very much. A sharper transition means that the cluster covers only a region and immediately, its effects disappear. The softer change can means the objects next to the high values of the function also are described by that particular cluster prototype.

But it has already been pointed that the change does depend on how big σ is for that gaussian function and not on the characteristics of the cluster. Therefore, it would be a good idea to introduce a term in the function to modify this characteristic.

The concept of **kurtosis** [12] is related to all this. There exist a positive and a negative kurtosis. *Positive* kurtosis means that the peak of the distribution is narrow

and the transition is abrupt while *negative* kurtosis means that the peak covers a wide area and the transition is smooth.

To modify the transition zone of the distribution, only a parameter (H) in the exponent of the exponential function must be introduced:

$$f(x) = \mathrm{e}^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)^H} \tag{5.5}$$

If $H > 1$ kurtosis will be positive. Sometimes, the resulting function is called *Super-Gaussian*. While when $1 > H > 0$ kurtosis will be negative or *Sub-Gaussian*.

So, with the introduction of the parameter H in the normalized gaussian distribution the transition of the membership of the clusters can be modified. Then, if the clusters are more independent a positive kurtosis will make their membership functions be valid in better defined region.

5.3 Cost index definition

Taking into account all the considerations about the form of the MFs, the type of local models and how they will be integrated mentioned above the cost index can be built.

The output model, as has already been said, will be a weighed combination of the linear local models. Many model structures could be adopted for each local model, but linear models have been chosen due to their interpretability, even when dimension of the input space is high. To define each linear local model the vector of regression parameters ($\vec{\beta}$) that best define the output from the inputs will have to be found. Equation (5.6) shows the form of only one of this models. This shows the estimated output of the object k for cluster i . There it can be seen how the parameters vector is length $D + 1$, where D is the number of inputs. This is because there is an independent term which will be added to all the linear combination of inputs.

$$\hat{y}_{loc\_ik} = \left(\vec{\beta}_{1D}\right)_i \cdot \vec{x}_k + (\beta_0)_i \tag{5.6}$$

where β_{1D} means from the parameter 1 to the last parameter of the vector, while β_0 means the independent term or parameter 0 of the vector..

Last equation shows how is each one of the local models chosen for this case. For each cluster, parameters ($\vec{\beta}$) will have to be found to describe the output of that cluster properly. So, for the data set, a set of c linear models will be defined, being c the number

of clusters. Also, for every object, a level of membership to each model will be computed. Therefore, the global output for each object will be the addition of all estimations of local models multiplied each by the value of the membership function for that object to all clusters:

$$\hat{y}_{glob\_k} = \sum_{i=1}^{c} \mu_{ik} \cdot \left(\left(\vec{\beta}_{1D} \right)_i \cdot \vec{x}_k + (\beta_0)_i \right) \tag{5.7}$$

Again, this is the estimated output of object k.

The goal of most cost indexes in regression applications is to minimize the global error of the model (or models). However, in this case, this goal is slightly different. Obviously, the global error of the total models is desired to be very small, but now, the error of every local model is desired to be small as well. The reason for this has been said before. This index tries to discover local model in a data set, not only to describe the output with a combination of them, but also to describe a local area as much as possible only with that local model, valid regionally.

Thus, in the index, it will be included a term to consider the difference of the real output and each local model, i.e. **local error** and another term to include the difference of the real output and the total output of each object, i.e. **global error**. As the index will be possibilistic, there is no any extra term considering any restriction of the total value of the membership functions. Given that the cost function is composed by errors, it will have to be minimized.

It is very important, in this index, to consider the closeness of objets. In many indexes [21], [22] this closeness is considered including a term in the index to penalize long distances of objects and so boost the form of compact clusters. In this case, although close objects are desired, there is no need to add any extra term. The form of the membership functions consider by itself the distance given that objects close to the center will have high membership ($e^0 = 1$) while objects that are very separated from the center of the cluster will have almost null membership.

Finally, a factor for each term will be considerer: θ for global error and ϕ for local error. These two factors will have to be tuned depending on the application to give higher or lower importance to the term they multiply. Given that the importance of these factors lays on giving more relevance to one of the terms, both factors could be substituted by the relation between both $\gamma = \phi/\theta$, which is the same as normalizing $\theta = 1$.

In equation (5.8) the index including the two errors and the tuning factors can be seen. Function f_i represents the function of the linear model for cluster i, always for an object k and the set of regression coefficients of the cluster β_i.

$$J_{PCRM}\left(X;\sigma,p,\beta\right)=\theta\cdot\sum_{k=1}^{n}\left(y_k-\sum_{i=1}^{c}\mu_{ik}\cdot f_i\left(x_k;\beta_i\right)\right)^2+\phi\cdot\sum_{k=1}^{n}\sum_{i=1}^{c}\mu_{ik}\cdot\left(y_k-f_i\left(x_k;\beta_i\right)\right)^2 \tag{5.8}$$

When the normal distributions are introduced in the previous equation, the final index is obtained:

$$\begin{aligned}J_{PCRM}\left(X;\sigma,p,\beta\right)=\ & \theta\cdot\sum_{k=1}^{n}\left(y_k-\sum_{i=1}^{c}\mathrm{e}^{-\frac{1}{2}\cdot\left(\frac{x_k-p_i}{\sigma_i}\right)^{2H_i}}\cdot f_i\left(x_k;\beta_i\right)\right)^2+\\ &+\phi\cdot\sum_{k=1}^{n}\sum_{i=1}^{c}\mathrm{e}^{-\frac{1}{2}\cdot\left(\frac{x_k-p_i}{\sigma_i}\right)^{2H_i}}\cdot\left(y_k-f_i\left(x_k;\beta_i\right)\right)^2\end{aligned} \tag{5.9}$$

Then the index to be minimized is defined. The inputs variables are the x_k objects ($1\leq k\leq n$) and output vector $\vec{y}$ is formed by all y_k. The parameters to be found are: the center and the variance of each membership function and the parameters of the local model. All defined for each cluster.

Depending on the dimension of the input vector R^N, each center, variance and regression vector will have different length. The length of the two first are N while the length of the third is $N+1$ (always an extra term for the independent term). Therefore, given an application with c clusters and N inputs, the total number of variables to be found is $nv=c\cdot N+c\cdot N+c\cdot(N+1)=c\cdot(3N+1)$.

5.4 Global optimization

Once the index has been defined, it has to be considered the way it is going to be minimized. In many applications [40] [9] the procedure to find the minimum of the cost index is to select randomly an initial point and iteratively find the optimum value for the different unknown variables. Many times, the search is divided into the different unknown variables that have to be found and each one is computed independently keeping the others in the optimum value taken from the previous iteration.

This iterative method is quite fast in most cases, although the solution it offers could happen to be a local minima. This is due to the random initialization of the values. In order to avoid this, the experiment could be run several times and choose the most repeated solution, see Section 4.2, but this method never assures that the solution found is the global solution.

Given that many applications correspond to real cases, it is high recommendable to find the global solution of the problem. For this reason, in this case instead of iteratively finding the solutions, a global solver has been used. Among many global solvers, SSMopt [15] has been chosen. Although the computing time of the application when the global optimization is much higher, the optimal solution could be found. The applications where the proposed index and approach will be used will be *off-line* applications. Thus, the high computational cost of this type of optimization does not affect the performance of the model.

How the solver works is quite complex. Nevertheless, a very simple description of the steps of the solver in order to have a brief idea of its procedures is given here. First, a random generation of points is done and the results of the cost index for that set of points is computed. Second, for the one with the lowest index a local minimization is performed and the smallest value of the index is kept. These steps are repeated until the index does not change for a number of generations of sets of points or the maximum number of iterations is reached. SSMopt works well if enough time is permitted. Usually it needs a long time to converge to the global solution but this can be found.

Therefore, with the index defined as in the previous section and this global optimization method, the best parameters for the **centers**, the **variances** and the **linear coefficients** can be found for a particular data set.

5.5 Examples

5.5.1 Considerations

Before executing any experiment, some considerations have to be taken in order to set the proper framework.

First of all the data will have to be pre-processed. This pre-processing will be similar to the one done in Section 4.2. In this part, missing values will have to be treated and each attribute will have to be normalized. Given that the data sets considered do not have missing values, because they were precisely collected for the experiments, and only normalization is required in the pre-processing.

The second step is to consider all the parameters of the index. These parameters depend on the application and the desired characteristics for the output. In these cases the parameters to set will be:

- ◇ *Transition MF* (H): As mention before this parameter defines how sharp the membership functions are. It can be set individually for each MF (H_i) or globally. In this case it will be consider like a global parameter, constant for all MF. It has been set to $H = 6$ for all cases. The reason for this is because in this way the local models found will be quite independent from each other and valid in a well-defined region.

- ◇ *Error weighing factor* (γ): this parameter defines the relation between ϕ and θ. Setting this parameter is the same as setting $\theta = 1$ and $\phi = \gamma$. In this study, the output of the index is desired to have a set of local models that are valid in a region by themselves. For this reason, this parameter will be set to $\gamma = 5$, i.e $\theta = 1$ and $\phi = 5$, to give the error in the local models 5 times higher importance than the error in the global model, which is considered important too, but not so much.

With these assumptions, the output will only be representative of the data if the set adjust to them. If not, a higher error will be committed, but this is the aim of this index, to find models which follow these assumptions: independence of the clusters and high-representability of local models.

The last consideration is the number of clusters in the output (k). Given that it is an unknown variable, it would be a good approach to run several times the experiments with different values of k and choose the minimum k which gives a small error in the output. Usually, the higher the number of clusters is the lower the error. However, the k which best balances the error and the number of clusters should be taken.

Sometimes, if the user has some information about the signal or a proper visualization of the output can be observed, the number of clusters k can be set a priori to some value. Nevertheless, this is only possible when the data set has just one or two inputs (2D or 3D representations) or an expert in the system advises the user when programming the algorithm.

In the next two subsections, the experiments done using this index will be shown and compared with two algorithms: AFCRC [13] and TSP [14]. The reasons to compare the new algorithm with these two algorithms is that, in first place, both are designed to find a set of clusters and linear local models to represent the global output by combination of the local output models, i.e. the same target as the proposed algorithm. TSP is a possibilistic algorithm, while, AFCRC is a fuzzy algorithm. A comparison with one algorithm of each type will offer better conclusions about the performances of the new algorithm.

A total of four experiments have been performed. In first place a couple of data sets with only **one**-input - **one**-output will be considered. Later, a pair of multidimensional databases with only one output will be considered.

5.5.2 Unidimensional Systems

The results of two unidimensional experiments will be shown and the results achieved with this algorithm will be compared with those achieved with AFCRC in both cases. The first data set is artificial while the second is real.

PARABOLA

This signal has been obtained applying the parabola equation to a linear input and adding to some noise to the output in order to distort slightly the perfect response. In Figure 5.4 the signal has been represented.

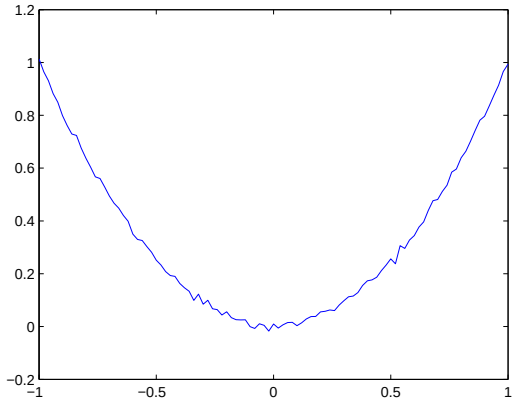

Figure 5.4: *Parabola with noise*

In this case the number of clusters taken is $k = 3$, because it seems a good number of local models: one for the base and one for each of the branches. Applying the index and the global optimization the results can be seen in Figures 5.5 to 5.7, where the global output of the model, each local output and the membership functions are respectively plotted.

The output is quite well estimated setting $k = 3$, using this index and global optimization. When using AFCRC or TSP algorithms, results are very similar. Mean squared error (MSE) for all cases is indicated in Table 5.1. It can be observed that in all cases the error is quite small. This is because the lineal models adjust quite well the output in the three cases and the membership functions are well distributed.

MSE can be a good indicator of the global model. Nevertheless, the aim of this work is to get a set of local models which represents the output locally. So, there

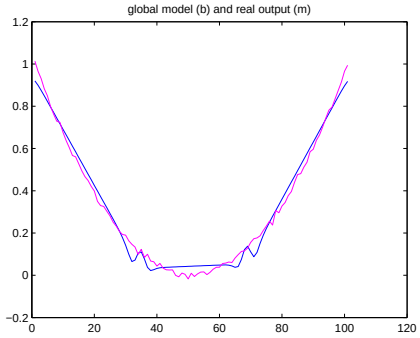

Figure 5.5: Y_{real} *and* Y_{global} *with PNCRM for the parabola*

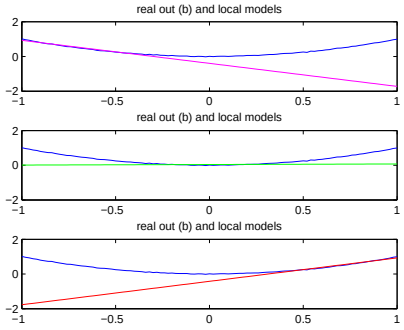

Figure 5.6: Y_{real} *and local models with PNCRM for the parabola*

should also be included one indicator of how good these models adjust the output just in the region where they are valid. This region can be defined as the region where the membership function has a value close to the unity, which is total validity. In this case, it will be taken the threshold $\lambda \geq 0.8$.

Once the threshold has been set, it can be computed the MSE for each one of the clusters, always considering the objects that go beyond this threshold. Table 5.2 shows MSE for local model and also the Percentage of objects represented by each model, for PNCRM, AFCRC and TSP.

Table 5.1: *MSE for parabola*

Algorithm	**PNCRM**	**AFCRC**	**TSP**
MSE	0.0013	0.0018	0.0026

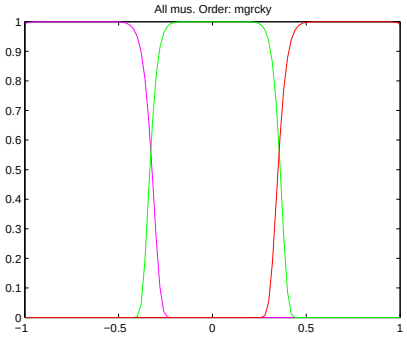

Figure 5.7: *Membership functions with PNCRM for the parabola*

Table 5.2: *Local MSE for parabola*

Algorithm	Charact.	Cluster1	Cluster2	Cluster3
PNCRM	**LMSE**	0.0011	0.0012	0.0011
	%	30.7	32.7	31.7
AFCRC	**LMSE**	0.0004	0.0014	0.0013
	%	21.8	33.7	32.7
TSP	**LMSE**	0.0015	0.0014	0.0015
	%	34.7	34.7	33.7

Local error is again very small for all algorithms. Comparing PNCRM with AFCRC, now, local error is smaller for AFCRC in one of the clusters while the proposed algorithm works better in the other two. Nevertheless, the percentage of objects represented with only one model is smaller too and this might be the reason for the error to be smaller. This means that PNCRM, although having a slightly bigger error, has a small local error and, besides, represents a bigger quantity of objects with only one model. Indeed, with PNCRM, only 4.9% of the objects are not fully represented by a model.

Comparing PNCRM with TSP, it can be observed that the local error for TSP is bigger for clusters. The number of objects well represented by each clusters is slightly bigger too, being all the objects of the set completely represented by a cluster. Indeed, the total is larger than 100% because the MFs share zones with high value in several of them.

To see this better the MFs of AFCRC and TSP have also been represented, see Figures 5.8 and 5.9. If compared with MFs of the proposed algorithm, Figure 5.7, it can

be seen how the shape of the MF for AFCRC and TSP are quite clear and interpretable, although for the PNCRM are much smoother and softer.

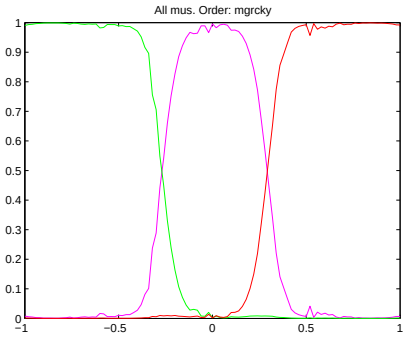

Figure 5.8: *Membership functions with AFCRC for the parabola*

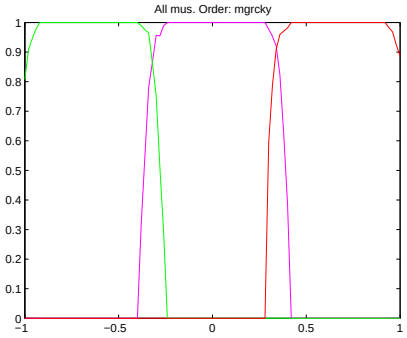

Figure 5.9: *Membership functions with TSP for the parabola*

It is important to indicate that for running AFCRC and TSP some tuning parameters are needed to set. For AFCRC, this is not difficult as in [13] the value where they should be set is indicated. However, for TSP the tuning parameters for this experiment are not indicated anywhere, as happens in every new experiment, and fixing them is quite difficult. Indeed, it is a trial and error work and it takes long time for them to be fixed to a value which proportions a good solution in the output. For this experiment the parameters have been set to: $q = 1$, $p = 1$, $\gamma_f = 0.1$, and $\gamma_q = 1 \cdot 10^{-3}$

GROWTH KINETICS

This signal represents the growth kinetics ratio of a bioreactor (unities: $hours^{-1}$) which is function of the substrate concentration (unities: g/l). Figure 5.10 shows this relationship.

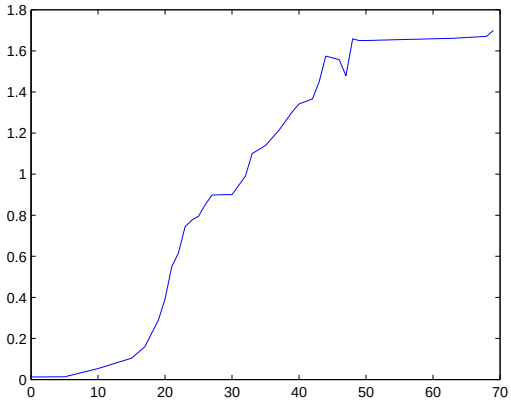

Figure 5.10: *Growth kinetics of a bioreactor*

Visualizing the form of the signal, a proper number for k seems to be $k = 3$, given the more or less 3 different zones of the graph. Once this parameter has been set, experiments can be run. Results when applying PNCRM can be seen in Figures 5.11 to 5.13, where the global output, the local models response and the MF, are, by this order, plotted.

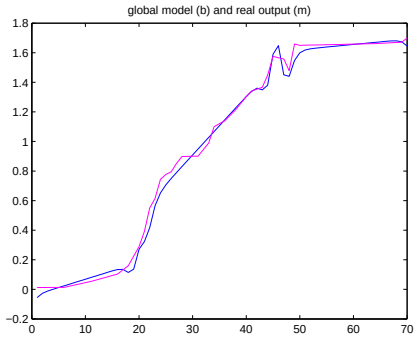

Figure 5.11: Y_{real} *and* Y_{global} *with PNCRM for the growth kinetics of a bioreactor*

In this experiment, in a similar way to the previous one, the output is well fitted setting $k = 3$.

When using AFCRC algorithm, results are very similar, while now TSP differs a bit more than in the previous experiment. MSE for all algorithms is shown in Table 5.3. Again, the error is quite small in the all cases. The error using AFCRC is half the error using PNCRM, while the error using TSP is thrice using PNCRM.

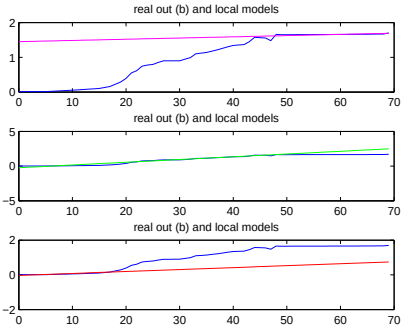

Figure 5.12: Y_{real} *and local models with PNCRM for the growth kinetics of a bioreactor*

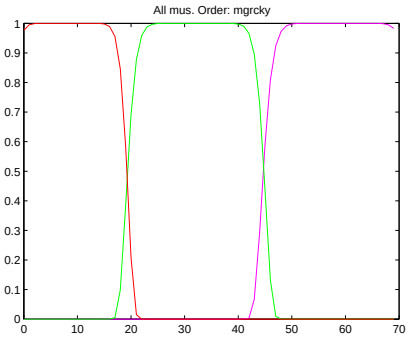

Figure 5.13: *Membership functions with PNCRM for the growth kinetics of a bioreactor*

Table 5.4 shows the local MSE for each cluster for all algorithms when $\lambda \geq 0.8$. Again, as in the previous case, the local error is slightly bigger for the PNCRM algorithm than for the AFCRC. The percentage of points represented only by one model for the proposed index is, again, larger than for AFCRC, leaving only 5.7% of the points with mixed representation. Given that the local error is in both cases small, the larger percentage of objects represented is an advantage of PNCRM over AFCRC in this case.

If the results of TSP are considered, it can be seen how now the error in two local models is larger than for the other two algorithms. As in the previous experiment, the number of objects represented is larger than when using AFCRC or the proposed

Table 5.3: *MSE for growth kinetics of a bioreactor*

Algorithm	**PNCRM**	**AFCRC**	**TSP**
MSE	0.0018	$9{\cdot}10^{-4}$	0.0057

Table 5.4: *Local MSE for growth kinetics of a bioreactor*

Algorithm	Charact.	Cluster1	Cluster2	Cluster3
PNCRM	**LMSE**	0.0011	0.0015	0.0007
	%	34.3	32.9	27.1
AFCRC	**LMSE**	0.0010	0.0012	0.0008
	%	28.6	32.9	27.1
TSP	**LMSE**	0.0011	0.0026	0.0064
	%	32.9	35.7	31.4

algorithm, reaching the totality of the set. Looking at the three algorithms it can be said that PNCRM offers the lower error with the highest number of points locally modelled.

On the other hand, the membership functions of both algorithms are not as similar as in the previous case. In plots of the three cases, Figures 5.13, 5.14 and 5.15, it can be seen how resulting MFs for PNCRM are smooth and locally independent, while MF for AFCRC are more irregular. For TSP there exist an overlapping between two of the MFs in the high zone. This means that, in opposition to what was indicated before, not all the objects are well represented by one cluster, but some objects fit well in two clusters (objects 22-24) while others are not highly described by any local model (objects 43-46).

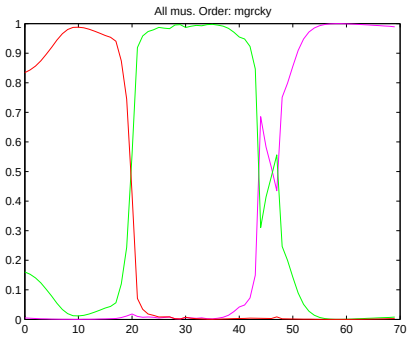

Figure 5.14: *Membership functions with AFCRC for the growth kinetics of a bioreactor*

Therefore, this case, although for PNCRM the error modeling the output is slightly bigger than the one using AFCRC, this model will be more interpretable than the other and, moreover, the local models will have a validity by themselves, which is the goal followed by PNCRM. Although TSP results are quite interpretable too, the error is larger, making this an slightly worst option than the propose algorithm for this case.

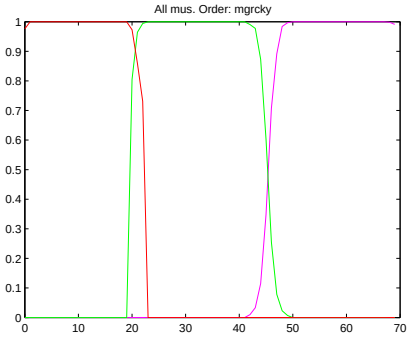

Figure 5.15: *Membership functions with TSP for the growth kinetics of a bioreactor*

As in the previous case, it is important to point out the great difficulty to tune the parameters of the TSP algorithm while parameters for AFCRC were again indicated in [13].

5.5.3 Multidimensional Systems

For this part the index has been used find lineal local models in two "real" experiments. Measurements for both were taken in the *Department of Systems Engineering and Control*[2] at Universitat Politècnica de València[3]. The first is a biomass sensor and the second a thermic system. A better description of both systems is given below.

BIOMASS SENSOR

In order to estimate the biomass concentration in fermenters in the *DISA* a sensor has been developed based on measurements of the absorbance of a sample of the media. This device allows the variation of the gain of the systems, with the regulation of the intensity of the LED diode of reference. The voltage at the output is function of the concentration of biomass in the system.

The relation between the output voltage of the sensor (H_r) and the biomass concentration (c_H) is not lineal, as can be observed in Figure 5.16. Therefore, an approximation by several local models can be a good approach to model this relationship, now dependent on two input variables.

[2]In order to abbreviate it will be used its acronym in Spanish from now on: DISA
[3]Abbreviated: UPV

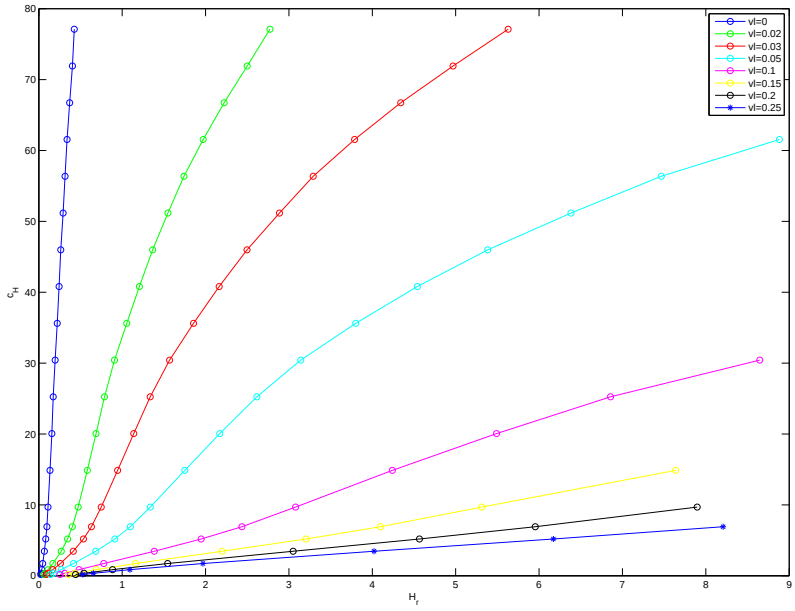

Figure 5.16: *Relation between c_H and H_r for different values of v_l*

Table 5.5: *MSE for biomass sensor, k=4*

Algorithm	**PNCRM**	**AFCRC**	**TSP**
MSE	67.58	11.97	311

Given the highly-nonlinear behaviour of the output, the number of clusters k is difficult to set for this case. For a first approximation $k = 4$ will be set. The comparison of the real output and the output computed by the algorithm can be seen in Figure 5.17. Table 5.5 compares the mean squared error using PNCRM, AFCRC and TSP for the same number of clusters.

It can be observed how, now that the systems is more complex, the local models lose a bit of meaning by themselves. Thus, the error using PNCRM is 5.6 times larger than the error applying AFCRC, but the membership functions will follow a hypergaussian distribution and will be locally interpretable.

Error when using TSP is much larger, but this is due, mainly, to bad setting of the tuning parameters, which have not been able to be set properly for this case, due to its difficulty. For this reason, results of applying TSP algorithm will not be presented in further analysis, because they are not representative of the performance of this algorithm.

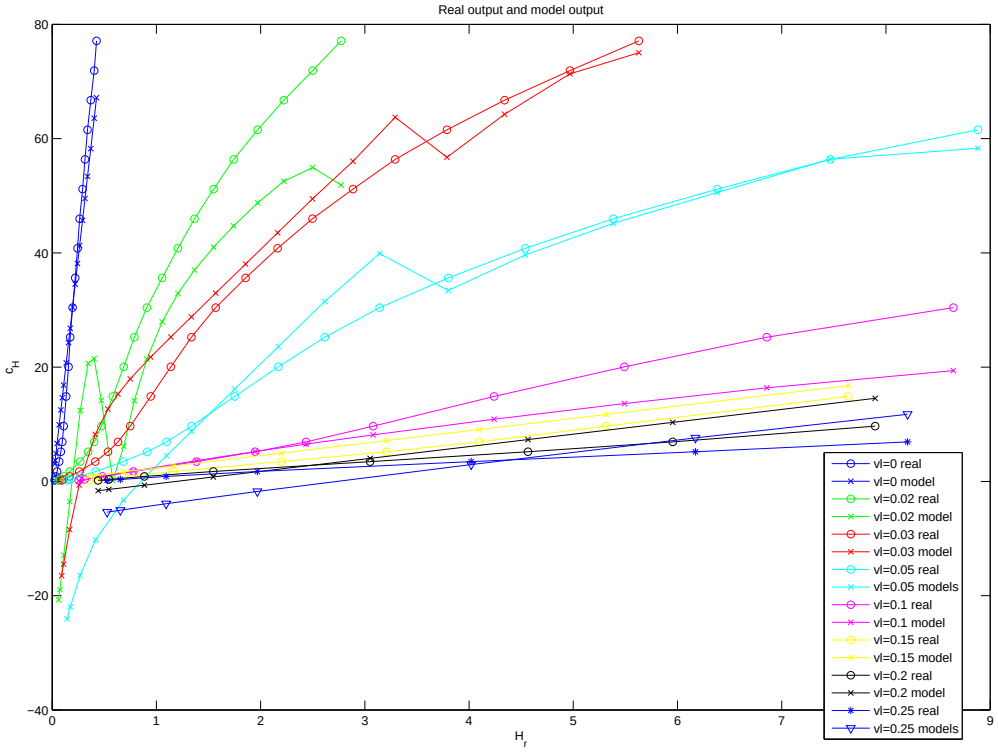

Figure 5.17: *Real and model outputs when k=4 for the biomass sensor*

Thus, it can be concluded that TSP performance can not be extensively tested due to the great difficulty of the tuning of its parameters. Only AFCRC will be used for these comparative studies.

Local errors and number of objects represented (Table 5.6) indicate that although the error is larger, more objects (89.7% versus 61.1%), are fully represented with just one local model. Moreover, it can be seen how mainly two of the local models found with PNCRM are cause of the large MSE, because two of the models have a much lower error than the global output.

If a lower error is desired, then the number of clusters should be increased. Different experiments have been done setting k to different values. Table 5.7 shows the MSE each case. It can be seen how with $k = 10$ the error or PNCRM is comparable to the original AFCRC experiment with $k = 4$. Now the number of clusters is larger, but they are valid locally and highly interpretable.

Table 5.6: *Local MSE for biomass sensor*

Algorithm	Charact.	Clust1	Clust2	Clust3	Clust4
PNCRM	**LMSE**	10.8	1.7	53.7	64.1
	%	30.8	6.8	29.9	22.2
AFCRC	**LMSE**	9.1	2.6	1.7	14.5
	%	12	18.8	20	10.3

Table 5.7: *MSE for biomass sensor, PNCRM with increased k*

k	4	7	8	10
MSE	67.6	20.2	17.1	12.6

Table 5.7 also shows how the error from $k = 7$ is reduced just a bit with every cluster added. Thus, this might suggest that $k = 7$ is the best trade off between the number of clusters and the committed error. Figure 5.18 shows the real output and the estimated output for this value of k. Although some points have a bigger deviation, the modeled output is quite close to the real output. Due to the high nonlinearity of the system, the local models are only an approximation of the real system.

These results show how with this new index a small error can be achieved too, and at the same time, smooth MFs and locally interpretable models. However, in order to get all these together the number of clusters have to be increased until a cluster is assigned to each locally independent region.

The shape of the MFs function correspond to hypergaussians for PNCRM, however as now the input space is 2D the plotting of the functions will be more difficult to visualize. Figure 5.19 the shape of all MFs can be seen.

In the region near the origin, many points are concentrated. So that, many clusters are just valid there, leaving a few for the rest of the space. Thus, the regions with high number of objects will be divided into several clusters in order to find locally valid models.

The MFs for the AFCRC algorithm with $k = 4$ can be seen in Figure 5.20. It can clearly be observed how the MFs for PNCRM are much smoother and better locally defined than those of the AFCRC, even though the number of clusters is smaller for this second one.

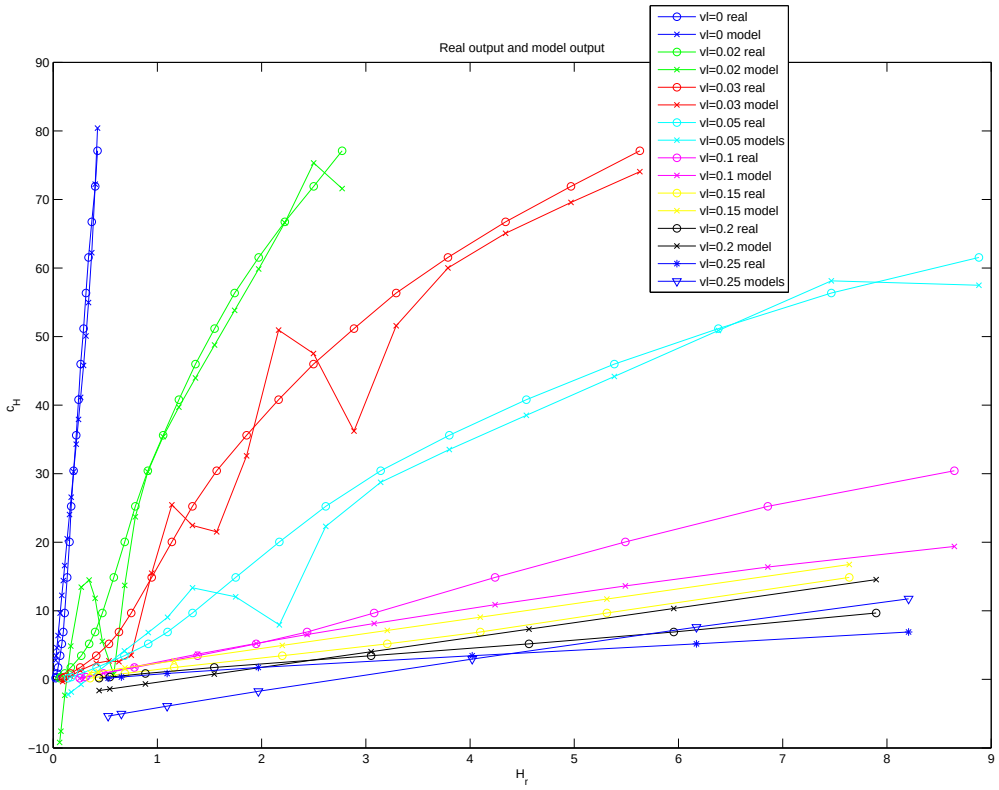

Figure 5.18: *Real and model outputs when k=7 for the biomass sensor*

THERMIC SYSTEM

This system can be also found in one lab in DISA. The system consist in a scale model of an oven. Basically, the system is inside a plastic box to keep the temperature. It includes a resistor, joined to an internal radiator, through which a current proportional to the source voltage flows. This resistor will be the element which introduces energy in the system through its heating. There is also a fan, which introduces air in a continuous way in the box, making the temperature homogenous.

The actuator of the resistor is a voltage source. This source can vary from 0 to 7.5V being this a variation within the range 0-100%. In order to measure the temperature in the resistor, there is a sensor (thermocouple, type K) installed in it. This thermocouple gives after a conditioning circuit, an voltage at the output between 0 and 10V, which will cover the range of temperatures of the sensor.

The process has a non-linear behaviour with regard to the input and, besides, depends on the perturbation in the internal temperature of the oven. According the

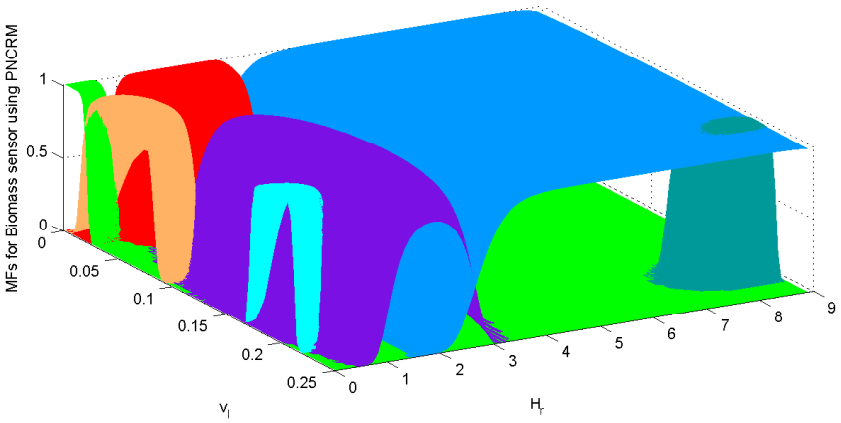

Figure 5.19: *MFs for PNCRM when k=7*

thermophysics laws and omitting the losses in the resistor, the system can be described using this equation:

$$\dot{y} = k_1 \cdot u^2 - k_2 \cdot (y - x_i) \tag{5.10}$$

where: y is the temperature of the resistor (°C), u is the voltage of the actuator's input (V), x_i is the temperature of the periphery of the resistor (°C), and k_1 and k_2 are experimental parameters of the model.

The values for k_1 and k_2 were computed in [47] and are, respectively: $k_1 = 0.0000792^oC/V^2$ and $k_2 = 0.00536$. The temperature of the periphery will be considered as the room temperature ($y_{room} = 25^oC = x_i$), because the constant flow of air from the fan makes this possible. The variations in the temperature will be considered as a perturbation, so they are not included in the characteristic equation.

The discretization of eq. (5.10), with a general sampling time T, results in:

$$y_{k+1} = y_k + T\left(k_1 \cdot u_k^2 - k_2 \cdot (y_k - y_{room})\right) \tag{5.11}$$

Given that the dynamic change of the considered system is not very fast, setting $T = 10$ is enough to get a good discretized signal.

For this process, three signals have been generated, in order to cover different possible situations and study what happens in each case.

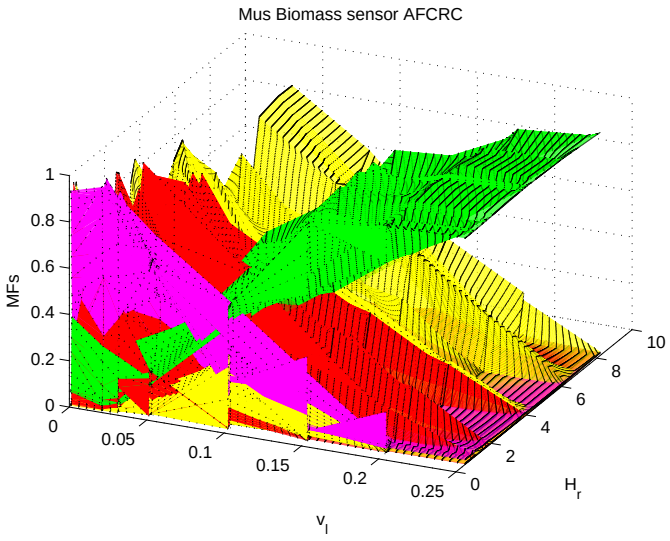

Figure 5.20: *MFs for AFCRC when k=4*

Simulated Data

For this experiment, the input of the process is a set of steps with variable amplitude, always within the input range and with enough length of time to let the output end the transitory response. This type of experiments is very simple, but they have a low frequency component which will difficult the identification of the system.

The input and output of the system can be observed in Figure 5.21, both plotted versus time. Figure 5.22 is the output-input covered space for this experiment.

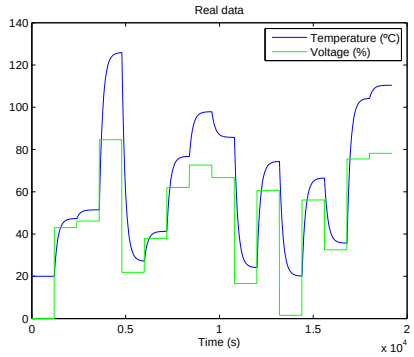

Figure 5.21: *Input and output for simulated data of the oven*

The proper number of clusters is unknown as in the previous cases. For a first attempt, it will be set $k = 2$. Figure 5.23 shows the results of the model for two clusters.

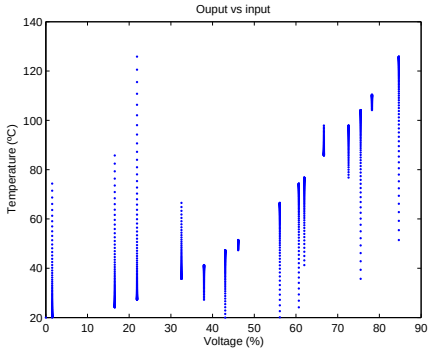

Figure 5.22: *Output vs Input for simulated data of the oven*

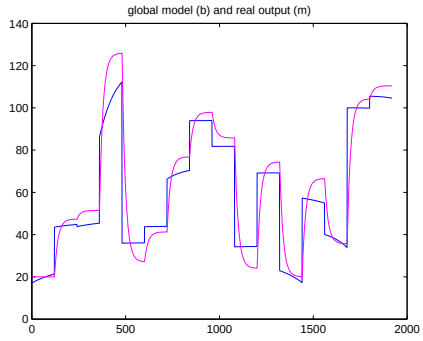

Figure 5.23: *Real and modeled output for simulated data of the oven, k=2*

In Table 5.8 the MSE is shown for both PNCRM and AFCRC algorithms. It is important to remark how in both cases the model found is quite bad (very high error) and very similar to each other.

Looking at the error, it can be concluded that this system is not easy to model just with two clusters. This is due to the low frequency variation of the input signal. Given that the experiment is not rich in information, identifying the output is quite a hard work. Moreover, this process is clearly nonlinear, eq. (5.11), and approaching it by two linear models is just a rough approximation.

Table 5.8: *MSE simulated data of oven, k=2*

Algorithm	**PNCRM**	**AFCRC**
MSE	136	138

Table 5.9: *MSE for simulated data of oven using PNCRM, increased k*

k	2	4	6
MSE	136	109.3	71.1

Table 5.10: *Local MSE for simulated data of oven*

Algorithm	Charact.	Clust1	Clust2	Clust3	Clust4	Clust5	Clust6
PNCRM	**LMSE**	69.4	183.2	37.4	26.5	13.2	33.1
	%	12.5	10.3	23.8	25	11.5	12
AFCRC	**LMSE**	18.5	3.4	7.2	6.2	1.8	31.8
	%	30.5	3.4	4.7	20.3	3.8	0.5

In Table 5.9, it can be seen how the error can be drastically reduced increasing the number of clusters k. So, when the data set does not include much information modeling the signal by local models is very difficult and a larger number of local models will be needed to estimate the output accurately.

For the case when $k = 6$, local errors have been computed (Table 5.10). As in the three previous cases, AFCRC offers smaller local errors, but the number of objects represented with just one model is bigger with PNCRM (95.1% vs 63.2%). The reason for this is that this algorithm tries to model the maximum number of points with just one model and therefore, the more points that are included in a region, the larger the error will be. Therefore, in order to get a smaller error, the number of cluster should be increased. This is not a good example, because the committed error is very large in both cases, as the signal is very difficult to represent with just a few clusters.

Complete Simulated Data

In the last case, the stimulating signal was not rich in information. Figure 5.22 shows how the spectrum output-input is only partially covered and this is the reason why the identification could not be done properly. If the output-input space filled with more information, the identification would be easier.

So, a new set of data has been generated in order to cover this space and have a richer base. Figure 5.24 shows the base now.

If the number of clusters is set to two again, the error, Table 5.11, decreases quite a lot with regards the previous experiment. This is because the data set is now more completed and easier to model just with two local model.

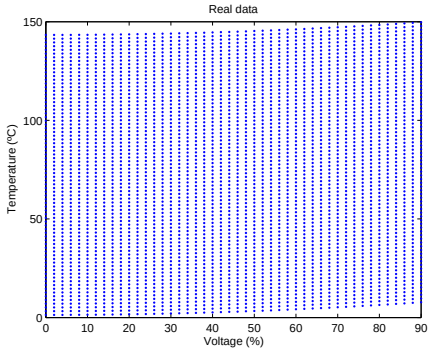

Figure 5.24: *Output vs Input for completed simulated data of the oven*

Table 5.11: *MSE completed simulated data of oven, k=2*

Algorithm	**PNCRM**	**AFCRC**
MSE	9.7	0.24

As in the previous cases the error committed using PNCRM is larger than the error committed using AFCRC. The reasons are the same as explained before. Nevertheless, if now the local error committed by each model is considered, Table 5.12, it can be seen how the proposed algorithm offers a much lower error in both cases an besides the number of objects covered by each model is much larger. Figure 5.25 shows the region covered by each model. The transition zone is the main cause of the large global error in the output, given that each model has low error.

If the error wants to be reduced, an option could be to increase the number of clusters, i.e. $k = 4$. However, in this particular case, if the number of clusters is increased the global error also increases. The reason why this happens is because with more clusters the number of transition zones will increase, and will introduce more error in the global output. This is so because this data set does not follow a distribution by independent clusters. Therefore, if the global output wants to be reduced, in this case one option is

Table 5.12: *Local MSE for complete simulated data of oven*

Algorithm	Charact.	Clust1	Clust2
PNCRM	**LMSE**	0.03	0.03
	%	47.6	47.5
AFCRC	**LMSE**	0.1	0.1
	%	25.6	25.3

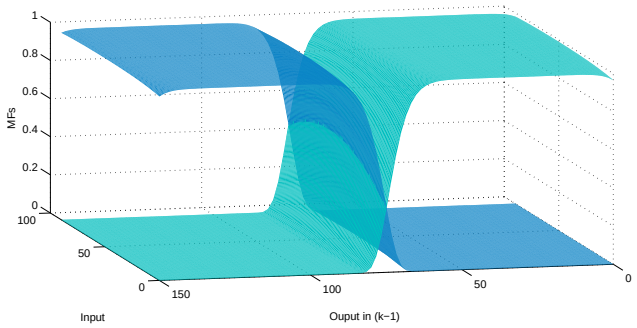

Figure 5.25: *MF using PNCRM for complete simulated data of the oven, k=2*

Table 5.13: *MSE completed simulated data of oven, k=2, different* γ

γ	5	2	1
MSE	9.7	6.8	5.6

to reduce the parameter γ of the index, to give less importance to the local errors. Table 5.13 shows different values of MSE when γ is reduced.

Reducing γ the global error is improved. This is because now the global error has a bigger weight in the index and the optimization of it will lead to a smaller global error. However, the aim of this work is to find local models valid in one region and therefore, the weight of the local error will have to be bigger than the one of the global error. With this reduction of γ it has been shown how a data set which does not follow the proper distribution of independent groups can not be modeled correctly by this index.

Real Data

In order to obtain a data set with real data, an experiment has been performed in the scale model described previously and available in one of the labs in DISA. The experiment has been performed in the same terms as the simulated data experiment. A series of steps was introduced in the input of the system, voltage of the resistor, and always enough time was allowed for the output to reach its stable value for that step. The applied input and the output are represented in Figure 5.26.

Again, this is a quadratic function, (u^2), with addition to some perturbations that are present as in every real process. Therefore, the nonlinearities are even more noticeable in this case.

The number of clusters will be set to $k = 2$ as in the previous experiments of this case. Figure 5.27 shows the real output and the modeled output using PNCRM and

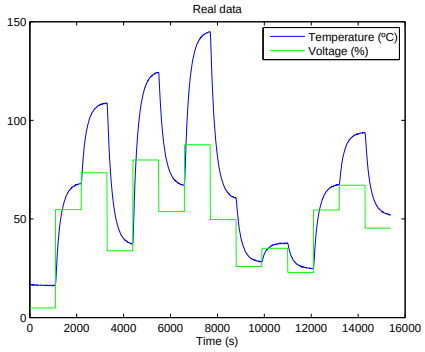

Figure 5.26: *Input and output for real data of the oven*

Table 5.14: *MSE completed real data of oven, k=2*

Algorithm	**PNCRM**	**AFCRC**
MSE	0.126	0.044

$k = 2$. Observing the figure, it can be concluded that the systems is well modeled by just two linear models.

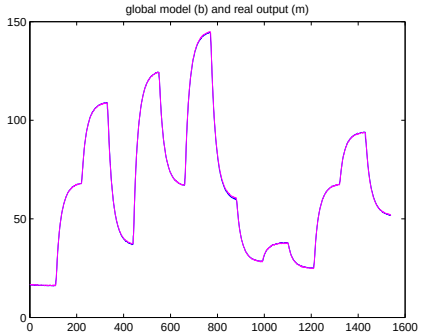

Figure 5.27: *Real and modeled output for real data of the oven, k=2*

Table 5.14 shows the MSE for this experiment when $k = 2$, for both PNCRM and AFCRC. In both cases the error is quite small, being for AFCRC slightly smaller. Nevertheless, as it has been commented many times before, with PNCRM the error might be a bit larger for the same number of clusters, but these clusters are more interpretable and MFs are smoother, and with a determined shape.

Increasing the number of clusters can lead to a smaller error, Table 5.15. Just with two clusters the error is quite small and the models and the weighing (MFs) adjust

Table 5.15: *MSE for real data of sensor using PNCRM, increased k*

k	2	4	8
MSE	0.126	0.074	0.063

Table 5.16: *Local MSE for real data of oven*

Algorithm	Charact.	Clust1	Clust2
PNCRM	**LMSE**	0.04	0.08
	%	35.7	64.3
AFCRC	**LMSE**	0.03	0.04
	%	32.4	53.3

quite well the output, Figure 5.27. For this reason, increasing the number of clusters might be an addition of complexity of the model representing the output and the error is not reduced by much as this is already quite small. So, for this case $k = 2$ seems to be an appropriate number of clusters to represent the process.

With $k = 2$ the local errors have been computed (Table 5.16). Looking at the results same conclusions drawn for the previous cases are applicable here, as the error is larger for PNCRM, although small, and the number of points represented by each model is larger for the proposed algorithm. Indeed, in this case the 100% of the points are covered by this algorithm.

5.6 Analysis of the results

In the previous examples it has been seen how the described index performs in different situations. The introduction of a certain shape for the membership functions assures a smooth and regular shape for this functions but also implies a restriction.

This restriction is the factor that causes that the proposed algorithm has usually a larger error than others similar, in this case AFCRC, for the same number of clusters. This happens because the data sets chosen do not follow the pattern for which this algorithm was designed: several independent clusters. Therefore, if error is desired to be reduced, the number of clusters have to be increased to overcome the already-shaped membership function issue.

The comparison with TSP can only give conclusions only for cases of one-input and one-output. This is so because tuning the parameters of TSP is so difficult that doing it for cases when more inputs are present extends the purpose of this work. The

conclusions obtained for unidimensional cases are that PNCRM works similarly to TSP offering in some cases lower error and softer membership functions.

It has been seen that when the data sets do follow the independent cluster distribution pattern, like in the two first examples (parabola and kinetics of bioreactor), this algorithm does perform very well, reaching low errors, both globally and locally.

Although the introduction of hypergaussian MF implies a bigger error, it also entails to interpretable and smooth MF, which are valid locally and the region they bound is easy to be found.

The global optimization method used to find the solution has some advantages and drawbacks. The main advantage is that with global optimizers the global solution can be found, avoiding undesired local minima. However, the main drawback is the computation time, which is quite large. In this case simulations went from few hours to several days.

With the exponent of the hypergaussian, H, more sharp MF can be obtained. Indeed, this parameter could be introduced, instead of fixed, as a parameter in the index to be optimized. With that the optimal transition could be obtained. Moreover, it could be introduced a parameter H_i for each cluster, making it more flexible.

Finally, the parameter γ could be modified, depending on the characteristics desired in the resulting model. Here it has been set to $\gamma = 5$ to obtain models which clearly represent the total data set locally. Modifying this parameter, the set of models obtained will represent the data set in different ways.

Therefore, this index considers the restriction of certain shape for the MF and gives more importance to the local error of the models in order to find independent clusters. If the data set follows this pattern, then this index will found the best set of models to represent the output of the set.

6. Future Work

6.1 Integration

In the two previous chapters, two techniques: the first to convert categorical data into numerical values keeping similar properties and the second to find independent linear local models in a database.

It has been mentioned many times through this document how mixed databases are increasing in importance. It has also been said that for clustering purposes it is highly recommendable to be able to treat both types of attributes. However, this is not only desirable for finding groups or clusters. When in a data set the output wants to be estimated, using a set of linear models, it is also interesting to include all information available, both the numerical and the categorical attributes.

There are many applications where the output is not the prototype of the cluster, but a local model and including information is an essential part to find both, the validity regions of each local model and the parameters of them. Thus, the future work will focus on integrating both techniques, which can complement each other.

6.2 Application: Continuous Glucose Monitoring

6.2.1 Type 1 Diabetes

Diabetes mellitus is a condition in which there is a chronically raised blood glucose concentration [46]. It is caused by an absolute or relative lack of the hormone insulin. This means that insulin is not being produced by the pancreas, or there is insufficient insulin or insulin action for the body's need.

The two main types of diabetes are: *type 1* (DMI) or insulin-dependent diabetes, and *type 2* (DMII) or non-insulin-dependent diabetes. Type 1 diabetes will be the focus of this description. It is caused by an autoimmune destruction of the insulin-producing β cells of the islets of Langerhans in the pancreas, resulting in absolute insulin deficiency.

Diabetes also affects the rest of the body and can cause serious complications. The most common damages are in the eye, kidney and nerves. Other problems are associated to diabetes too, but are not so frequent as the aforementioned. Diabetes is a major disease which is increasing in number of patients and is a life-disease, here its importance.

The normal regulation of glucose in blood is done by the pancreas, through insulin and glucagon. Insulin is synthesized and secreted from the β cells while glucagon comes from the α cells. γ cells also in the pancreas produce somatostatin. Islets interact with each other through direct contact and through their products, i.e. glucagon stimulates insulin secretion while somatostatin inhibits insulin and glucagon.

Glucose is the main stimulator of insulin release from β cells. Glucose must be metabolized within these cells to stimulate the secretion. Glucose is also carried by special transporters. In normal subjects, blood glucose concentrations are maintained by biological regulators within narrow limits (90-126 mg/dL). Glucose uptake is done by the muscles and tissues. The brain consumes about the 80% of the glucose used by the whole body, but this is not regulated by insulin.

Insulin lowers the glucose levels partly by suppressing glucose output from the liver. Relatively low concentrations of insulin are needed in a stable state, while much higher concentrations are needed after meals.

In type 1 diabetes the most common cause is the T-cell autoimmune destruction of the islet β cells and unless insulin is replaced, severe insulin deficiency results in hyperglycaemia. Therefore, patients which suffer this disease have to inject insulin into their bodies to avoid this. This injection have to be done between 3-4 times a day. In order to inject the right amount of insulin, patients have to measure the glucose blood level before injecting insulin.

To measure the insulin level several devices exist. The most common ones are the **glucometers**. They are devices which estimates the level of glucose from a sample of blood (obtained from pricking the patients finger) placed in a test strip which has the sensor. The sensor is an enzyme and computing the level of oxygen taken by it an estimation of the glucose in blood can be obtained. This is widely used and the patients have to do it by themselves.

Glucose sensors for **continuous monitoring** are becoming very popular. Among them, the needle-type enzyme electrodes that are implanted subcutaneously can be found. These are connected to a monitor which will offer an estimation continuously.

For the injection therapy, **insulin pens** have become a very popular option. The insulin is contained in a pen-shaped barrel, which incorporates a needle. Advantages of this are: convenience and ease of injection, less pain and good patient acceptance. This is an *open-loop* system.

Battery driven syringe **pumps** can form part of a *closed-loop* system. This device is placed around the waist and a cannula is introduced in the abdomen. This device is composed by a sensor to take measurements, a monitor to display them and the devise to do the insulin injection. As there is an injection, it is a closed loop system.

6.2.2 Critical Patients

In critical patients, it is well known that hyperglycemia is very frequent. This is due to the effects of stress and contraregulator hormones and also to the resistance to insulin critical patients have. The effects of hyperglycemia are diverse and many studies have determined that a proper control glucose (< 110 mg/dL) would reduce the morality of ICU patients.

The use of continuous monitoring of glucose, same as in type 1 diabetic patients, could improve the control of the levels of glucose in ICU patients. It has been studied how the continuous and variable injection of insulin in critical patients leads to a more stable control of glucose level.

The measurements of glucose can be done from intravenous sensor or from capillary sensor. The first one provokes a higher risk of infections and the frequency of samples does not allow a real time process, otherwise, they are very accurate. On the other hand, capillary measures can be taken faster, have less risk of infections, and permit a real time process, however, they are not as accurate as the blood measures and are more painful as more measurements are taken.

Therefore, Continuous monitors seem to work well in critical patients, although some points must be considered, because there exist many differences between critical and non-critical patients. For example, the placement of the sensor can influence in the value of the sensor, as the temperature of the region and the flow of blood influence the concentration of glucose. Besides, the treatment of the patient will also influence the variations of the level of glucose.

The reliability of this technology is still under study, but, for sure, a proper control of hyperglycemia in critical patients reduces some lateral problems and the risk of mortality. For this reason, it seems a proper approach to try to use the continuous monitoring and injection in the cases where the treatment allows it and try to facilitate and improve the control of the high levels of glucose.

6.2.3 Patients Databases

In the National project ***Insulaid2****: Advanced Strategies for Closed-loop Glucose Control in Type 1 Diabetes Mellitus and Critically-ill Patients*, the *Group of Control of Complex Systems* from *Universitat Politècnica de València* and the *Modal Intervals and Control Engineering Lab* from *Universitat de Girona* work together. The University Hospital *Dr. Josep Trueta* from Girona is also working in the project.

As part of this project, two sets of data are available: One, contains information about patients with type 1 diabetes in normal conditions. The other contains information about patients who are in the intensive care unity at hospital and suffer hyperglycemia (high level of glucose). Due to the hyperglycemia, the patients undergo an insulin treatment. Their level of glucose is also continually monitored and insulin is administrated.

PATIENTS WITH DMI

This database contains many information about **22** patients. Some variables are statics: age, smoker, civil status, type of insulin used, low and high blood pressure, etc. Others are referred to the state of the patient in the last 6 months: red blood cells, cholesterol, glucagon, etc. Several dynamic measurements were also taken. Although most of the variables were numerical, a big part of them were categorical.

Patients volunteered to perform a study and take measurements during three days. The first day, patients were at hospital from 9 a.m. to 9 p.m. There, a blood glucose (G_b) measurement was taken every 15 min for two hours after a meal and 30min otherwise. All patients had a monitor incorporated and estimated glucose (G_m) samples from this monitor were taken every 5 min. The estimated measurements come from the glucose in the intersticial fluid (G_p) which is were the sensor of the monitor was implanted. Capillary glucose (G_c) measures were taken 3 times a day, before each meal [4].

[4]The three measurements of glucose from the body G_b, G_c, and G_p correspond to different ways of measuring the glucose. The value obtained depends on the part of the body from where the measurement is taken, given the diffusion of the glucose through the body. There exist a relation between the different measurement, as in [33], but there is no a accurate equation for it yet.

Besides, information about each meal was included, i.e. grams of composition of each meal, divided into: simple carbohydrates, complex carbohydrates, proteins, fat and fibre. Also, information about the quantity of insulin injected and the timing for both meals and injections.

The last two days patients were at home, and blood glucose samples were not taken. However, capillary measures (G_c) were still taken three times a day and monitor estimations were stored again every 5min. The information about the composition of meal, the administrated insulin and their times were also kept those days.

The **monitor** [5], apart from the estimation of the level of glucose in blood, offered a value for the **voltage** (v_m) and **current** (i_m) coming from the glucose sensor. It is with these information that the actual model of the glucose prediction in the monitor works and offers the estimation using linear regression techniques.

ICU PATIENTS

The subjects of study are patients in the ICU at University Hospital *Dr. Josep Trueta* in Girona. The population for the study has been set to 120 patients to have a significative group. The patients were volunteers or their tutor or person in charge allowed the study.

Apart from the patient general information, similar to the variables defined for the previous data set (type 1 diabetic patients at home) and information about meals, the main variables stored for each patient were the arterial glucose[6] and the blood glucose estimated by the monitor from the intersticial fluid[7].

Another important parameter is the localization of the sensor. Three different zones are possible: abdomen, high part of the leg, and around the chest region. As the sensor can be used for three says (72h) if the patients stays longer, the sensor will be replaced by another with same characteristics.

There are medical conditions to include or exclude a patient in the experiment, apart from the authorization of the patients themselves. These conditions (both for inclusion or exclusion) are related in the protocol of the project.

[5]Minimed®CGMS Gold.
[6]measured using Hemocue Glucose 201 RT®.
[7]measured with Paradigm REAL-Time of Medtronic Minimed®.

6.3 Work

The actual model of the glucose monitor for type 1 diabetic patients is not a very good estimator given that the measurements obtained for the level of glucose in blood which were compared with the real measures taken the first day at hospital are very different. Furthermore, precision differs depending on the glucose rang and trend. Having a good prediction of the measured variable, i.e a good model, is the first step to have, in the future, a good controller.

For this reason, the aim of the future work is to apply both proposed methods in order to improve the estimation of the monitor by computing a better model based in the information available.

The information contained in the data set is, nevertheless, a lot of information. There is no security of which variables of all influence in the output yet. Therefore, as a first step it will be necessary to perform a variable selection.

The global optimization has been performed using a particular global solver. The results seem to be good, but it takes a long time to get them. Therefore, it seems a good idea to perform the same optimization with other solvers in order to compare the results and computation time with the actual one and then decide which one will be used.

It has been mentioned that the different measurements of glucose (G_b, G_c, and G_p) are correlated but the relation between them is not well known. So, it will be important to see how these variables are related before doing the prediction of the level of glucose in blood, because the sensor in the monitor has an estimation of G_p and small differences with G_b can contribute to the error in the estimation.

Once this has been done, the proper number of cluster will have to be obtained. There is no information a priori for this, so different values of k will be checked and the one with the best trade off with small error and not very big k will be taken.

So, the final goal of this project is to find a set of k lineal local models that can represent the output of the continuous monitor of glucose for DMI quite accurately in function of the input mixed type information selected.

For critical patients, the goal is very similar. Although the data sets will be different and the influential variable will not be the same, the target here is to obtain the relation between blood glucose and sensor measurements, so, later, a set of k_1 linear local models can be obtained to represent the output. Note that k and k_1 might be different.

Bibliography

[1] J. Abonyi, M. D. Alexiuk, P. Angelov, and B. Bird. *Advances in Fuzzy Clustering and its applications*. John Wiley & Sons, Ltd, 2007. 2.3.2, 2.3.2, 2.3.2, 2.3.2

[2] R. Agrawal, J.E. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications, December 14 1999. US Patent 6,003,029. 2.4.2

[3] A. Ahmad and L. Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data & Knowledge Engineering*, 63(2):503–527, 2007. 3.3.1, 4.3

[4] P. Andritsos. Data clustering techniques. *Toronto, University of Toronto, Dep. of Computer Science*, 2002. 1.1.1, 3.2

[5] P. Andritsos. *Scalable Clustering of Categorical Data and Applications*. PhD thesis, University of Toronto, 2004. 1.2.2, 1.3, 3.2

[6] M. Ankerst, M.M. Breunig, H.P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 49–60, 1999. 2.4.2

[7] D. Barbará, Y. Li, and J. Couto. COOLCAT: an entropy-based algorithm for categorical clustering. *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589, 2002. 3.2

[8] I. J. Benítez-Sánchez. Técnicas de agrupamiento para el análisis de datos cualitativos y cuantitativos. Master's thesis, Technical University of Valencia, 2005. 3.3.2

[9] J.C. Bezdek, R. Ehrlich, and W. Full. Fcm: The fuzzy c-means clustering algorithm. *Computers and Geosciences*, 10, 1984. 5.4

[10] P.S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. *Knowledge Discovery and Data Mining*, pages 9–15, 1998. 2.4.2

[11] K. Chidananda Gowda and E. Diday. Symbolic clustering using a new dissimilarity measure. *Pattern Recognition*, 24(6):567–578, 1991. 1.2.3

[12] L. T. DeCarlo. On the Meaning and Use of Kurtosis. *Psychological Methods*, 2:292–307, 1997. 5.2.2

[13] J. L. Díez. *Técnicas de agrupamiento para identificación y control por modelos locales*. PhD thesis, Universitat Politècnica de València, 2003. 5.5.1, 5.5.2, 5.5.2

[14] J. L. Diez, A. Sala, and J. L. Navarro. Target shape possibilistic clustering applied to local-model identification. *Engineering Applications of Artificial Intelligence 4th*, 2005. 5.5.1

[15] J. A. Egea-Larrosa. *New Heuristics for Global Optimization of Complex bioprocesses*. PhD thesis, Universidade de Vigo, 2008. 5.4

[16] D. Fisher. Cobweb: Knowledge acquisition via conceptual clustering. *Machine Learning*, 2:139–172, 1987. 2.2.1

[17] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS—clustering categorical data using summaries. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 73–83, 1999. 3.2

[18] J.H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40(1-3):11–61, 1989. 2.2.1

[19] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: an approach based on dynamical systems. *The VLDB Journal The International Journal on Very Large Data Bases*, 8(3):222–236, 2000. 3.2

[20] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000. 3.2

[21] JA Hartigan and MA Wong. A K-means clustering algorithm. *JR Stat. Soc., Ser. C*, 28:100–108, 1979. 2.2.2, 3.3.1, 5.3

[22] RJ Hathaway and JC Bezdek. Switching regression models and fuzzy clustering. *IEEE Transactions on fuzzy systems*, 1(3):195–204, 1993. 2.3.2, 5.3

[23] RJ Hathaway, JC Bezdek, and W. Pedrycz. A parametric model for fusing heterogeneous fuzzy data. *Fuzzy Systems, IEEE Transactions on*, 4(3):270–281, 1996. 3.2

[24] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. *Knowledge Discovery and Data Mining*, 5865, 1998. 2.4.2

[25] C.C. Hsu, C.L. Chen, and Y.W. Su. Hierarchical clustering of mixed data based on distance hierarchy. *Information Sciences*, 177(20):4474–4492, 2007. 3.3.2

[26] Z. Huang. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998. 3.2, 3.3.1, 3.3.1

[27] M. Indulska and ME Orlowska. Gravity based spatial clustering. *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 125–130, 2002. 2.4.2

[28] A.K. Jain and R.C. Dubes. *Algorithms for clustering data*. 1988. 1.2, 1.2.1

[29] AK Jain, MN Murty, and PJ Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999. 1.3

[30] P.E. Jouve and N. Nicoloyannis. A New Method for Combining Partitions, Applications for Distributed Clustering. *Parallel and Distributed computing for Machine Learning*, 2003. 3.2

[31] G. Karypis, E.H. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999. 2.2.1

[32] L. Kaufman and P.J. Rousseeuw. Finding groups in data. an introduction to cluster analysis. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, New York: Wiley, 1990*, 1990. 2.2.2

[33] K. Kuwa, T. Nakayama, T. Hoshino, and M. Tominaga. Relationships of glucose concentrations in capillary whole blood, venous whole blood and venous plasma. *Clinica Chimica Acta*, 307:187–192, 2001. 4

[34] Computer machine learning repository. http://archive.ics.uci.edu/ml/, 1998. 4.2

[35] D. Maravall Gómez Allende. *Algoritmos de Agrupacion de Clases: Clustering*. Ed. RA-MA, 1993. 2.2.2

[36] Y. Miin-shen, P. Hwang, and C. De-hua. Fuzzy Clustering Algorithms for Mixed Feature Variables. *Fuzzy Sets and Systems*, 141(2):301–317, 2004. 3.2

[37] R.T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 144–155, 1994. 2.2.2

[38] E.A. Patrick. *Fundamentals of Pattern Recognition*. Prentice Hall, 1972. 2.2.2

[39] D. Gunopulos R. Agrawal, J. Gehrke and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Min. Knowl. Discov*, 11:5–33, 2005. 3.2

[40] M. Ryoke, Y. Nakamori, and K. Suzuki. Adaptive fuzzy clustering and fuzzy prediction modelsryoke1995afc. In *Fuzzy Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE International Conference on*, volume 4, 1995. 2.3.2, 5.4

[41] R. Rastogi S. Guha and K. Shim. Cure: an efficient clustering algorithm for large databases. *Information Systems*, 26:35–58, 2001. 2.2.1

[42] J. Sander, M. Ester, H.P. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998. 2.4.2, 2.4.2

[43] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973. 2.2.1

[44] C.S. Wallace and D.L. Dowe. Intrinsic classification by MML-the Snob program. *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*, 37:44, 1994. 3.2

[45] W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 186–195, 1997. 2.4.2

[46] G. Williams and J. C. Pickup. *Handbook of Diabetes*. Blackwell Publishing, 2004. 6.2.1

[47] M. Matínez X. Blasco, J.M. Herrero and J. Senet. Nonlinear parametric model identification with genetic algorithms. application to a thermal process. *Connectionist models of neurons, learning processes and artificial intelligence. Proceedings of 6th International Conference Artificial and Neural Networks*, 2001. 5.5.3

[48] X. Xu Z. He and S. Deng. Scalable Algorithms for Clustering Large Datasets with Mixed Type Attributes. *International Journal of Intelligent Systems*, 20:1077–1089, 2005. 3.3.1, 3.3.2

[49] M.J. Zaki, M. Peters, I. Assent, and T. Seidl. Clicks: An effective algorithm for mining subspace clusters in categorical datasets. *Data & Knowledge Engineering*, 60(1):51–70, 2007. 3.2

[50] H. Zengyou, X. Xiaofei, and D. Shengchun. Squeezer: an efficient algorithm for clustering categorical data. *Journal of Computer Science and Technology*, 17(5):611–624, 2002. 3.2, 3.3.1

[51] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for large databases. *Proc. SIGmod*, 6:103–114, 1996. 2.2.1