Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

Heterogeneous neural networks and the leader2 algorithm

Jerónimo Hernández González

Advisor: Lluís A. Belanche Muñoz

September 3rd, 2010

Contents

1 Introduction

This paper is the final document written to gather the impressions and conclusions which we have come to during the development of this master thesis.

In this research project you will find the description of a new kind of artificial neural network, *Heterogeneous Neural Network 2* (*HNN2*), which can be seen as a general abstraction of the Radial Basis Function network. The model of neuron used is an improved version of the one presented by Belanche [1] and the neural network is initialized using a clustering algorithm, *Leader2*, developed at [2].
We will explain the way we have followed to get this artificial neural network that works allways with understandable information, uses the concept of similarity and allows users to improve the algorithm results taking advantage of expert information.

The basic *Heterogeneous Neural Network* (*HNN*) is also known as *Similarity Neural Network* (*SNN*), by the importance of the similarity measures inside this method. The basic idea is that a combination of similarity functions, comparing variables independently, is more capable of catching better the singularity of an heterogeneous data set than other methods which require previous data transformation. Each variable has its own characteristics, which is information that can be used by the expert that knows it to choose its most suitable similarity function, taking advantage of all the information he has. If this is done for each variable, we will be working probably with a similarity measure that understands better the data. *Missing values* are also a relevant characteristic of heterogeneous data, so we have to learn to deal with them. All these ideas are applied to *HNN* and *Leader2*, joint to several improvements performed to the neural network, like regularization or *Alternate Optimization*, in order to fit better the data but avoiding overfitting. This is why we have called it *Heterogeneous Neural Network 2* (*HNN2*).

This document is divided in several chapters. Initially, we will give an in-depth description of the problem which we want to solve. In the second chapter, State of the art, you will get a wide perspective of how was the field in which this project has been developed before we started. Then, there is a description of the used methodology, where you can find the main decisions and the development itself, followed by the explanation of the experimental settings done to test the *HNN2*. Their results are commented and evaluated in the next chapter, and next some conclusions are inferred. Finally, you will find the references used in the research and several annexes with additional relevant information.

But in first term, before starting the description of the problem and in the way of making the reading easier, it is necessary to provide you some vocabulary to know exactly the meaning we have given to several key words. Next, in the same terms, you will find the most used symbols with their description.

1.A Vocabulary

Here you will find the definition of some terms which are constantly used in this document and can have different interpretations. For this reason, we want to stablish the meaning we use:

Similarity It is defined as a value between 0 and 1 that measures the resemblance of two values. It is such that the more resemblant are two values, the bigger is the similarity. There are several methods to measure it, which change for different data types. Usually, the similarity of a value with itself is 1. It will be nearer to 0 for values with lower resemblance.

Distance It is defined as a value between 0 and 1 that measures the resemblance of two values. It is such that the more resemblance are two values, the smaller is the distance. The distance is a metric which accomplish the triangular inequality, inside the group of the disimilarities. For this reason, a distance can be expressed as a similarity doing:

$$Similarity = 1 - Distance$$

The distance here is bounded between 0 and 1 in order to make easier the movement to similarity, but it is usually defined in $[0, +\infty)$.

Leader2 Clustering algorithm based on the leader algorithm [3] developed at [2]. It is the modification of an one-pass algorithm, that groups instances choosing some reference instances (leaders) and assigning to each leader all the instances that are, at least, s_{min} similar with an specific leader. It only needs the s_{min} parameter to perform (and the data set, of course).

Cluster Set of instances that have the same reference instance and their similarity with it is higher than s_{min}. For this reason, the s_{min} is a key parameter that controls the number of clusters. If an instance has a similarity higher than s_{min} with two different leaders, it will belong to the most-similar leader cluster because an instance can only belong to one cluster.

Leader It is an instance that is used as reference of a cluster since it is created. It will be used to assess if instances belong to the cluster that the leader represents. For this, the similarity between the instance and the leader will be calculated and only would be accepted if it is higher than s_{min}.

ANN An *artificial neural network* (*ANN*), also called *neural network* (*NN*), is a mathematical model based on the structure of biological neural networks that processes data through an interconnected group of —artificial— neurons. It can adapt to the data and learn from the examples. It usually distributes the neurons in several layers, each one of them has an specific role in the learning process. It can be seen as a black-box where you only know about the first and the last layer, but not about the middle ones.

Neuron A neuron is a mathematical function and the basic unit in an *ANN*. There are plenty of different neurons (mathematical functions) and, inside an *ANN*, we find at least two kinds of neuron. The common characteristics are that they usually receive one or more inputs, executes the function and returns a single output. Also, the input(s) can be weighted, for which some parameters are used. In the mathematical function, which can be also called transfer function, usually there is a parameter that controls its behaviour, getting different responses to the same input by changing this paramater.
Attending to the situation of a neuron into the *ANN* structure, we can classify it in:

- **Hidden neuron** Hidden neurons are all the neurons which are not in the last layer. They are the contents of the black-box: we don't know the values that they interchange, we only know the inputs.
- **Output neuron** They are the neurons of the last layer. We know their outputs, because they are also the outputs of the *ANN* (black-box).

HNN It is a two-layer *ANN*. The first-layer neurons use *S-neuron*, the neuron model described at [4], and the second-layer transfer function (there is only one output neuron) is a linear function that adds the weighted outputs of the first layer.

SNN See *HNN* definition.

s_{min} **(Minimum similarity)** It represents a similarity value, so it has to be between 0 and 1. It is used as necessary condition in *Leader2* algorithm for an instance belonging to a cluster: the similarity between the leader of that cluster and the instance has to be higher than s_{min}.

Regularization In machine learning, regularization is a technique that introduces additional information to cope with a problem preventing overfitting. This information is usually a penalty for complexity.

Alternate Optimization It is a technique that combines the optimization of two parameters in an alternate way. It optimizes the first parameter and uses its new value (after optimizing) to perform the optimization of the second one. Then, it performs the same with the first optimization and so on. It requires initial values for the parameters.

Missing value Missing value is the name used when there is lack of information. It is the case of unknowing the value associated with a variable for one instance.

Convergence A numerical series converges if after n iterations the numerical value stabilize around a non-infinite number.

Data set It gathers all the information that represents the problem. It is made up of a conjunt of instances, usually divided in *training* and *test* sets. The data set has to be a good example of the data generator behaviour.

Instance It is each one of the observations of the data set gathered to be analysed. It can be called *example*, too.

Data type Each variable represents an specific characteristic which determines the set of possible values. The nature of this set is the data type, which is also different for each variable.

Variable It is each one of the characteristics collected to represent examples in the data set. In this way, an instance is a set of collected values of the variables characterizing the problem. Attending to the meaning of the data it contains, we can talk of the *class variable*, which indicates the observed result that returns the real model. It will be a numeric continuous value if the problem is a regression problem or a label of the class which it belongs to if it is a classification problem.
Attending to the data type, there are different kinds of variables:

- **Continuous** A variable is continuous if it can take any real value inside an interval and if it is related with other possible values through an ordering relationship. The number of possible values is infinite.

* **Discrete** A variable is continuous if it can take any natural value inside an interval and if it is related with other possible values through an ordering relationship. The number of possible values is infinite only when at least one of the two interval bounds is $\pm\infty$.
* **Ordinal** A variable is ordinal if it can take any value inside a set of possible values (numerical or not) and if this set can be ordered. The number of possible values is not infinite.
* **Categorical** Categorical variables take their non-numerical values from a known set. There is not an infinite number of possible values and it is not possible to stablish any order or grade relationship between them.
* **Binary** This kind of variable indicates the presence or absence of some characteristic. There are only two possible values (+/-), but only the positive one is a relevant informative value. If two different instances have the negative value (characteristic absence), the comparison can not be performed because it is not an interpretable situation. If this situation could be interpreted, the variable would be a two-valued categorical variable.
* **Fuzzy** This kind of variable represents continuous variables whose values are fuzzy numbers. A fuzzy number is a number with an decreasing interval of possible membership. It is used to express imprecision.

1.B Symbols

Here you will find the most used symbols with their description. This is necessary to follow the text reading:

m Number of variables in the data set (without taking into account the class variable).

n Number of instances in the data set.

λ Regularization constant or parameter. When the method uses a different regularization parameter for each second layer weight (λ_i), that is called local regularization.

h Number of hidden neurons or clusters.

t Number of output neurons.

σ Width parameter of the *RBF* networks. It is possible to use a different width parameter for each first layer neuron (σ_i).

p Smoothness parameter of the *heterogeneous neurons*, S-neurons. It is possible to use a different smoothness parameter for each first layer neuron (p_i).

$\mathscr{X}$ *Missing value.*

X Data set. It is a set of instances that represents an space from which the examples are collected.

x It represents an instance. x_i is the i^{th} instance. An instance is composed by some variables such that $x_i \equiv \{x_{i1}, \ldots, x_{im}\}$.

s **or** *S* It means similarity. If you find s_{ijk}, it means that we are talking about the partial similarity between the i^{th} and the j^{th} instances of the data set, for the k^{th} variable. From here, an equivalent nomenclature can be inferred: $s_{ij} \equiv s(x_i, x_j)$ and then $s_{ijk} \equiv s(x_{ik}, x_{jk})$.

w It means second layer weight. Their are associated with one hidden neuron and one output neuron. With only one *output neuron*, w_j is the weight associated to the j^{th} *hidden neuron* output. With more than one *output neuron*, w_{kj} is the weight of the k^{th} *output neuron* associated to the j^{th} *hidden neuron* output.

2
Definition of the problem and definition of the goals

Machine learning is a field of artificial intelligence where some models are used usually to analyse real situations and, then, try to predict the outcome of a new case with a similar situation.

Machine learning methods need data to be transformed using a codification that they can interpret. This data usually represents real situations, which are described by natural characteristics with their own values. For example, describing a car, you can see the color, the number of doors, if it is automatic, its width/height, etc. All these characteristics (*variables*) are different in the way they express the information and also in the type of information they express. A car can be automatic or not, but a car can be black, red, gray, blue, etc. Also there is information that could be numeric, like the number of doors, that is represented by a number but that really represents an ordered group of possible values because often you only find three or five-doors cars. It is not the same that the height, which is a continuous real number that, in the case of a car, usually is between 1,20 and 2,5 metres. That is known as heterogeneous data and the world is usually described with this kind of data.

Traditional machine learning methods requires, if they have to cope with heterogeneous data, a transformation that converts all this data into numeric values. With this transformation possibly you are losing some important information, because it can not be represented with the new code or because whoever is preprocessing the data doesn't know it. But you also lose the possibility of interpreting easily the data, because it will be now codified in a non-human understandable way. Finally, the dimensionality of the data set grows exponentially with this kind of transformation.

In addition, we have considered that the use of *ANN* matches with this idea of variety and uncertainty. Then, we proposed to define an *ANN* that would be able to process the data in its original representation. We have chosen this method because one of the most important *ANN* features is adaptability, which is an interesting characteristic for a method dealing with heterogeneous data problems. In this way, different parts of the algorithm have to be able to cope with heterogeneous data, from initializer method of the first layer to activation functions in the neuron models.

If the new method is able to deal with original data (accepting only basic transformations) we will get interpretable information, because the data is the one we have collect or, simply, we know about it. Another advantage of using original data is to retain all the information that could be in the data and take advantage of it.

To do that, we have to build an *ANN* where if one reference instance is needed, it has to be one of the original data set, in order to preserve the interpretability. It also has to be able to deal with heterogeneous data, which requires a measure of similarity/disimilarity calculated independently for each variable and then aggregating them, to take advantage of all the original information. This method has to use original heterogeneous data to improve the fitting. So, at least, this method has not to be worse than other known methods because, if not, we will not be demonstrating the advantage of using data without transformations.

The heterogeneuos data has, as its name indicates, many different faces. So, since we are not able to contemplate all these data types in advance, we have to do a customizable method where users define their own data types and partial similarity functions.

3

State of the art

The contents of this master thesis belongs to the artificial intelligence area called *data mining*. This area groups all the techniques that, using the computational power developed the last years, extract information and knowledge of the data bases recognizing concepts or patterns that the human mind can not detect. In this sense, *machine learning* [5] is the discipline that designs and develops algorithmic techniques that allow computers to learn from a data set. These methods extract automatically a common pattern or concept from the data and adapt it in order to infer some conclusions.

3.A Similarity and missing values

Usually, in the learning process, the instances have to be compared and, for this, different similarity/dissimilarity methods have been proposed. Traditionally, the most used metric has been the distance metric, which is called only distance. Distance methods are an specific case of the dissimilarity methods which fulfill the triangular inequality. The use of similarity measures is so justified in the literature [1] [6], specially in some topics where the idea of distance doesn't fit because it can not be defined.
In classical artificial intelligence, the concept of *similarity* appears mainly in case-based reasoning (CBR), but it has had also a long and successful history in the literature of cluster analysis and data clustering algorithms.
Belanche enumerates the properties that every similarity measure fulfills. For the space X, which represents all the possible cases of the problem, he defines:

1. *Non-negativity*. $s_{ij} \geq 0 \ \ \forall x_i, x_j \in X$
2. *Symmetry*. $s_{ij} = s_{ji} \ \ \forall x_i, x_j \in X$
3. *Boundedness*. There is a maximum attained similarity: $\exists s_{max} \in \mathbb{R}^+ : \ s_{ij} \leq s_{max} \ \ \forall x_i, x_j \in X$.
4. *Minimality*. Reflexivity in the strong sense. $s_{ij} = s_{max} \Longleftrightarrow x_i = x_j \ \ \forall x_i, x_j \in X$

The semantics of $s_{ij} > s_{ik}$ is that instance i is more similar to instance j than to instance k.

Belanche also defines the concept of similarity aggregator, which is a function that combines some similarity measures preserving the original properties of the similarities. One basic approach of aggregator is the one called Heterogeneous Euclidean-Overlap Metric function, *HEOM* [7]. It defines the aggregator as the squared root of the squared partial similarities sum. The variables can be categoric and be treated with overlap function or continuous and be treated with continuous range function. *Missing values* are considered to have the maximum distance (minimum resemblance).

In depth, Belanche defines the aggregator as:
For a set of n similarity measures $s \equiv \{s_1, s_2, \dots, s_n\}$, where $s_i \in [0, s_{max}]$, with $s_{max} > 0$. A similarity aggregation is a function $\Theta : [0, s_{max}]^n \to [0, s_{max}]$ fulfilling some properties (minimality, symmetry, idempotency, continuity, etc.) which introduce a specific semantics that express:

1. Even small contributions can only *add* something in favour of the overall measure.
2. The eventually missing pieces are regarded as *ignorance* and do not contribute in favour nor against the overall measure.

The following is a valid family of similarity aggregators, such as Belanche defined it:

$$\Theta(s) = f^{-1}\left(\frac{1}{m}\sum_{i=1}^{m} f(s_i)\right) \tag{3.1}$$

where f is a strictly increasing and continuous function such that $f(0) = 0$ and $f(s_{max}) = s_{max}$.

The heterogeneous similarity [1], is basically the aggregation of the partial similarities calculated independently for each variable using the most suitable (heterogeneous) similarity measure.

In order to do the comparison process more understandable, most authors prefer the similarity because it is a bounded value ($s \in [0, 1]$). With this unified criteria you can imagine better how resemblant every two examples are.
In relation with this, some authors have exposed their partial similarity measures and the way to combine them, that is, their similarity aggregator [6] [1]. Gower explains different similarity measures for several types of variables as well as an aggregator that is able to cope with *missing values*:

$$G(s_1, \dots, s_m) = \frac{\sum_{k=1}^{m} s_k \delta_k}{\sum_{k=1}^{m} \delta_k} \tag{3.2}$$

where s_k is a partial similarity to be aggregated and $\delta_k \in \{0, 1\}$ is an partial similarity associated value which is equal to 0 only when s_k is *missing*.

All this is extended by Belanche, who exposes new partial similarity measures (he recognizes new data types) and a generalized aggregator based on the *generalized* or *power mean* with exponent q. This is the definition of his similarity aggregator:

$$\Theta(s) = M_q(s_1, \dots, s_m) = \left(\frac{1}{m}\sum_{k=1}^{m} (s_k)^q\right)^{\frac{1}{q}}, \; q \in \mathbb{R} \tag{3.3}$$

Note that this is the general aggregator (3.1) using $f(z) = z^q$.

Changing the q value, you obtain different means (-1: harmonic mean, 0: geometric mean, 1: arithmetic mean, 2: quadratic mean...) and it fulfills the property called *generalized mean inequality*: if $p < q$ then $M_p(s_1, \dots, s_m) \leq M_q(s_1, \dots, s_m)$ and the two means are equal if and only if $s_1 = s_2 = \cdots = s_m$.
Belanche uses this equation to aggregate the partial similarities but also to solve the problem of the *missing values*. In this way, he replace the missing-value partial similarities with the aggregation of non-missing-value partial similarities:

$$\{s_1, \dots, s_m\} \equiv \left\{(s_k)_{k \in \mathscr{C}}, h \cdot M_q((s_k)_{k \in \mathscr{C}})\right\}$$

where $\mathscr{C}$ is the index set of non-missing-value similarities and h is the size of this set. So, the h partial similarities that are *missing* ($(s_k)_{k \notin \mathscr{C}}$) are replaced with the aggregation of the rest of partial similarities which are not *missing values*, $M_q((s_k)_{k \in \mathscr{C}})$.

Missing information is a very common problem in data analysis because there are many causes for the absence of a value. The *missing values* problem acquires more relevance when its number arises. There are basically three ways of dealing with them: completing the data gaps, extending the learning methods to cope with incomplete data or forgetting instances with *missing values*. This third approach is the worst because you are wasting data and its use is only justified if the quantity of data available is large enough, and the proportion of instances affected is small [8]. But it is possible the *missing values* to be normal in a situation (a sensor doesn't work if another one achieves some position, for example). In this case, deleting this instances is a serious error.
On the other hand, the first approach is maybe the most common used one, replacing the *missing value* with some other value which could be justified (mean of the variable, mode...). A complementary action would be to add a new variable indicating the instances that have *missing values* in an specific variable. Thus, all variables with some *missing value* would go with a new variable of these characteristics, and it would not be incompatible with replacing into the original variable.
Finally, the second approach is the one that follow [9] and [6]. They use the known information to estimate the final similarity despite of *missing values*. Using the known values, their methods estimate which would be the possible effect of the *missing values* over the final similarity. The main difference with the heuristic classical methods is that they estimate a new value for each *missing value* whereas, from this point of view, the final aggregated similarity function is estimated dealing with *missing* information.

3.B Learning methods

In the situation we are working, where there is a data set to be learnt, machine learning methods can be separated in two main groups attending to the use of the class variable that the method does. If the method doesn't need class variable to perform, it is an unsupervised method. In the other case, when the class variable is used, the method is called supervised.
The supervised learning tries to learn to relate some characteristics of the examples with an concept. The objective is to be able to predict, later and using these relations, which is the concept which a new example belongs to. Otherwise, the unsupervised learning tries to create a good representation of how the examples are organized. With this representation, some concepts could be inferred.

3.B.1 Clustering algorithms

A kind of unsupervised learning is the *clustering*, where the instances are grouped attending to some measure of similarity between them. One of these methods is the *Leader2* [2]. An adapted version of this algorithm has been used in this thesis to initialize the first layer of the network. It was chosen basically because it works with heterogeneous data and uses similarity measures.
Others characteristics of the *Leader2* algorithm are that it only needs one parameter (s_{min}) to perform and the leaders (reference instances) are instances of the data set. Also, it preserves all the properties of the former leader algorithm [3] and, furthermore, *Leader2* adds new ones. These properties are:

1. For any instance (i), the similarity with its leader is higher than s_{min}.
2. For any two leaders (l,k), the similarity between them is lower than s_{min}.
3. For any two instances (i,j), if they are the same, then they have the same leader.
4. For any instance (i), the similarity with its leader is higher than its similarity with any other leader.
5. The lowest similarity of an instance (i) with its leader will be higher than the highest similarity between two different leaders.

This algorithm covers all the instances of the data set taking one at anytime and evaluating if it can belong to any cluster already created. If it can not, a new cluster will be created using this new instance. After processing all the instances, several clusters will have been created. The number of clusters can not be estimated before starting the learning process, but it is possible to stablish a relationship with the s_{min} parameter: a higher s_{min} (tends to 1), implies a higher number of clusters.

3.B.2 Artificial Neural Networks

With respect to supervised learning techniques, in this thesis we have develop a new kind of *Artificial Neural Network* that follows this learning paradigm. There are a lot of types of *ANN* models and the most important ones are well explained at [8] and [10]. The *ANN* consists of an interconnected group of neurons that processes information using a connectionist approach. Each neuron is a process unit and their combination, distributed in layers, builds the model response. There are many classifications for the different networks and one of them divides the *ANN* models attending to whether it allows feedback between the different layers or not. In this sense, the *ANN* model that only allows forward communication between the layers is called *feedforward ANN* (figure 3.1).

Traditionally, some of the *ANN* models have been more frequently used than others ones. The most populars *ANN* are the *Multilayer Perceptron* and the *Radial Basis Function Networks* [8]. These last networks use a two-layer feedforward network (Fig. 3.2) where the hidden neurons implement a radial function. The neurons on the second layer, output neurons (usually only one), perform the linear combination of the outputs of the first layer multiplied by their weights. Output neurons usually incorporate another input, which is a constant input associated to an independent weight (w_0), the bias. So, the first layer neurons have m inputs and the second layer ones have $h+1$ inputs. The model of the output neurons is simple. For the k^{th} output neuron, the function is:

$$y_k(x) = \sum_{j=1}^{h} w_{kj}\varphi_j(x) + w_{k0} \tag{3.4}$$

But the function of the hidden neurons is, as we said before, a radial function, which requires two parameters: the width (σ) and the center (μ). The center has to be an instance comparable with the inputs and the width is a real number.
The function is the following:

$$\varphi_j(x) = \exp(-\frac{\|x-\mu_j\|^2}{2\sigma^2}) \tag{3.5}$$

where $x \equiv \{x_1,\ldots,x_m\}$ and $\mu_j \equiv \{\mu_{j1},\ldots,\mu_{jm}\}$.

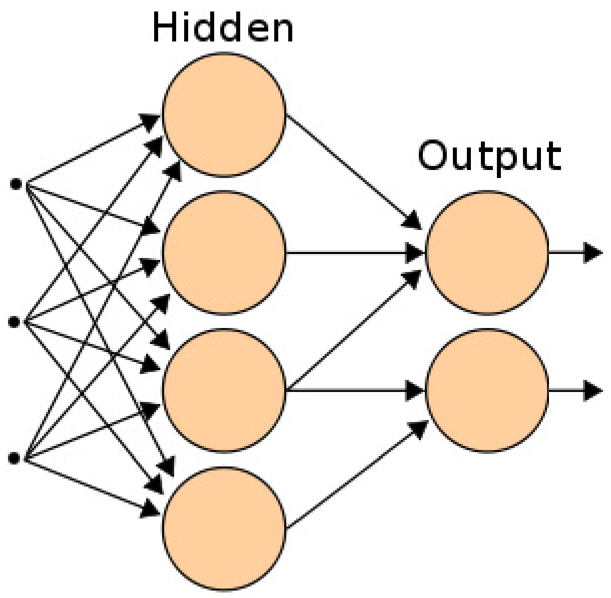

Figure 3.1: Two-layer feedforward *ANN*.

Here two basic question stand out: which is the right σ? and which μ has to be used at each hidden neuron? Then, there are some others, as if it is better to use one or several σ's, but this is a question that maybe is easier to answer in an experimental way.
For the first question, many different approaches have been applied to give a value to this parameter. At [11], the authors defense the idea of choosing the right value using *Cross Validation*. Others prefer heuristic functions to get this value [8] [10] [12].
For the second question, the number of radial functions is determined by the complexity of the mapping to be represented rather than by the size of the data set [8]. In this way, many approaches use an unsupervised learning method, like a clustering algorithm, to initialize the network. The number of clusters (complexity) is used as number of *hidden neurons* and the clustering centroids as centers of the functions.

In order to avoid overfitting, some functionalities can be introduced in the model construction procedures, as the one called *regularization*. There are many ways to perform it, but the approach most widely used in *ANN* is penalizing large weights. They are associated with complexity because they usually mean a higher effort in fitting the data.
In this way, Orr [13] [14] has develop a great job in regularization for *RBF* networks. Since they have a two-layers structure, it is only possible to perform regularization over the weights of the second layer because the weigths of the first layer are the clustering centroids.
In this situation it is possible to perform two kind of regularizations: the standard one (only one regularization parameter, λ [13]) or the local regularization (one parameter per weight, λ_i [14]). Also different methods can be used when we are dealing with regression problems, where the most used regularization method is called *Ridge Regression*, or classification ones.

So, the base of this research project is at [9], where Belanche proposes an *ANN* model, the *Heterogeneous Neural Network*. It is inspired by *RBF* networks, but really it is a sort of generalization. The *HNN* is an *ANN* that copes with heterogeneous data and uses the similarity measures to build the learning model. Its structure is the same that the *RBF* network structure,

but with an only difference: the neuron model chosen is the S-neuron.
The S-neuron (*similarity-based neuron*) is an H-neuron (*heterogeneous neuron*) where the function used is a similarity function (or a similarity aggregator). Then, a feed-forward *Heterogeneous Neural Network* (*HNN*) is a feed-forward artificial neural network where the hidden neurons of the first layer are S-neurons and all their values are transferred to the second (linear output) layer (Fig. 3.2).

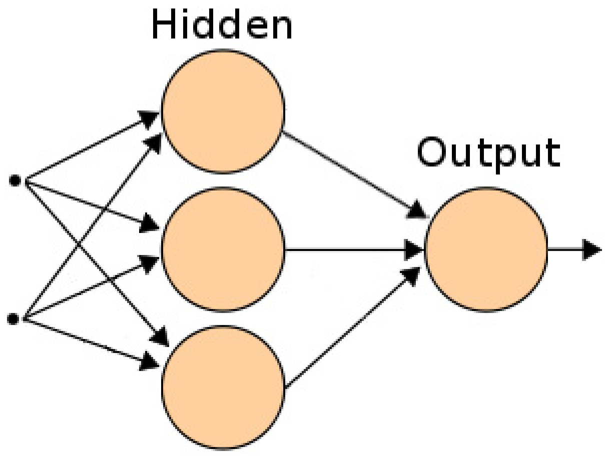

Figure 3.2: Configuration of *RBF networks* and *HNN*.

All the data is heterogeneous in this approach, which means that even the first layer weights allow *missing values* since they are instances of the input data set. The S-neuron adds a non-linear component to the linear aggregator and, in this way, Belanche [9] proposes a family of sigmoidal functions to operate in the $[0,1]$ interval:

$$f(x,p) = \begin{cases} \frac{-p}{(x-0.5)-a(p)} - a(p) & \text{if } x \leq 0.5 \\ \frac{-p}{(x-0.5)+a(p)} + a(p) + 1 & \text{if } x \geq 0.5 \end{cases}$$

$$a(p) = \frac{-0.5 + \sqrt{0.5^2 + 4 * p}}{2} \tag{3.6}$$

where $p > 0 \in \mathbb{R}$ is a parameter controlling the shape of the function inside of the family (Fig. 3.3). Belanche proposes $p = 0.1$ because it shows an averaged behaviour. The function fulfills $\forall p \in \mathbb{R}^+,\ f(0,p) = 0,\ f(1,p) = 1,\ \lim_{p \to \infty} f(x,p) = x$ and $f(x,0) = H(x-0.5)$, being H the Heaviside function.

The advantages of the S-neuron are many: it is intuitive, fulfills all the desired properties for aggregation, is computationally cheap and it doesn't take assumptions about how the partial similarities are computed.
Then, the *Heterogeneous Neural Network* calculates:

$$y_k = \sum_{i=1}^{h} w_{ki} \varphi_i(x) + w_{k0}$$

$$\varphi_i(x) = f(x,p)$$

where f is the function 3.6, $p > 0$ and $k \in \{1 \ldots t\}$.

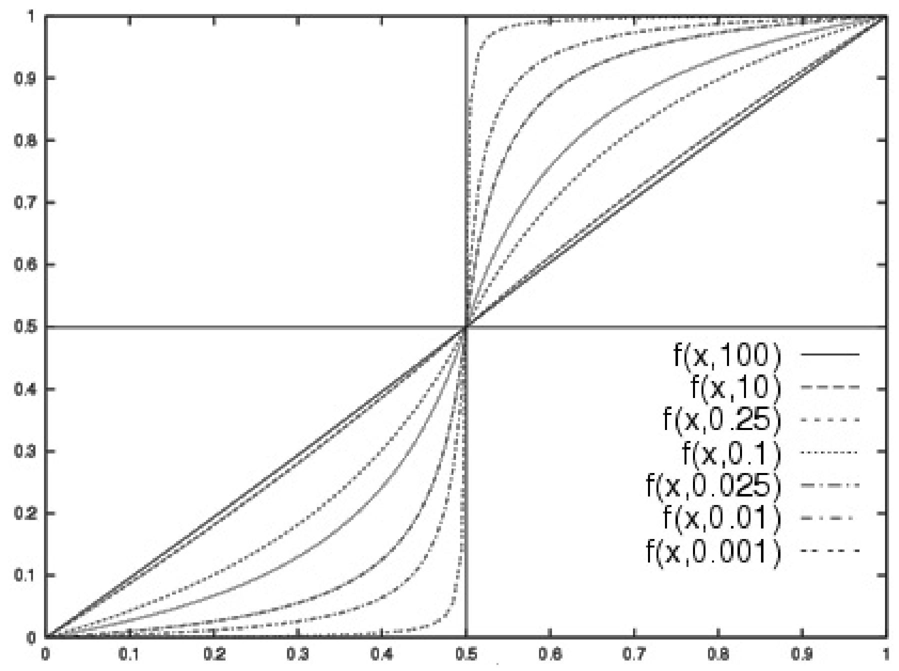

Figure 3.3: Family of sigmoidal functions $f(x,p)$ for different values of p.

4 Methodology and strategies to solve the problem

This thesis is a project that has been developed in six months, and since it is a long time work, it is necessary to plan the methodology. The work has been distributed around all the period, dividing it into several clear tasks and imposing a working methodology.

4.A Methodology

The method we used in the development of the thesis can be divided in four basic subtasks:

1. *Idea*. At any point, before starting the research and the documentation, you have to know exactly what do you want or what you are going to search. This is important because the idea is the fact that guides the course of the documentation and it helps to avoid situations where the field of the idea grows excessively and you lose the way.
   When a situation like this happens, the idea have to be adjusted if you can find an interesting way inside the wide grown field, reconsidered if you realize you were wrong or left if any interesting way can not be found.
   New ideas, not initially thought, can emerge during the development of the project. In this case, you have to evaluate if you are interested in it and, in a positive case, apply this research method to it.

2. *Documentation*. It is the first stage of the research, when you look at previous works of people that have worked before in the field of the research.

3. *Proposal*. This is a key task, where you propose your own solution or application if you think that there is a gap which you are able to fill (because you have the required knowledge and this new idea).

4. *Implementation*. It is the moment when the idea becomes something tangible. You have to build the idea taking into account the considerations of the whole implementations and respecting all the rules. The intention is to add the idea to the system without affecting parts no related with it. After this task, you will have the same system but with a new functionality or with a previous functionality redesigned.

5. *Test*. It is the last step, where you have to verify the usefulness of this idea. Sometimes this step it is not necessary because the results can not be verified, at least, in a short time.

Otherwise, when it has to be performed, the results can be differents attending to the type of problem.
If there is a single solution, you have to evaluate its contribution. If it is a basic component of the project, the idea has to be beneficial for it. If not, if it is a secundary or optional functionality, it has to do a essencial contribution in some specific situation.
In some cases, the solution is not unique, but there are several approaches. In this case, you have to return the default, principal or most relevant solution, which will be looked up first.

We follow this method at two level. In a wide level, we follow it to develop the whole thesis. In a close level, we follow it to develop each small idea that integrates the thesis. Following this method have allowed us to develop our project.

4.A.1 Planning

This master thesis is targeted to improve the *Heterogeneous Neural Network*, getting a better version that we have called *Heterogeneous Neural Network 2*. But, in a paralel way, we have incorporated to the *HNN2* the necessary functionalities to work like a *RBF network*. All this work was divided into ten principal points:

Data types We have to decide which data types incorporate in this version of the algorithm: continuous, categoric, binary... This classification is only a first version; a requirement of the method is that the algorithm has to be extensible.

Similarity measures For each data type, there are many different similarity measures and we have to choose several (per data type) to implement. In any case, this similarity measures list could be extended.

Missing values The objective is to develop an algorithm where the method to deal with *missing values* can be changed. But the solution to this point has to be a proposal with a default method, and some other proposals, to cope with *missing information*.

Aggregator Dealing with heterogeneous data is a problem which we will try to solve using partial similarities (similarity per variable). Then, these similarities have to be combined to get the whole similarity, and that is what an aggregator does. The aggregator has to fulfill some properties and we will look for a good approach to apply.

Leader2 This method was developed, in its first version, for general porposes. Now we will get a new specific version adapted to all these functionalities we want to incorporate to *HNN2*. We want to develop also an *Leader2* distance version.

Neural Network structure This thesis is based in the *Leader2* and in the *HNN*. This neural network is the base for the development of the *HNN2*. We will build a two-layer feed-forward neural network and we have to choose the method to calculate the weights of the second layer. Like a secondary experiment, from this point we have in mind developing a paralel *RBF* network based on the same theory and using the *Leader2* algorithm as initializer.

Regularization We will incorporate a regularization method that, during the building of the second layer, keeps reasonable weights in order to avoid overfitting. In this case, we have to decide also between a local version (several regularization parameters, one per weight) or a clasical version (only one parameter). This process could be different in regression to in classification, so these possibilities have to be also explored to cover both. Finally, this(these) parameter(s) can be optimized.

Standard deviation (p/σ) The second parameter of the hidden neurons and the only one that we can change once the centers have been fixed has to be adjusted. We have to choose a first value for it and some heuristic function is the first option. But before, we have to decide if we will use several parameters (one per hidden neuron) or only one. This(these) parameter(s) is also optimizable and, for this reason, we propose an *Alternate Optimization* in order to optimize this and regularization parameter together.

Data set choise The problems which have been chosen to test this learning algorithm have to have heterogeneous data and *missing values*, for a correct test. It is necessary also to test all the kinds of problems, so the set of problems has to be representative (basically, regression/classification).

Experimental setting . With this point we will try to demonstrate that the learning method performs well and obtains good results. In order to do this, some tests have to be performed: looking for the best configuration of the algorithm, and others comparating it with different algorithms.

But, some other tasks have to be carried out in a paralel way: writting this final report, improving the efficiency of all the improvements that we add to the algorithm…

Finally, some of these points were grouped in sets of very closely linked ones. This is the case of *Data types* and *Similarity measures* points, which are directly liked because depending on which data types are selected, you have to provide different similarity measures. We also grouped *Aggregator* and *Missing values* points, because the *missing value* treatment is performed during the aggregation process.

4.A.2 Guidelines

For the implementation, we decided to use the R environment. R is a language and environment for statistical computing and graphics [15]. R provides a wide variety of statistical techniques (linear and nonlinear modelling, classical statistical tests, classification, clustering…) and is highly extensible. It can be run on a wide variety of platforms (Linux, Windows, MacOS…). It makes data manipulation, calculation, matrix calculations, many tools for data analysis and others functionalities of classical programming languages.
R allows users to add additional functionalities by defining new functions. There are several directories from which you can get many packages with non-default R funcionalities. For computationally-intensive tasks, C/C++ and Fortran code can be linked and called at run time.
All these characteristics were enough to convince us. Here we have all we need to implement the theoretical conclusions and to demonstrate their practical utility.

In addition, an important criteria in the implementation of the thesis is *modularity*. Our objective is to get a method easily extendable. In this way, basic units of code have to be implemented in separated functions which could be changed easily by the user if it were necessary. That is basic in this thesis, because we work with many similarity measures, for example, and maybe users could want to use their own measures.

Of course, another basic criteria is *efficiency*. R is not prepared for complex weighted programs, and for this reason they allow users to implement these complex parts in C/C++, which do a more efficient use of the computer resources. So we have to implement a R code as efficient as possible, leaving for further work its implementation in other language.

The third basic criteria is the use of *justified decisions*. All the changes incorporated to the learning algorithm have to be justified by either the methodology explained above or a reference to a previous work that justifies the decision.

4.B Development: decision and implementation

In this section you can look up all the decisions that we have taken for the final implementation of the *HNN2*. We will respect the work division done for the development of the thesis and we will use the same structure in this document. Only we leave the experimental settings out of this section because they have their own section later.

4.B.1 Data types and Partial similarity measures

We consider that there is an space (X) from where we take some examples to represents a particular case of something. These examples are allways collected in the same terms taking into account the same characteristics. So, the i^{th} example could be represented as a vector x_i with m components, and each one of them is a characteristic of the described object. In this way, the k^{th} variable (characteristic) of the example is represented as x_{ik}.
According to Belanche [9], a similarity measure is an unique number expressing how “resemblant” two patterns are, given these characteristics. Then, a similarity between two instances ($x_i, x_j \in X$) is represented as $s(x_i, x_j)$ or using an equivalent notation: s_{ij}. But in this thesis, the similarity is applied to each variable independently and then they are aggregated later; the meaning is the same but the representation is $s(x_{ik}, x_{jk})$ or s_{ijk}.
In practice, we define the similarity as a measure enclosed into the interval $[0,1]$ and that is greater the most similar the examples are. Then, the maximum similarity happens when you compare an example with itself or with another one that is identical.

Attending to the variables, it is possible to find different types of variables and to use different similarity measures to compare a variable. The data types and similarity measures selection we present here is based on the Belanche’s selection [1]. But also, we have used some of the Gower’s concepts [6].

Now we present the list of similarity measures, separated attending to the data type they require, which were chosen to be included in this project. Note that the *HNN2* has been written in some specific way that allows users to incorporate their own similarity measures, so this list is only the default collection.

Binary variables

We consider that a variable is binary if only two situations can happen ([$+$]: presence/ [$-$]: absence) and absence in both of a pair of examples is not taken as a match [6]. In this case, treating the variable as purely two-valued categoric can result in a loss of information because you are giving to a value more importance that what it really has. Actually, you usually can know which of the two matches is the relevant one (true-true or false-false) and build the binary variable based on this.
The difference between a two-valued categoric variable and a binary variable is that in the latter the two values are exclusive and the two only possibilities. For instance, a variable *color* with possible values *black* and *white* would be categoric, whereas a variable *is-black?* with possible

values *yes* and *no* would be binary. Realize that in the first case you can incorporate new colors but in the second one no values can be added. But sometimes the difference is more conceptual than practical.
In the literature there are many similarity measures for binary variables, where the negative match treatment is different attending to the author. There are some approaches that propose all the binary variables are treated together (*Jaccard* similarity, *Simple Matching*...). Nowadays, it is more common the separated comparison of binary variables. In this way there is the frequency-based Belanche's approach [9].

Belanche defines his proposal as:

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ 1-P_{ik} & \text{if } x_{ik} = x_{jk} \end{cases} \tag{4.1}$$

where $x_{ik},x_{jk} \in 0,1$ are two binary values of two instances $x_i,x_j \in X$ and P_{lk} would be the fraction of values of variable k that take on the same value than x_{lk} over the whole space X_k, which is the space for the k^{th} variable. We calculate it taking into account only the instances in the sample (data set) because we usually don't have the real distribution of the variable.

If the values are different, there is not similarity (you are comparing *presence* with *absence*). If they are the same value, the similarity is the inverse of the value probability.
For a variable k, it is more particular finding the same value at two instances (x_{ik}, x_{jk}) when it is an strange value than finding the same value at two instances when it is the most usual value for this k variable. For this reason, we think that the first case has to have a higher similarity than the second one and the simplest function that does this is: $h(z) = 1-z$ and $z = P_{ik}$.

But we introduce here some improvements in order to solve some situations.
In the first case, we propose a more complex function h to invert the probability. We know that the probability is a real number between 0 and 1. For this reason, the h function has to be a function that performs the transformation $[0,1] \to [0,1]$ inverting the behaviour of the original function. In order to get this behaviour, h function has to be decreasing. In this way, we take into account three possible functions (you can see them in the figure 4.1):

$$\begin{aligned} h_1(z) &= 1-z \\ h_2(z) &= 2^{1-z}-1 \\ h_3(z) &= \frac{1-z^2}{1+z^2} \end{aligned}$$

The second improvement is based on the Laplace correction. The data set is a sample of the real space (X), from where some problems can be generated. If our binary variable is out of balance and the probability of some of the two values is near to 0 or 1, we would get a sample with only one value (useless constant variable). Using this similarity function this situation would generate a global maximum or minimum that is not real. Then, we suggest the probabilities to be corrected in advance in this way:

$$\begin{array}{ll} P_{ik} = \frac{p}{m} & \text{if } P_{ik} = 0 \\ P_{ik} = 1-\frac{p}{m} & \text{if } P_{ik} = 1 \end{array}$$

where $p \geq 1$ and $P_{ik} \in (0,1)$. We propose $p = 1$ because it is the simplest behaviour.

Finally, the similarity function is defined as:

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ h(P_{ik}) & \text{if } x_{ik} = x_{jk} \end{cases} \tag{4.2}$$

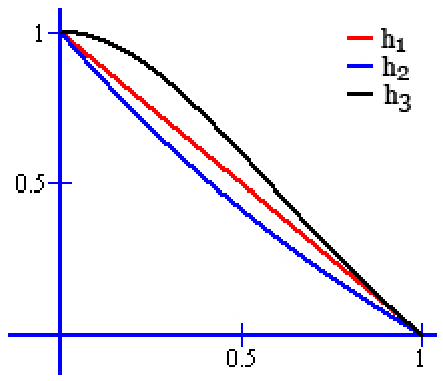

Figure 4.1: Graphical representation of the three possible h functions.

The algorithm incorporates others similarity functions for single binary functions. The first one 4.3 was proposed at [6], which is an approach that returns 0 if the values are **not** equal, 1 if they are equal and this value is positive, or $\mathscr{X}$ if they are also equal but the value is negative. The second one 4.4 is the overlap function applied to binary variables, which returns 1 if the values are equal or 0 if they are not.

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ 1 & \text{if } x_{ik} = x_{jk} \text{ and } x_{ik} = 1 \\ \mathscr{X} & \text{if } x_{ik} = x_{jk} \text{ and } x_{ik} = 0 \end{cases} \tag{4.3}$$

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ 1 & \text{if } x_{ik} = x_{jk} \end{cases} \tag{4.4}$$

Some other approaches where the similarity is calculated taking into account all the binary variables together has been included in this learning algorithm. But they are not literally implemented because using the aggregation functionality (see later), it is possible to simulate them. In concret, we have Jaccard similarity (4.17), that uses 4.3, and Simple Matching (4.18), that uses 4.4.

Categoric variables

Usually, it is assumed that categoric variables are that variables which have two or more (not infinite) possible values, no order exists among these values and the only possible comparison is equality. Now $x_{ik}, x_{jk} \in Xc$ are not more binary values, but categoric. X_k is the categoric space which defines the different possible values of the k^{th} variable. The number of possible values can be defined as $\|X_k\| \in \mathbb{N}$ and $2 \geq \|X_k\| < \infty$, but it is not usually huge (an one-digits or two-digits number).
The basic similarity measure for these variables, which has been named already in the previous subsection (4.4), is the *overlap*. It is defined in the same way but two or more possible values as:

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ 1 & \text{if } x_{ik} = x_{jk} \end{cases} \tag{4.5}$$

Based on our binary similarity function, the probability approach, we have extended a similar version for categoric variables. The only difference is again the number of possible values.

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 0 & \text{if } x_{ik} \neq x_{jk} \\ h(P_{ik}) & \text{if } x_{ik} = x_{jk} \end{cases} \tag{4.6}$$

The change in these two last similarity measures regarding their binary version is a change related to the kind of variable that actually doesn't affect the similarity functions because they work in the same way with 2 or any finite number of possible values.

Ordinal variables

These variables can be seen as a brigde between the categoric and the continuous variables. It is assumed that the values of an ordinal variable form a linearly ordered space $(X_k, \leq)$ and the number of them is finite.
For this kind of variables we use the Lin's proposal [16], which is defined as:

$$s_k(x_{ik}, x_{jk}) \equiv \frac{2log(\sum_{l=i}^{j} P_{lk})}{logP_{ik} + logP_{jk}} \tag{4.7}$$

where $x_{ik}, x_{jk} \in X_k$, $x_{ik} \leq x_{jk}$, P_{lk} would be the fraction of values of the k^{th} variable that take on the value x_{lk} over the whole space X_k and the summation run through all the values x_{lk} such that $x_{ik} \leq x_{lk} \leq x_{jk}$.

Continuous variables

The following group of variables are the continuous. A continuous variable is any real variable that can take infinite values in a range $((r^-, r^+)$, where $r = r^+ - r^-)$. In an standard situation, the range is all $\mathbb{R} : (-\infty, +\infty)$, but it can change attending to the particular conditions of each variable.
In this way, a large number of similarity functions have been proposed. But also distance functions can be taken into account because, knowing the range of these variables, distance functions can be bounded in the $[0, 1]$ interval by dividing the classical distance by the variable range, $[0, +\infty) \rightarrow [0, 1)$. Then distance can be converted into similarity using the relation $similarity = 1 - distance$.
In practice the range usually is unknown and that is why we calculate the range based on maximum and minimum values of the continuous variable in the sample $\hat{r} = \max(x_{ik}) - \min(x_{jk})$ when it is necessary. If the sample is not so good, the possibility of getting a value out of the range rises, which generates a conflict situation. The easier way to solve this problem is to preprocess new data and changes the value out of the range by the bound value $(\forall x_{ik} > r^+, x_{ik} = r^+$ and $\forall x_{jk} < r^-, x_{jk} = r^-)$.

We have incorporated some different measures. For any two values $x_{ik}, x_{jk} \in \mathbb{R}$:

$$s_k(x_{ik}, x_{jk}) \equiv 1 - \frac{|x_{ik} - x_{jk}|}{r_k} \tag{4.8}$$

where r_k is the range of the k^{th} variable.
The second similarity measure is based on the classical distance measure called *canberra*. This distance, in its original state, only works with non-negative values, so it is bounded by the interval $[0, 1)$.

$$s_k(x_{ik}, x_{jk}) \equiv 1 - \frac{|a - b|}{|a| + |b|} \tag{4.9}$$

where the subtraction is used to convert distance into similarity: $[0, 1) \rightarrow (0, 1]$.

The same subtraction is used in the next function to get a similarity measurement from a distance function:

$$s_k(x_{ik}, x_{jk}) \equiv 1 - \min\left(\frac{|x_{ik} - x_{jk}|}{p}, 1\right) \tag{4.10}$$

where $p = 0.5$. This p value is key in this function, where it says the scope of the measurement. For two *far* values x_{ik}, x_{jk} the similarity is 0 and then, the nearer values, the higher similarity. p defines here the *remoteness* concept (how far they are) and attending to it we obtain the step where the similarity begins being higher than 0.

Finally we include a last continuous similarity measure, that is define as:

$$s_k(x_{ik}, x_{jk}) \equiv \frac{1}{1 + |x_{ik} - x_{jk}|} \tag{4.11}$$

Note that we don't separate integer from continuous variables. Given that $\mathbb{N} \subset \mathbb{R}$, any similarity in $\mathbb{R}$ is also valid in $\mathbb{N}$.

Fuzzy variables

Variables represented as fuzzy sets has been studied in depth by Belanche [17] [9]. Several fuzzy similarity functions have been proposed and different choices are possible. In possibility theory, the result is the likeliness of co-occurrence of two vague propositions, with a value of absolute certainty.
We understand *fuzzy* as describing the vague observation of an object. When the measurement of the specific value of a variable is done, some imprecision can be introduce in the result in the form of indetermination. This is useful in situations where we consider that two close values have some similarity but when they are far to a certain extent, there is no similarity. This happens usually in the real life. For example, attending to the person people age, two people of 12 and 14 years old are similar between them (probably, they are students) and they are not similar with someone of 40 (working), or with someone of 70 (probably retired).
Our proposal is based on fuzzy numbers. A *fuzzy number* in X_k (the reference set) is a convex and normalized fuzzy set F with piecewise continuous μ_F, where symmetry of μ_F is not required [9].

The first of the approaches we have included in this learning algorithm is based on a triangle of fixed dimensions for all the values 4.2.

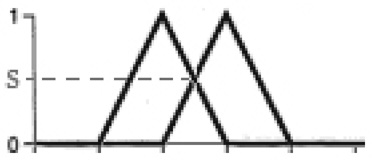

Figure 4.2: Triangle representation of fuzzy sets.

$$s_k(x_{ik}, x_{jk}) \equiv \begin{cases} 1 - \frac{|x_{ik} - x_{jk}|}{2 \cdot P} & \text{if } |x_{ik} - x_{jk}| \leq (2 \cdot P) \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

where $(2 \cdot P)$ is the bottom side length of the triangle that represents the fuzzy number. In our proposal, it is a fixed value at $P = 1.5$. You can see the way we follow to get this function at annex A.4.

The opposite effect is the following, where the shape of the convex function depends on some percentage of the value that represents.

$$s_k(x_{ik}, x_{jk}) \equiv \begin{cases} 1 - \frac{|x_{ik}-x_{jk}|}{(x_{ik}+x_{jk})\cdot P} & \text{if } |x_{ik}-x_{jk}| \leq ((x_{ik}+x_{jk})\cdot P) \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

where P is the percentage that allows the function to calculate the width of the shape. The bottom side of the triangle that represents the fuzzy number has a length of $2 \cdot x_{lk} \cdot P$. In our proposal, $P = 0.05$, that is a 5%. This approach is the only one that we propose where the convex shapes depend on the values that they represent. You can see the way we follow to get this function at annex A.5.

The following function has been built in the same way, but using the shape of a trapezoid 4.3 instead of the shape of a triangle. Doing that you are supposing that the imprecision level in the variable measurement is higher and you want to secure that the very near values are considered the same (without any devaluation).

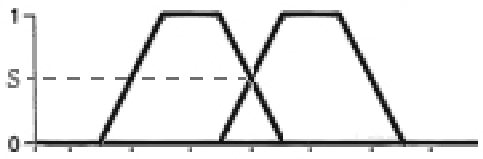

Figure 4.3: Trapezoid-shaped representation of fuzzy sets.

$$s_k(x_{ik}, x_{jk}) \equiv \begin{cases} 1 & \text{if } |x_{ik}-x_{jk}| \leq (2\cdot P_a) \\ 1 - \frac{|x_{ik}-x_{jk}|-(2\cdot P_a)}{2\cdot(P_b-P_a)} & \text{if } |x_{ik}-x_{jk}| > (2\cdot P_a) \text{ and } |x_{ik}-x_{jk}| \leq (2\cdot P_b) \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

where $(2 \cdot P_a)$ is the upper side length of the trapezoid and $(2 * P_b)$ is bottom side length of it. In our proposal, they are two fixed values at $P_a = 0.5$ and $P_b = 1.5$. You can see the way we follow to get this function at annex A.6.

It is possible to use other functions that use curved shaped. We incorporate this approach that uses an exponencial function as convex shape of the fuzzy number 4.4.

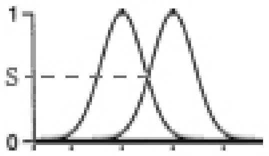

Figure 4.4: Gaussian-shaped representation of fuzzy sets.

The function returned of crossing two equal exponential functions, centered at different points, is used as similarity function:

$$s_k(x_{ik}, x_{jk}) \equiv e^{-\frac{\left(\frac{(x_{ik}+x_{jk})}{2}-x_{ik}\right)^2}{2\cdot\sigma^2}} \tag{4.15}$$

where σ^2, that states the width of the exponential function, has to be set. In this approach, there is an unique $\sigma^2 = 0.5$ but it could depend on the value that the function represents. You can see the way we follow to get this function at annex A.7.

Remember that the reasoning done to arrive to these fuzzy functions is explained in depth at the annex A.

4.B.2 Aggregator and Missing values

One of the porposes of this master thesis is that the learning algorithm resulting of this research works with heterogeneous data. We have decided to implement a method where each variable could use its own similarity measure (partial similarities). That allows users to take all the knowledge into the data if they really know the data characteristics.
But, in order to support this idea, we need a function that combines all this partial similarities in a final global one. This is the reasoning that we have followed to look for an aggregator that combines well the partial similarities.
Gower [6] proposed his version that was based on the arithmetic measure. A more complex aggregator was the one proposed by Belanche [1]. He assures that the aggregation functionality fulfills a semantic role, in the way it has to preserv some properties of the similarity measures.

In the way we have defined *similarity measure*, before the combination we have a vector $s \equiv \{s_1, s_2, \ldots, s_m\}$, where $s_i \in [0,1]$. A similarity aggregation is a function $\Theta : [0,1]^m \to [0,1]$ fulfilling some properties (Minimality, Symmetry, Monotonicity, Idempotency, Cancellation law, Continuity, Compensativeness...). This is an adaptation of the Belanche's work [1] on aggregation functions to our particular case, where we have m similarities between 0 and 1.
Belanche's work is useful because he assures that given $s \equiv \{s_1, \ldots, s_n\}$ (where s_i is the partial similarity for the object in X_i), a new measure obtained from any such aggregation operator $\Theta(s)$ is a similarity measure in $X = X_1 \times X_2 \times \ldots \times X_n$. If a valid aggregation is that that fulfills all the properties Belanches defined, a valid family of similarity aggregations could be:

$$\Theta(s) = f^{-1}\left(\frac{1}{m}\sum_{i=1}^{m} f(s_i)\right)$$

where f is a strictly increasing and continuous function such that $f(0) = 0$ and $f(1) = 1$.
A family of similarity aggregators that fits in this definition is:

$$\Theta(s) = M_q(s_1, \ldots, s_m) = \left(\frac{1}{m}\sum_{k=1}^{m} (s_k)^q\right)^{\frac{1}{q}}, \ q \in \mathbb{R}$$

which has been included by Belanche as aggregator in his works and it is obtained by taking $f(z) = z^q$. It fulfills the *generalized mean inequality*:

$$M_q(x_1, \ldots, x_n) \geq M_{q'}(x_1, \ldots, x_n) \Longleftrightarrow q > q'$$

This family includes some well-known means depending on q. For example, the *arithmetic mean* is the mean you get if you use $q = 1$. Others means are the *geometric mean* with $q = 0$, the *quadratic mean* with $q = 2$ or the *harmonic mean* with $q = -1$. The family is known as *generalized means* or *power means*.

We consider that this is a good option and we have decided to maintain this family as aggregator in our learning algorithm, but it will incorporate some additional functionalities, as we will see below.

***Missing values* in the aggregation**

A basic characteristic of methods dealing with heterogeneous data that we have forgot expressly is the *missing values* treatment. We incorporate this functionality in the aggregator because we consider it is the best moment. Comparing two instances, the partial similarities for the most of the variables usually can be performed normally. Sometimes instances have some *missing values*, so the partial similarity can not be calculated, or the result of a partial similarity function is *missing*. In these two cases the partial similarity is $s_i = \mathscr{X}$, so the s vector changes:

$$s \equiv \{s_1, s_2, \ldots, s_m\}, \text{ where } s_i \in [0,1] \cup \{\mathscr{X}\}$$

This is a problem that have to be solved because the M_q functions are not able to cope with *missing values*. We are presenting a learning algorithm, so if some user would prefer to replace *missing values* before calling the *HNN2*, there is no problem. But we have to provide some funcionality to cope with them and the only option we take into account is to estimate the aggregated similarity using any $s_i = \mathscr{X}$ with the right value with this objective.
This is a problem that grows with the percentage of partial similarities that are *missing*. For example, if there are 10 variables and only 1 *missing value*, the aggregated similarity depends a 10% on this *missing*, it is no so important. But, if there are 9 over 10 that are *missing*, now we have a 90% of uncertainty, which is a very important problem.
In this way, we have proposed some possible approaches.

1. The first proposal is based on the Gower aggregator [6]. There, when a partial similarity s_i is *missing*, the flag that goes with this partial similarity (δ_i) is set to 0 (when it is usually 1). This does that the aggregator calculates the arithmetic mean without taking into account s_i. This approach supposes that the values that in this moment are *missing* would not change the mean.
   But, if you can not delete any partial similarity but you have to replace them in order to perform the aggregation, an equivalent behaviour is to use the mean of non-*missing value* partial similarities to replace the *missing values*.

   $$M_q(s_{i\in\mathscr{C}}) = M_q(s), \text{ where } s_i = \mathscr{X} \text{ is replaced by } s_i = M_q(s_{i\in\mathscr{C}})$$

   $$s \equiv \{s_1, \ldots, \mathscr{X}, \ldots, s_m\} \Rightarrow s \equiv \{s_1, \ldots, M_q(s_{i\in\mathscr{C}}), \ldots, s_m\}$$

   where $\mathscr{C}$ is the set of indexes that represent non-*missing value* partial similarities. That is true for any q and this is the demonstration:

   $$\left(\frac{1}{m}\left(s_1^q + s_2^q + \cdots + s_m^q\right)\right)^{\frac{1}{q}}$$

   $$\left(\frac{1}{m+1}\left(s_1^q + s_2^q + \cdots + s_m^q + \left(\left(\frac{1}{m}\left(s_1^q + s_2^q + \cdots + s_m^q\right)\right)^{\frac{1}{q}}\right)^q\right)\right)^{\frac{1}{q}}$$

   $$\left(\frac{1}{m+1}\left(s_1^q + s_2^q + \cdots + s_m^q + \frac{1}{m}\left(s_1^q + s_2^q + \cdots + s_m^q\right)\right)\right)^{\frac{1}{q}}$$

   $$\left(\frac{1}{m(m+1)}\left(m \cdot s_1^q + m \cdot s_2^q + \cdots + m \cdot s_m^q + \left(s_1^q + s_2^q + \cdots + s_m^q\right)\right)\right)^{\frac{1}{q}}$$

$$\left(\frac{1}{m(m+1)}\left((m+1)\cdot s_1^q+(m+1)\cdot s_2^q+\cdots+(m+1)\cdot s_m^q\right)\right)^{\frac{1}{q}}$$

$$\left(\frac{1}{m}\left(s_1^q+s_2^q+\cdots+s_m^q\right)\right)^{\frac{1}{q}}$$

This demonstration has been done for only one additional value. But easily it can be extended to several additional value.

So, using this property we implement the Gower approach replacing the *missing* partial similarity measures by the mean of the non-*missing value* partial similarity measures.

$$s_{i\notin\mathscr{C}} = M_q(s_{i\in\mathscr{C}})$$

2. The second method is a very basic approach: to replace the *missing* partial similarity measures by 0. Doing this you are considering that there is no similarity if one of the instances has some *missing* information. That is, you are isolating this example with respect to any other example, there is no a minimum similar case.

$$s_{i\notin\mathscr{C}} = 0$$

3. Another basic approach is to replace the *missing* partial similarity measures by $\frac{1}{2}$. Doing this you are considering that the *missing* information does the example more similar to any other example in average. Usually the middle is used as reference point in comparing two any things, so this same principle is behind our proposal. If similarity is defined between 0 and 1, the nearest point to any other is $\frac{1}{2}$. Latter we will talk about normalizing into aggregators, which will do this approach more interesting because right now, non-normalized variables can have their mean in a point different to $\frac{1}{2}$.

$$s_{i\notin\mathscr{C}} = \frac{1}{2}$$

4. The last approach is based on an interval of possible values. When we are calculating an aggregation of partial similarities, the result depends on all the partial similarities in the same way. So, if some of the partial results is *missing*, attending to the value used to replace the $\mathscr{X}$ and the non-*missing value* partial similarities, we can stablish an interval where the result could be. The center of the interval is the value we propose to replace *missing values*.
We define this interval as:
The lower bound (*lo*) would be the aggregation using 0 as value to replace $\mathscr{X}$. That is the minimum possible global similarity attending to the non-*missing value* partial similarities.
The upper bound (*up*) would be the aggregation using 1 as value to replace $\mathscr{X}$. That is the maximum possible global similarity attending to the non-*missing value* partial similarities.
The center would be calculated also as the aggregation of these two previous values.

$$lo = M_q(s_1,\ldots,s_m) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 0^q\right)\right)^{\frac{1}{q}}$$

$$up = M_q(s_1,\ldots,s_m) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1^q\right)\right)^{\frac{1}{q}}$$

$$s_{i\notin\mathscr{C}} = M_q(lo,up)$$

It could seem that the two last proposals are the same, because both are based on the idea of going towards the center if there are many *missing values*. But they only are exactly the same proposal for $q = 1$. Developing the previous function we get:

$$lo = M_q(s_1, \ldots, s_m) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 0^q\right)\right)^{\frac{1}{q}} = \left(\frac{1}{m}\sum_{i\in\mathscr{C}}(s_k)^q\right)^{\frac{1}{q}}$$

$$up = M_q(s_1, \ldots, s_m) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1^q\right)\right)^{\frac{1}{q}} = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1\right)\right)^{\frac{1}{q}}$$

$$M_q(lo, up) = \left(\frac{\left(\left(\frac{1}{m}\sum_{i\in\mathscr{C}}(s_k)^q\right)^{\frac{1}{q}}\right)^q + \left(\left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1\right)\right)^{\frac{1}{q}}\right)^q}{2}\right)^{\frac{1}{q}}$$

$$M_q(lo, up) = \left(\frac{\frac{1}{m}\sum_{i\in\mathscr{C}}(s_k)^q + \frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1\right)}{2}\right)^{\frac{1}{q}}$$

$$M_q(lo, up) = \left(\frac{\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}} 1}{2 \cdot m}\right)^{\frac{1}{q}}$$

$$M_q(lo, up) = \left(\frac{\sum_{i\in\mathscr{C}}(s_k)^q + \frac{1}{2}\sum_{i\notin\mathscr{C}} 1}{m}\right)^{\frac{1}{q}}$$

$$M_q(lo, up) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \frac{1}{2}\sum_{i\notin\mathscr{C}} 1\right)\right)^{\frac{1}{q}}$$

And the function, using $\frac{1}{2}$ as replacing value:

$$M_q(s_1, \ldots, s_m) = \left(\frac{1}{m}\left(\sum_{i\in\mathscr{C}}(s_k)^q + \sum_{i\notin\mathscr{C}}\left(\frac{1}{2}\right)^q\right)\right)^{\frac{1}{q}}$$

The only one difference is in the second term of the summation, that for $M_q(lo, up)$ can be expressed as:

$$\frac{\sum_{i\notin\mathscr{C}} 1}{2}$$

And, for the other expression, the term can be expressed as:

$$\frac{\sum_{i\notin\mathscr{C}} 1}{2^q}$$

So, only when $q = 1$ the two results are exactly the same.

Note that this method is only valid for $q \geq 0$. For $q < 0$, M_q suffers this transformation:

$$\Theta(s) = M_q(s_1, \ldots, s_m) = \left(\frac{1}{m}\sum_{k=1}^{m}(s_k)^q\right)^{\frac{1}{q}} = \left(\frac{m}{\sum_{k=1}^{m}\left(\frac{1}{s_k}\right)^p}\right)^p \text{, } q < 0 \text{ and } p = |q| > 0$$

When we calculate the lower bound we use 0^q, which in the case of a negative q for the aggregator would be $\frac{1}{0^p} = \infty$. Then, in the summation the other partial similarities doesn't matter because $\forall x_i = \left(\frac{1}{s_k}\right)^p, x_1 + x_2 + \cdots + \infty + \cdots + x_m = \infty$ and the last inversion $\left(\frac{m}{\infty}\right)^p = 0^p$, that is 0 for all $p > 0$.

So, these two last approaches try to represent the same concept through different ways. This last one is less general (not valid for $q < 0$) and also it is a little bit biased by the non-*missing value* partial similarities mean. Against it, the third approach is valid for any q and it is totally independent, because the value is not calculated.

You can see a camparative table of the different methods proposed for two different values: $q = 1$ (table 4.1) and $q = 2$ (table 4.2).

Table 4.1: Table comparing different replacing *missing value* functionalities. That is supposing 20 partial similarities and $q = 1$. The method 1) is Gower, 2) $s_i = 0$, 3) $s_i = \frac{1}{2}$ and 4) *interval.*

$s_{\mathscr{C}}$	Number of $\mathscr{X}$	[lo, up]	1	2	3	4
0.9	1	[0.855, 0.905]	0.9	0.855	0.88	0.88
0.9	19	[0.045, 0.995]	0.9	0.045	0.52	0.52
0.1	1	[0.095, 0.145]	0.1	0.095	0.12	0.12
0.1	19	[0.005, 0.955]	0.1	0.005	0.48	0.48

Table 4.2: Table comparing different replacing *missing value* functionalities. That is supposing 20 partial similarities and $q = 2$. The method 1) is Gower, 2) $s_i = 0$, 3) $s_i = \frac{1}{2}$ and 4) *interval.*

$s_{\mathscr{C}}$	Number of $\mathscr{X}$	[lo, up]	1	2	3	4
0.9	1	[0.877, 0.905]	0.9	0.877	0.884	0.891
0.9	19	[0.201, 0.995]	0.9	0.201	0.527	0.718
0.1	1	[0.097, 0.244]	0.1	0.097	0.148	0.186
0.1	19	[0.022, 0.975]	0.1	0.022	0.489	0.69

These two tables show the practical behaviour of these methods. The Gower proposal is less variable, the second one is too much pesimist and the others two are in the middle, but the proposal of the interval is not so impartial as we had said above theoretically.

Normalized aggregation

When we aggregate some partial similarities we are assuming that all of them have the same importance. We could give different importances to each partial similarity using a weighted aggregation but that is not the case.
But what is really happening is that we consider that the partial similarity are homogeneous and this is not true. Each partial similarity has its own mean in $[0,1]$. In this way, the partial similarities with a higher mean finish imposing themselves since they do a more important aportation to the aggregation. For example, for two variables distributed at $[0.2, 0.55]$ and $[0.7, 0.9]$, even the highest similarity of the first variable is less important than the lowest similarity of the second variable.

We propose to incorporate a functionality that gives similar importance to all the partial similarities using the same mean. In order to achieve that, the algorithm has to perform a pre-process where the partial similarity means are calculated. Then, these means will be used to normalize to the variables. But we only have the data set, so we calculate the sample mean comparing all the instances between them.

$$\overline{s_k} = s_{..k}$$

where $s_{..k}$ means that all the instances are comparated with all of them for the k^{th} variable.
This value is used to get all variables with mean at 1: $\hat{s}_{ijk} = \frac{s_{ijk}}{\overline{s_k}}$. Doing this, any $s_{ijk} < \overline{s_k}$ changes to $\hat{s}_{ijk} < 1$, any $s_{ijk} > \overline{s_k}$ changes to $\hat{s}_{ijk} > 1$, and only when $s_{ijk} = \overline{s_k}$, $\hat{s}_{ijk} = 1$.
Now, we have to translate the mean to $\frac{1}{2}$ and fit the function into $[0,1]$, because right now it goes from 0 to $+\infty$. We want to use the function $f(z) = \frac{1}{1+e^{-z}}$ because it has two horizontal asymptotes at $y = 0$ and $y = 1$, and it moves from $x = -\infty$ to $x = +\infty$ crossing the vertical axis at the point $(0, 0.5)$. Then, we only need a function that transforms the current interval $(0,+\infty)$ to the interval input of the f function $(-\infty,+\infty)$. That is, $g : (0,+\infty) \to (-\infty,+\infty)$ crossing the horizontal axis at $(1,0)$ to use 1 as the central value (remember that when $s_{ijk} = \overline{s_k}$, $\hat{s}_{ijk} = 1$). But, actually, the function f we have described is the logarithm, $g(x) = ln(x)$. So, we have:

$$f(z) = \frac{1}{1+e^{-z}} \text{ with } z = g(x) = a \cdot ln(x)$$

where a is a constant that allows us to reshape the normalization function (see the effect of changing it at figure 4.5).

If both functions (f and g) are combined we get:

$$f(x) = \frac{1}{1+e^{-a \cdot ln(x)}} = \frac{1}{1+\left(e^{ln(x)}\right)^{-a}}$$

$$f(x) = \frac{1}{1+x^{-a}} = \frac{x^a}{x^a+1}$$

And the final function $n : (0,+\infty) \to (0,1)$ we propose to transform the $\hat{s}_{ijk}$ is 4.16:

$$n(x) = \frac{x^a}{x^a+1} \tag{4.16}$$

where a is a factor that stablishes the curve of the function 4.5. We have chosen the value $a = ln(10) = 2.3 > 1$, because the black line (in the figure) is the one that works and adjusts the distribution of similarities better. With lower a values, the similarities are allways near to $\frac{1}{2}$ whereas, using a value as the one we propose, the similarities are more spread over the interval $(0,1)$.
This function has to be called every time that a similarity is calculated with $n(\hat{s}_{ijk}) = n\left(\frac{s_{ijk}}{\overline{s_k}}\right)$.

In this context, we defend the use of the third approach to replace *missing values*, using $\frac{1}{2}$. Using this normalized aggregator, the mean value or expected value for every variable is now $\frac{1}{2}$. In this way, if there are *missing values* in an aggregation, the most logical solution is to use the expected value to replace them and perform the aggregation. This simple solution is justified by the normalization:

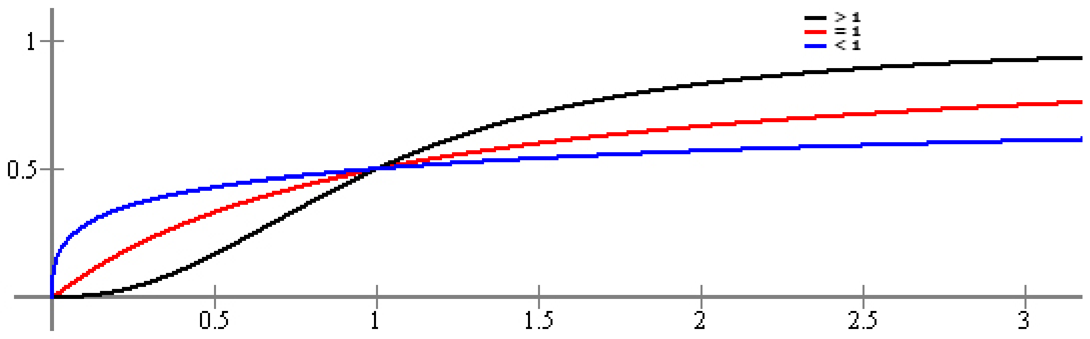

Figure 4.5: Graphical representation of the three a values for the normalizing function n.

When there is no *missing* information, $s_{ijk} \neq \mathscr{X}$, then the aggregator performs over the normalized partial similarities, $s'_{ijk} = n(\frac{s_{ijk}}{s_{..k}})$.

When there is *missing* information, $s_{ijk} = \mathscr{X}$, then $\frac{1}{2}$ is used as its normalized partial similarity $s'_{ijk} = \frac{1}{2}$. That is using $s_{..k}$ to replace the *missing value*, $s_{ijk} = s_{..k}$, and then apply the normalization $s'_{ijk} = n(\frac{s_{ijk}}{s_{..k}}) = n(\frac{s_{..k}}{s_{..k}}) = \frac{1}{2}$.

So, in the way we have defined the whole aggregator, using $s_{..k}$ (k-variable partial similarity mean) to replace s_{ijk} has the effect we expected over the non-*missing value* aggregation, moving it towards the central data model characterization.

Partial aggregations

But we have thought that this concept of aggregation can be used in other fields. Sometimes we could want to unify several partial similarities in an unique higher order partial similarity. That could be because some variables represent something and they have to be treated together and not independently as any other variable.
This conception generates several levels of aggregators and, for example, if the first level aggregator has l variables, the second and final level would have $m - l + 1$ variables (where the 1 of this formula is the result of the first level aggregator).

$$\Theta_i(\mathscr{O}_i) = M_q(s_{k \in \mathscr{O}_i})$$

$$\Theta(s) = M_q(\{s_{k \notin \mathscr{O}_i}, \forall i\} \cup \{\Theta_1(\mathscr{O}_1)\} \cup \cdots \cup \{\Theta_n(\mathscr{O}_n)\})$$

where $\mathscr{O}_i$ is the conjunt of partial similarities (variables) grouped by the i^{th} aggregator.

In these cases, the normalization would be not necessary because we are relating similar variables, so they have to be compatible.

There are two basic cases that we have taken into account since we decided to use several levels of partial aggregators. They are combinations of partial binary similarities into an unique measure. In this way, we are thinking in two similarity measures that are based on the four values of the table 4.3. Each letter represents the number of partial similarity measurements that compare two values following its associate combination of binary values (a: number of ++, b: +-, c: -+, d: –).

Table 4.3: Four possibilities of comparing two binary values.

	1	0
1	a	c
0	b	d

Following the table 4.3, the Jaccard similarity can be defined as:

$$s = \frac{a}{a+b+c} \tag{4.17}$$

This function, using our aggregator, can be set with $q = 1$, the binary partial similarity function "Binary" 4.3, and Gower proposal to replace *missing values*.
"Binary" only returns 1 when the two values are 1. Then it returns 0 when one value is 1 and the another is 0, and returns $\mathscr{X}$ if both are 0.
Gower doesn't take into account $\mathscr{X}$; his proposal is similar to replace *missing values* with the non-*missing values* mean. Then, performing the arithmetic mean (aggregator with $q = 1$) you get the Jaccard coefficient.

Following the table 4.3, the Simple Matching similarity can be defined as:

$$s = \frac{a+d}{a+b+c+d} \tag{4.18}$$

This function, using our aggregator, can be set with $q = 1$, the binary partial similarity function "Binary2" 4.4, and Gower proposal to replace *missing values*.
"Binary2" returns 1 when the two values are the same or returns 0 when they are different.
Performing the arithmetic mean (aggregator with $q = 1$) you get the Simple Matching coefficient.
Gower doesn't take into account $\mathscr{X}$ since his proposal is similar to replace *missing values* with the non-*missing values* mean. Here this functionality has no relevant importance because if there are $\mathscr{X}$, they come from the data collection and not from the partial similarity results, as in the previous method.

***RBF* version**

In order to implement the *RBF network* in a paralel way, we have to supply an aggregator specific for this version.
In the *RBF* networks, *missing values* and heterogeneous data are not allowed. Then, the aggregator is a simple function that implements an exponencial *RBF*:

$$\Theta(s) = e^{-\frac{\sum_{k=1}^{m}(s_{ik}-s_{jk})^2}{2\sigma^2}} \tag{4.19}$$

where $\sigma^2 = m$. This value was chosen because practical proves demonstrate that it can represent a good solution as heuristic for *RBF*'s width if you don't have any more information.

4.B.3 *Leader2*

The *Leader2* clustering algorithm was presented at [2]. It improves the *leader* algorithm, its Hartigan's classical version [3]. A in-depth study of the original algorithm and the similarity concept support the changes incorporated in the new version.

This algorithm is unsupervised, that is, it doesn't uses information about any target to fit the data. It gets instances, one by one, and asigns them to their most similar cluster. A cluster is represented by a *leader*, which is used to calculate the similarity of any instance with the cluster. The clusters are created dynamically when the similarity of an instance with any cluster is less than a minimum. This minimum is the only required parameter of the algorithm, the s_{min}, and it is the key that controls the number of clusters that the algorithm finally returns. Here, there exists a directly proportional relationship, a low s_{min} value implies a low number of clusters, and a high s_{min} generates a lot of clusters. s_{min} is a similarity, so it is defined also in $[0,1]$.
When an instance doesn't belong to any cluster, a functionality that searchs the best candidate to be the *leader* of a new cluster starts. Its only condition is that the instance that started this functionality has to belong to the new cluster. It searchs between the non-assigned instances a group of very-similar instances. Between them, the algorithm gets the instance that is more similar in average with all the other instances into the group, and this is the *leader* of a new cluster. Everytime a new cluster is created, previous assigned instances are re-evaluated. If some of these instances is more similar with the new cluster than with the cluster which it belongs to, this instance is re-assigned to the new cluster.
Finally we get a set of clusters that group instances attending to similarity measures. An instance is *leader* of a cluster or it belongs to its most-similar cluster. *Leader2* has a hard dependence on the input instances order that is followed to cover all the instances. That and the s_{min} determine the appearance of the solution returned by the *Leader2*.

The changes that we have incorporated into the algorithm during this project are basically implementation changes. The algorithm continues working in the same way.
The more important change was the incorporation of all the theory we discussed above. All the aggregators, similarity functions and *missing values* replacers are implemented to be used as heterogeneous similarity measure. The basic aggregator allows recursive calls, in order to perform several levels of aggregation.
When the normalization is used, a preprocess is called to calculated the partial similarity means. In order to improve the performance of the algorithm a $n \times n$ matrix is used to keep the similarity between all the instances. That is because we confirmed through practical proves that some similarity measures were done several times. In terms of resources we sacrifice some memory, that nowadays is cheaper and more common finding computers with big memory spaces, with the objective of gaining in processor time.
Other deep change is the new data structure used to represent the instances that belong to each cluster. In the one hand we have a n-length vector that keeps the similarity of each instance with its cluster. In the other hand, there is a h-length list that, for each cluster keeps a vector with the instances that belong to it. The characteristic that gives more relevance to this vector list is that the vector are sorted by similarity. The instances more similars with the leader are located in the top positions of the vector. The leaders naturally will be in the first position of each cluster vector. This new structure allows a more direct access to the less-similar instances of each cluster, which is useful for the reassignation of instances done everytime that a new cluster is created.

Leader2 supervised version

Following the idea implemented by Wettschereck and Dietterich [11], we have implemented an semi-supervised version of the *Leader2* algorithm. They defense the use of supervised clusterings to initialize the first layer reasoning that if the results of this algorithm will be used in a supervised method (the *Artificial Neural Network*), a supervised clustering would help to separate better the instances in the first layer of the *ANN*.
Before we said semi-supervised version because the only change we introduce with respect to the original version is that instead of calling the algorithm only once using all the instances, we call it x times, where x is the number of classes. Each time the algorithm is called, we give it the instances of a different class. Finally, once the algorithm has been called once per class, we aggregate the clusters returned by each call in a global solution.
For this reason we say that this is a supervised version, because we separate the instances by class and then we call the algorithm for each group of instances independently. In this way, two instances of different classes can not be in the same cluster, although they were the most similar couple of instances.

So, this version breaks some basic similarity principles of the algorithm [2]. The most data sets don't have naturally well separated classes, so some instances could have a more similar cluster than the one it belongs to. That is because there are two divions, the first one by class and the second one by similarity. Also, since the *Neural Network* is built, when the *HNN2* receives a new instance, it goes through the first layer without taking into account its class label and maybe the most excited neuron will not be the one which represents the cluster to which it would belong attending to this supervised version.

Leader2 distance version

We have also implemented a distance version of the *Leader2*. The way in which the algorithm works is the same, but it uses distance measures instead of similarity measures. Doing this we are limiting the scope of the algorithm because now this version only works with dissimilarity measures that fulfill the triangle inequality, that are a subset of the dissimilarity measures and so, a subset of the similarity measures.
But there is a great advantage: distance version can incorporate some improvements that outperform the algorithm (pruning the search tree and leaving earlier non-promising calculations). So, when a problem can be treated using only measures that fulfill the triangle inequality, it should be treated with this new distance version in order to take advantage of its performance.

First of all, the change between similarity and distance entails a complete change in the algorithm conception. Traditionally the algorithm is based on a measure that moves between 0 and 1, being maximum resemblance at 1. Now, the distance version preserves the interval, that is again $[0,1]$, but the maximum resemblance is now 0. The aggregator works in the same way, but similarity measures are now distance measures. The only parameter that the algorithm has is d_{max} in this new version (before s_{min}). Behind the two versions the idea is exactly the same, but changes with the function that relates distance and similarity: $distance = 1 - similarity$, such as we have defined distance and similarity concepts.

Once the algorithm has been rewritten using distance measures, some improvements have been implemented. The first one is based on the idea that Elkan proposed for K-means clustering algorithm [18].

The first idea is, taking advantage of the new sorted storage of the instances belonging to a cluster, to use the triangle inequality (4.20) to avoid unnecessary calculations [18] [19]. For example, a new cluster has been created and it is exploring the others looking for instances to reassign. When that is done, the checking starts with the farther instances of the cluster and goes towards the nearer ones. Then, if the distance of an instance with its *leader* is less than half of the distance between its *leader* and the *leader* of the new cluster, it can stop because no more instances will move between the two clusters.

$$d(a,b) \leq d(a,c) + d(c,b) \tag{4.20}$$

Using that we know that if $d(a,b) \geq 2 \cdot d(a,c)$, then $d(b,c) \geq d(a,c)$. So, having a instance a, its leader $l(a)$ and a new cluster represented by the leader $l(x)$ where $l(i)$ returns the *leader* of the instance i, if $\frac{d(l(a),l(x))}{2} \geq d(a,l(a))$, the comparations could stop because $d(a,l(x)) \geq d(a,l(a))$. The following instance e of the vector related to the cluster $l(a) = l(e)$ can not be moved because it is stored in a sorted vector such that $d(a,l(a)) \geq d(e,l(e))$. Following the triangular inequality, if a doesn't move, e neither.

The second idea is based on the same reasoning to avoid some calculations, trying to predict if it is impossible to reach the distance required when we want to assign an instance to a cluster. For this, it is necessary to know the distance between the different *leaders* and the distances between the instance to assign and the *leaders* checked so far. With this data some triangulations can be drawn and notice the algorithm if a way that it is checking is not promising.

We have rule out a third idea [20], which consists in avoiding the calculation of some partial distances if we are aggregating the partials results and we realize that this incomplete result is not so promising. In this case, the algorithm would leave the similarity calculation before all the partial similarities are calculated. We have decided not to implement it because it performs inside the aggregator, that is a functionality that we have left out of the basic algorithm in order to get the algorithm more abstract. In fact, users can define their own aggregator. There is another problem, working with heterogeneous data implies taking into account *missing values* and their behaviour in this case is not so clear. Studying the effect of *missing* information here requires an in-depth study that goes away from our research.

Initializing *RBF networks*

The *Leader2* clustering algorithm can be used to work as *RBF* network initializer selecting the radial basis aggregator (4.19). This is the only functionality that has to be changed in the algorithm along with the obligation of using allways continuous variables.
But also some preprocesses can be saved in this version, like the calculation of partial similarity means, which would accelerate the algorithm.

4.B.4 Neural Network structure

As we have already pointed out, the *Heterogeneous Neural Network 2*, our *ANN* model, is based on Belanche's work [9] [1]. Belanche's proposal is a sort of generalization of the *RBF* networks. The structure of the *HNN2* is also the same that in *HNN* or *RBF* networks, a two-layer feed-forward *ANN* (fig. 3.2).
The *HNN2*, as the *HNN*, is an *ANN* that deals with heterogeneous data and uses similarity measures to build the learning model. The *Leader2* clustering algorithm is used to initialize the network. The first layer is composed by h S-neurons, one per cluster and using its *leader* as center of the neuron. These neurons use as input heterogeneous data instances without preprocessing. According to this definition, S-neuron inputs are m-length vectors among which there might be reals, fuzzy sets, ordinals, categorical and *missing* data.

The neurons in the second layer perform a simpler function that collects all the outputs of the S-neurons multiplied by a weight and then mix them to return a combined output. Each one of the second layer output are also *HNN2* outputs, so if it is a regression or a two-classes classification problem, there will be an only one output neuron. Otherwise, there will be one output neuron per class.

But let's see the *HNN2* in detail.

S-neuron

The S-neuron (*similarity-based neuron*) is an H-neuron (*heterogeneous neuron*) that uses a similarity function to calculate the response of the neuron. Our proposal reconsiders the neuron model presented by Belanche [9] incorporating some changes.

In the original version, the S-neuron model has the following parts:

1. The *weight vector* (μ_i). It is usually an instance of the data set that is used as center of the neuron. It represents the point of maximum similarity for the neuron. The more similarity is an instance with the weight vector, the higher the response of the neuron.

2. Similarity measure. It is the aggregator that returns a global similarity measure for all the partial similarities. Belanche defines it in this way:

$$s(x,\mu_i) = \Theta(s_k(x_k,\mu_{ik})_{k=1,\dots,n}) = M_q(s_k(x_k,\mu_{ik})_{k=1,\dots,n})$$

   In this situation an input instance and the *weight vector* corresponding to that neuron are allways compared.

3. The *smoothing parameter* (γ_i). It is a parameter defined in the interval $(0,1]$ that has as main purpose to make similarity measures smooth.

4. The *activation function* (f), is any sigmoid-like automorphism in (0,1). It takes the similarity and adds a non-linear component to the aggregator result, that is linear.
   Belanche proposes the family 3.6, above presented. Here we reproduce it again:

$$f(x,p) = \begin{cases} \frac{-p}{(x-0.5)-a(p)} - a(p) & \text{if } x \leq 0.5 \\ \frac{-p}{(x-0.5)+a(p)} + a(p) + 1 & \text{if } x \geq 0.5 \end{cases}$$

$$a(p) = \frac{-0.5 + \sqrt{0.5^2 + 4 * p}}{2}$$

   where p is a real-valued parameter that controls the curvature of the function. Belanche proposes $p = 0.1$, that gives a central behaviour (see at figure 3.3).

All these parts are combined to make the function ($\Gamma(x)$) computed by the S-neuron. This would be the function computed by the i^{th} hidden neuron:

$$\begin{gathered} \Gamma_i(x) = f(\gamma_i * s(x,\mu_i), p), \; p = 0.1, q \in \mathbb{R}, \gamma_i \in (0,1] \\ \text{with } s(x,\mu_i) = \Theta(s_k(x_k,\mu_{ik})_{k=1,\dots,n}) = M_q(s_k(x_k,\mu_{ik})_{k=1,\dots,n}) \end{gathered} \tag{4.21}$$

Our proposal takes this function and changes some aspects in order to improve it and stops doing unnecessary calculations. Following the same division, we have implemented a new S-neuron version as:

1. The *weight vector* (μ_i), that fulfills the same role.
2. Similarity measure. It is the similarity aggregator that we defined above:

   $$s(x,\mu_i) = \Theta_t(s_k(x_k,\mu_{ik})_{k=1,\ldots,n})$$

   Since the comparison is allways done when an instance arrives at a neuron, the aggregator only performs comparison between arrived instances and the *weight vector* of the neuron.
3. The function of the *smoothing parameter*, that in Belanche's proposal was done by a new parameter (γ_i), now we propose that this functionality is performed by the p parameter. In the way it was defined by Belanche, it already has the capacity of changing the response of the neuron, that is directly connected with the similarity measure.
4. The *activation function* (f) is the same function family that Belanche proposes 3.6. It fullfils the properties we want for our neuron, returning values in the interval $(0,1)$ and adding a non-linear component to the aggregator result. In the same way, we also propose $p = 0.1$ by the same reason: the shape that draws the function $f(x,0.1)$ gives a central behaviour (fig. 3.3), but as we will explain below, it is not allways the same.

So, the definition of our S-neuron version can be synthesize in the next function:

$$\begin{aligned}&\Gamma_i(x) = f(s(x,\mu_i),p),\ p\in\mathbb{R}, q\in\mathbb{R}\\ &\text{with } s(x,\mu_i) = \Theta_t(s_k(x_k,\mu_{ik})_{k=1,\ldots,n})\end{aligned} \tag{4.22}$$

where p is better in $[0,3]$ (greater than 3 the function becames so linear), p also acquires the γ skills and Θ_t is the aggregator we talked about in the previous sections. Now, p can change, but the *HNN2* could use also different p values for each neuron. That is, using p_i instead of a global p.

The main changes between the two versions are the new aggregator and the use of p instead of γ. We remove the use of γ in the way Belanche points out because it has not an obvious porpose. He defines γ as a parameter that multiplies the result of the function f looking for smoothness. But, in the way the neuron model is defined, $f\in(0,1)$ and also $\gamma\in(0,1]$. Since we multiply two values which are between 0 and 1 ($a,b\in[0,1]$), we know that the result will be lower or equal than the minimum value a or b. That is, $a*b\leq\min(a,b)$. For this reason we know that the real effect of γ over the f result is that it reduces the f result proportionally (see at figure 4.6).

The solution that we propose really changes the function. Changing p you get a different function of the f-family where it is more linear the higher is p (see at figure 4.6).

Second layer neurons

The neuron model for the second layer doesn't change with respect to the *HNN* Belanche's proposal. It is a linear model that performs a summation of all the responses of the first-layer S-neurons ($\Gamma_i(x)$). These responses are weighted by a coefficient before the summation w_{ki}, where k refers to be the k^{th} output neuron and i refers to come from the i^{th} hidden neuron. Once the summation is done, some neuron models propose apply a function to transform the output. That is really necessary in some kinds of learning, like classification, where usually a logistic or heavyside function is used to return a discrete response $\{0,1\}$. But here we will not use any function and the response will be the weighted summation. In the figure 4.7 you can find a representation of this kind of neuron.

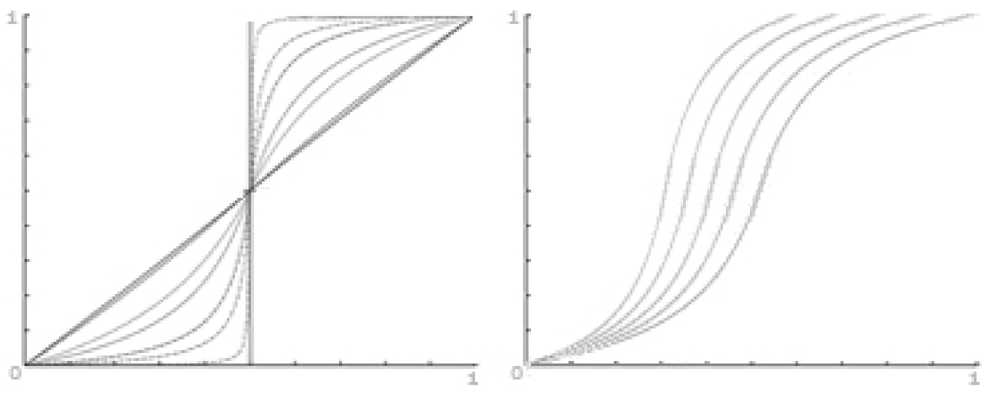

Figure 4.6: Effect of the parameters p and γ over the function f according to the two proposals. The first picture represents the effect of varying p and the second one represents the effect of varying γ for a fixed $p = 0.1$.

This decision is related with the inclusion of regularization, a method that controls the size of the weights in the second layer. We will explain it below.

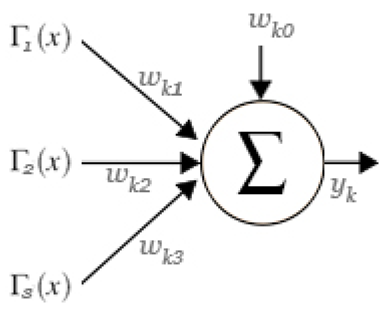

Figure 4.7: Representation of the neuron model used in the *HNN2* as output neuron.

Note that the summation incorporates an independent term w_{k0}, a constant coefficient that gives a linear component to make up for the non-linear responses of the first-layer S-neurons.

Attending to all these explanations, the k^{th} output neuron would calculate:

$$y_k = \sum_{i=1}^{h} w_{ki}\Gamma_i(x) + w_{k0} \tag{4.23}$$

where $\Gamma_i(x)$ is the function performed at the i^{th} hidden neuron and $k \in \{1 \ldots t\}$.

Training the *HNN2*

Since the *HNN2* is a two-layer feed-forward neural network, we can stablish exactly the steps to train it:

1. First layer weights.

2. p.

3. Second layer weights.

As we have pointed out before, the first layer weights are instances of the input data set. These instances are chosen by being the *leaders* (centroids) of the clusters returned by the *Leader2* clustering algorithm. This algorithm has been integrated into the *HNN2* and the similarity function it uses is the one we explain above. It can be completely customize, so using expert knowledge can be chosen the best similarity functions for each variable and the aggregator (q).

There are several ways to calculate the p value. This parameter is so similar with the width (σ) parameter of the *RBF* networks. Two of the principal ways to get it are to use cross validation or calculate it using an heuristic function. Both ideas are equally used in the literature. At [11], the authors defend the cross validation option because they don't trust the efectiveness of heuristic methods. Several other authors [8] [10] [12] prefer heuristic methods based on the information they have about the clusters. We have decided to use an heuristic method to calculate this value. We have included a new method we have designed and others taken from previous literature. We will see this in depth later and also a method called *Alternate Optimization* that optimizes this value using supervised information. In the same way that an *HNN* can use several γ values, one per S-neuron, or a *RBF* network can use several σ values, one per hidden neuron, our proposal, that uses p instead of γ or σ, can assign a different p value to each S-neuron (mP_i). This means that each hidden neuron will implement a different function of the f family. We have considered both possibilities and we have implemented a code that is able to perform in the two ways.

Once we have trained the first layer, we will train the output layer. And here, right now, there are only the weights to be trained.
The training could be different attending to the kind of learning process: regression or classification.

In regression problems you try to fit a function that has real numeric output. From this function you only have concrete examples (the data set instances) and the collected outputs that usually are affected by noise, so you don't really know the function. This kind of problems can be treated as a linear system of equations, where we have the output y (supervised learning), the different coefficients $Gamma_i$ (first layer responses: instances passed trough the S-neurons) and the variables, the weights w_{ki}. For each output neuron we would have $h+1$ variables, the weight associated with each S-neuron output and the constant coefficient, and n equations, one per instance. Depending on h and n, we would have an overdetermined system, if there are more instances than first layer neurons (the most common case), or an underdetermined system, if there are more hidden neurons than instances (problems with a low number of examples). This last case is only possible in one concrete situation: since the number of hidden neurons is lower or equal than the number of instances, $(h+1) > n$ only when $h = n$.

The method we have used to solve this system of equation is *least squares*. This method minimizes the sum of squared distances between the observed responses in the dataset and the responses predicted by the linear approximation. In order to achieve that, this method adjusts the second layer weights.

For the k^{th} output neuron, this is its output:

$$f_k(x_i) = \left(\sum_{j=0}^{h} \Gamma_j(x_i) w_{kj} \right) + \varepsilon_i$$

and the error that it generates would be calculate as:

$$SSE = \sum_{i=1}^{n} (y_{ki} - f_k(x_i))^2$$

This is the sum of squared error and it is the error which minimizes *least squares*.
Note that we suppose $\forall x, \Gamma_0(x) = 1$ in order to express the constant coefficient w_{k0} in the same way that the other weights of the output neurons.

We define the H matrix as:

$$H = \begin{bmatrix} \Gamma_0(x_1) & \Gamma_1(x_1) & \Gamma_2(x_1) & \dots & \Gamma_h(x_1) \\ \Gamma_0(x_2) & \Gamma_1(x_2) & \Gamma_2(x_2) & \dots & \Gamma_h(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Gamma_0(x_n) & \Gamma_1(x_n) & \Gamma_2(x_n) & \dots & \Gamma_h(x_n) \end{bmatrix} \tag{4.24}$$

That is a $nx(h+1)$ matrix.

Generalizing the first equation we get:

$$f_k(x) = Hw_k + \varepsilon$$

where w_k is a $(h+1)x1$ vector and y_k and ε are $nx1$ vectors. And the error can be also expressed in this way:

$$SSE = (y_k - Hw_k)^t (y_k - Hw_k)$$

And this expression, which we have to minimize, has an unique global minima:

$$w_k = \min_{w_k \in \mathbb{R}^{h+1}} SSE = \left(\frac{1}{n} \sum_{i=1}^{n} h_i h_i^t \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} h_i y_{ki} \right) = (H^t H)^{-1} H^t y_k \tag{4.25}$$

where $h_i \equiv [\Gamma_0(x_i), \Gamma_1(x_i), \Gamma_2(x_i), \dots, \Gamma_h(x_i)]^t$ and $A = H^t H$.

Doing that we can get the second layer weights and finish the training of a regression problem in our *HNN2*.

In classification problems you try to determine some groups, each instance belongs to a class and that is what you have to estimate. In order to solve this kind of problems, logistic regression can deal with two-class problems. For multi-class classification, the extended logistic regression is called multinomial logit m odeling. The only difference is that now we have a function that transforms the summation of the output neuron model. That is:

$$f_k(x_i) = g\left(\left(\sum_{j=0}^{h} \Gamma_j(x_i) w_{kj} \right) \right)$$

where $g(t) = \frac{1}{1+e-t}$. All the other parameters are the same and now the weights can be computed numerically by using *iteratively reweighted least squares*.

But we are not going to use that. Latter we will explain how we incorporate regularization into the calculation of second layer weights. This kind of regularization is called ridge regression

and it is only defined for linear regression methods.
Our solution, since getting a regularization method for classification problems doesn't fit in our job, is to convert classification into regression problems. That is done defining *m* class variables which have two possible values, 0 and 1. The value 1 appears in the i^{th} class variable of an instance if this instance belongs to the class *i* in the original class variable. Otherwise, the new variables have value 0. Note that, if there are only two classes only one variable is necessary, because the first class can be represented by 0 and the second one by 1.

RBF version

The version of *RBF* networks that we have gone implementing in a paralel way to the *HNN2* has the same structure. Both are two-layer feed-forward artificial neural networks. The two training processes only differ in some characteristics that define the *HNN2* first and, in other way, *RBF* networks.
In order to contrast the different version we use the same traning division than above to explain the hidden neuron model of *RBF* neurons:

1. The *weight vector* (μ_i), that fulfills the same role.
2. Similarity measure. Now, it uses the radial basis similarity aggregator 4.19. As we already now, the data is all continuous in order to allow the summation of the squared partial substractions. That is a distance that latter the exponential transform into similarity.
3. Here, in terms of an *smoothing parameter*, *RBF* needs its width parameter (σ). It has the capacity of changing the width of the radial basis implemented by the neuron. That means that achiving a high similarity comparing two instances will be easier because the gaussian function falls more slowly.
4. The *activation function* (f) is here the gaussian radial basis itself.

So, this is the definition of our version of *RBF* neuron model:

$$\Gamma_i(x) = f(x, \mu_i, \sigma) = e^{-\frac{\sum_{k=1}^{m}(x_k-\mu_{ik})^2}{2\sigma^2}}, \ \sigma \in \mathbb{R} \tag{4.26}$$

Note that it is defined for the i^{th} neuron.

The training process is exactly the same, with the *Leader2* clustering algorithm to get the first layer weights and linear regression to get the second layer ones. σ is calculated using heuristic measures, but that is also explained below. The use of different width parameters, one per hidden neuron, is also allowed.

4.B.5 Regularization

Regularization is a technique that incorporates additional information, usually of the form of a penalty for complexity, in the training of a model trying to prevent overfitting. Andre Tikhonow was who proposed this method, but he was trying to solve *ill-posed problems* where no unique solution exists. Specifically the kind of regularization that takes his name, also called *Ridge Regression*, is the one we have used in this thesis.
Ridge regression, applied to the *HNN2*, could be incorporated as some restrictions for smoothness bounding the growth of the second layer weights, for example. This is an idea that Orr [13] has developed for *RBF* networks. He defines ridge regression from the perspective of bias and

variance, and he uses that for rewriting all the equations that could have been affected by the penalty. In order to simulate the restrictions, Orr uses a parameter λ (regularization parameter) to stablish the scale of the penalty. A large λ forces to leave better fitted configurations if they require high weights.
This optimization is done through an iterative process, that re-estimates the λ and doesn't stop until the convergence is achieved. Orr defends using several initial values in order to get several final λ values after the iteration and choose the best one between them. That allows the algorithm to avoid local minima.

So, the *HNN2* is now defined in terms of the new equations. This is the new development, including the regularization parameter, for the k^{th} output neuron:

$$f_k(x_i) = \left(\sum_{j=0}^{h} \Gamma_j(x_i) w_{kj} \right) + \varepsilon_i$$

that doesn't vary, but the error includes a second summation:

$$SSE = \sum_{i=1}^{n} (y_{ki} - f_k(x_i))^2 + \lambda \sum_{j=0}^{h} w_{kj}^2$$

where the first term is the classical sum of squared error and the second one is the regularization term. Now, this is the equation that has to be minimized. The regularization parameter controls how much high weights are taken into account in the minimization equation. The minimization forces to compensate good results with low weights.

We define the H matrix in the same way that for linear least squares because it is not affected by the new penalty term. But, minimizing the generalized error equation we get this expression, which is the unique global minima:

$$w_k = \min_{w_k \in \mathbb{R}^{h+1}} SSE = (H^t H + \lambda Id_h)^{-1} H^t y_k \tag{4.27}$$

Now, the matrix A is defined as $A = H^t H + \lambda Id_h$, when it was only $A = H^t H$ so far.

We use a criteria (*Generalized Cross Validation, GCV*) defined by Orr in order to stablish a limit for convergence. It is related with an adjustment to the *SSE* over the training set. If this limit is not reached, we can use a maximum number of iterations to force the break of the loop. This criteria is:

$$c_{GCV} = \frac{n y^t A^{-1} y}{(tr(P))^2} \tag{4.28}$$

Note that this expression use A and P matrices, which means that it needs a λ value to be calculated. This criteria is used in this way:

$$\frac{\bar{c}_{GCV}}{|c_{GCV} - \bar{c}_{GCV}|} > \frac{1}{\varepsilon}$$

where $\bar{c}_{GCV}$ represents the value of c_{GCV} in the previous iteration and $\varepsilon = \sqrt[4]{\text{Machine:epsilon}}$. The *Machine Epsilon* gives an upper bound on the possible relative error that generates the forced rounding in floating point arithmetic when there is no space for more digits in the machine. It can be calculated as the smallest positive floating-point number x such that $1 + x \neq 1$. It was chosen by being the most common epsilon used in R, whereas the root was introduced to use a similar value to the one used by Orr in his calculations (0.001).

This stop condition says when a regularization parameter can not be optimized anymore. That happens when the criteria value doesn't change or its change is negligible.

Finally, we adjust the values of the parameters using a formula derived from this reasoning for re-estimating lambda:

$$\lambda = \frac{y^t A^{-1} y * tr\left(A^{-1} - \lambda A^{-2}\right)}{w^t A^{-1} w * tr\left(P\right)} \tag{4.29}$$

As you can see, it uses the λ previous value to recalculate the new one since λ involves the creation of some matrices. So this formula has to be applied once these matrices (weights, P and A) have been calculated using some previous λ value. In the first step, previous λ value is the initial value.
The function $tr(X)$ calcules the trace of an $n \times n$ square matrix, that is summation of the coefficients in the main diagonal of the X matrix:

$$tr(A_{n\times n}) = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{i=1}^{n} a_{ii}$$

SVD version

Orr defined some time later an efficient re-estimation of λ based on the Singular Value Decomposition of the matrix HH^t [14]. Using this decomposition all the formulas used in the regularization method can be rewritten.

The *SVD* factorizes an $n \times m$ matrix (M) of the form:

$$M = U\Sigma V^*$$

where U is an $n \times n$ unitary matrix ($U^*U = UU^* = Id_n$), the matrix Σ is an $n \times m$ diagonal matrix of non-negative real numbers, and V^* is the conjugate transpose of V, an $m \times m$ unitary matrix.
So, if we done this decomposition for the matrix H, Orr points out that the U columns are the eigenvectors ($u_{i}{}_{i=1}^{n}$) and the squared diagonal elements of Σ are the eigenvalues ($\mu_{i}{}_{i=1}^{n}$) of the HH^t matrix.

The function $tr(X)$ can be also defined in terms of *SVD*. The trace of a matrix is also the summation of its eigenvalues: $tr(A_{l\times l}) = \sum_{i=1}^{l} \mu_i$

Using the previous matrices, Orr rewrites all the equations replacing the old matrices by some coefficients. These coefficients can be calculated in an iterative process, avoiding the matrix calculations:

$$n - \gamma = \sum_{i=1}^{n} \frac{\lambda}{\mu_i + \lambda} \tag{4.30}$$

$$\eta = tr\left(A^{-1} - \lambda A^{-2}\right) = \sum_{i=1}^{n} \frac{\mu_i}{(\mu_i + \lambda)^2} \tag{4.31}$$

$$w^t A^{-1} w = \sum_{i=1}^{n} \frac{\mu_i \tilde{y}_i^2}{(\mu_i + \lambda)^3} \tag{4.32}$$

$$e^t e = y^t A^{-1} y = \sum_{i=1}^{n} \frac{\lambda^2 \tilde{y}_i^2}{(\mu_i + \lambda)^2} \tag{4.33}$$

where $\tilde{y}_i = y^t u_i$ is the projections of y onto the eigenvectors.

In this way, the *GCV* criteria can be rewrite as:

$$c_{GCV} = \frac{ne^t e}{(n-\gamma)^2} \tag{4.34}$$

But its use remains the same:

$$\frac{\bar{c}_{GCV}}{|c_{GCV} - \bar{c}_{GCV}|} > \frac{1}{\varepsilon}$$

And the formula used to recalculate λ is now:

$$\lambda = \frac{\eta e^t e}{(n-\gamma) w^t A^{-1} w} \tag{4.35}$$

So, there are two methods to perform the regularization over the second layer weights. Both have been implemented and adapted to be used by any version of neural network (*HNN2*, *RBF* networks...) developed in this thesis because all the methods share this code.

4.B.6 Standard deviation (p/σ)

The *HNN2* learning algorithm is a two-layer neural network. It has several parameters, so we have designed several functionalities to initialize them and save users from choosing a value for them.

One of these parameters is the one that controls the shape of the function that is evaluated in the first-layer neurons. In *HNN2*, this parameter is p and controls the step of the S-neuron function, its smoothness. In *RBF* networks, this parameter sets the width of the gaussian radial basis, σ.

In the literature, this have been a highly dealt problem for *RBF* networks. There are several ways to cope with it, but we have chosen the estimation in order to present an easily configurable system. In this way, we have had to propose an heuristic method that estimates a value for p because, since *HNN2* is a new learning algorithm, there are no previous proposals in the literature to apply here.
In this way, we have proposed an heuristic based on the concept of *compactness*. We associate the *compactness* of a cluster with a harder step of the f function (a lower p). When a cluster is more compact (there is a high number of examples with a high similarity with the leader), it is easier to decide whether a new example belongs to that cluster or not because the cluster is well-defined and the limits are clear. Using a family of functions like the f family, this behaviour is associated to a p that works similar to a heavyside function (p$\rightarrow$ 0). Following this reasoning, we have defined a *compactness index* B.1, that is a value $ic \in (0,1)$ that is closer to 1 the more compact is the cluster. Then, we use a function r to transform this coefficient into the p value.
In this way, this heuristic returns a different p for each cluster or hidden neuron. It has been defined as (B.2):

$$p_j = r(ic_j) = -\ln(ic_j) \tag{4.36}$$

Using the following formula to calculate the *compactness index*:

$$ic_j = \frac{m_j l_j}{m_j l_j + (\exp(0.1) - 1)\overline{ml}} \tag{4.37}$$

where m_j and l_j are the averaged similarity and the number of instances in the cluster j, and $\overline{m}$ and $\bar{l}$ are global averages (for the whole clustering).

If you want to see the reasoning we have followed to arrive to this proposal, you can find it in the annex B.

Heuristic measures to calculate σ

A similar parameter exists in *RBF* networks, σ. We don't have any proposal in this case, but we have collected some proposals in the literature. These are the methods we have incorporated:

The first method is a Bishop's proposal [8]. It can return a different σ for each cluster:

$$\sigma_j = l * mean\left(||mu_i - \mu_j||_{\forall i \in k-NN_j}\right) \tag{4.38}$$

where l is some number that multiplies the average (usually, $l = 2$), and $k - NN_j$ is the set of the k centers nearest neighbors ($k = \frac{h}{10}$) to the center of the j^{th} hidden neuron. In our case, the centers are the leaders of the clusters.

Another method design by Bishop is the following one, that returns an only global σ:

$$\sigma^2 = l * mean\left(||mu_i - \mu_j||_{\forall i,j \in \{1,\ldots,h\}}\right) \tag{4.39}$$

where l is some number that multiplies the average (usually, $l = 2$).

We have collected also an Haykin's proposal [10] that calculates also a global measure unique for all the neurons:

$$\sigma = \frac{\max_{i,j \in 1,\ldots,h} ||mu_i - \mu_j||}{\sqrt{2h}} \tag{4.40}$$

where l is some number that multiplies the average (usually, $l = 2$).
As you can see, it is so similar to the Bishop previous proposal.

In [11], the authors present a method that uses also the *nearest neighbors* concept to calculate one σ per neuron:

$$\sigma_j = \frac{1}{||mu_i - \mu_j||_{\forall i \in k-NN_j}} \tag{4.41}$$

where $k - NN_j$ is the set of the k centers nearest neighbors ($k = \frac{h}{10}$) to the center of the j^{th} hidden neuron.

Also Benoudjit [21] proposed his own method, based again on the *nearest neighbors* concept, which returns several σ_j:

$$\sigma_j = \frac{\sqrt{\sum_{i \in k-NN_j} ||mu_i - \mu_j||}}{k} \tag{4.42}$$

where $k - NN_j$ is the set of the k centers nearest neighbors ($k = \frac{h}{10}$) to the center of the j^{th} hidden neuron.

The two last proposals are reasoned out to apply known functionalities in the *HNN2*.
The first one is based on the concept of centroid. The main problem is that the reasoning followed to formulate this method maybe can not fit in *HNN2* because it uses centroids, that are an artificial exact center, and we use leaders, an instance of the input data set that represents a cluster but it is not exactly the center. In any way, the proposal is the next one:

$$\sigma_j^2 = \frac{1}{||[\mu_j]|| - 1} \sum_{x \in [\mu_j]} ||x - \mu_j|| \tag{4.43}$$

where $[\mu_i]$ is the set of instances in the cluster of the leader μ_i.

The second proposal is based on the concept of *ball*. A ball is defined in the following way:

$$B(a,r) = \{x | d(x,a) < r\}$$

If we transform this definition to the *HNN2* characteristics we get:

$$B(\mu_i, s_{min}) = \{x | s(x,\mu_i) \geq s_{min}\}$$

All this is in the context of *RBF* networks, so:

$$s(x,\mu_i) = \exp - \frac{||x-\mu_i||^2}{\sigma_i^2}$$

And then:

$$B(\mu_i, s_{min}) \rightarrow \exp - \frac{||x-\mu_i||}{\sigma_i^2} \geq s_{min}$$

And developing the reasoning:

$$-||x-\mu_i||^2 \geq \sigma_i^2 \ln s_{min}$$

$$\sigma_i^2 \geq - \frac{||x-\mu_i||^2}{\ln s_{min}}$$

$$\sigma_i \geq ||x-\mu_i|| \left(\ln \left(\frac{1}{s_{min}} \right) \right)^{-\frac{1}{2}}$$

$$\sigma_i \geq \max_{x \in [\mu_i]} ||x-\mu_j|| \left(\ln \left(\frac{1}{s_{min}} \right) \right)^{-\frac{1}{2}}$$

Finally, we can define a method based on this reasoning where the objective is to force this neuron to return the higher response when an instance of the cluster that represents this neuron arrives. Then, we return several σ_j:

$$\sigma_j^2 = l * \max_{x \in [\mu_j]} ||x-\mu_j|| \left(\ln \left(\frac{1}{s_{min}} \right) \right)^{-\frac{1}{2}} \tag{4.44}$$

where $[\mu_j]$ is the set of instances in the cluster of the leader μ_j and $l \geq 1$ is some number that can increase the width of the exponential (we use $l = 1$).

All these are deterministic proposals that define a σ or several σ_j usually based on some clustering characteristics which are used in a more or less justified way.
We have thought about that and we have wanted to incorporate some functionality that compensates the possible lack of correctness of the σ, that is the *Alternate Optimization*.

Alternate Optimization

The *Alternate Optimization* (*AO*) is a functionality we have incorporated to the method that combines two different optimizations. The first one, that has been explained above, is the weights regularization. In some way, this process could be seen as an optimization that achieves the best possible weights attending to some restrictions. The second optimization we want to introduce is the p/σ optimization that uses the *HNN2* results to improve the p/σ parameters calculated with

heuristic methods.
The problem here is that the weights optimization uses the p/σ parameter and vice versa, so we have proposed a method that alternates both optimizations in a sequential way. Then, an optimized parameter is used to the optimization of the other one. Changing a parameter implies changing the results, so once a parameter has been optimized, the other parameter needs to be optimized too. This implication never ends, so both optimizations should be alternated until a convergence situation were achieved. If no convergence is reached, we restric the maximum number of *AO* possible iterations.

In any way, the weights optimization or regularization has been already defined. Now, we present the p optimization. σ optimization performs the same but using the specific functions of *RBF* networks.

The f functions family 3.6 of the S-neurons is:

$$f(x,p) = \begin{cases} \frac{-p}{(x-0.5)-a(p)} - a(p) & \text{if } x \leq 0.5 \\ \frac{-p}{(x-0.5)+a(p)} + a(p) + 1 & \text{if } x \geq 0.5 \end{cases}$$

$$a(p) = \frac{-0.5 + \sqrt{0.5^2 + 4 * p}}{2}$$

And we have calculated its partial derivative with respect to p, which is used to estimate the better p value:

$$\frac{\partial f}{\partial p} = f'_p(x,p) = \begin{cases} \frac{-(x-0.5)+a(p)-p*a'(p)}{((x-0.5)-a(p))^2} - a'(p) & \text{if } x \leq 0.5 \\ \frac{-(x-0.5)-a(p)+p*a'(p)}{((x-0.5)+a(p))^2} + a'(p) & \text{if } x \geq 0.5 \end{cases}$$

$$a(p) = \frac{-0.5 + \sqrt{0.5^2 + 4 * p}}{2}$$

$$\frac{da}{dp} = a'(p) = \frac{1}{2a(p) + \frac{1}{2}} \tag{4.45}$$

In our implementation, we have taken advantage of the R-project possibilities and we have used the "optimize" function when there is an unique p value and "optim" when there are several.

This functionality is exactly the same but using the specific functions of gaussian *RBF*'s and optimizing the value of σ, the width parameter. That is:

$$\varphi(x,\sigma) = \exp(-\frac{\|x-\mu\|^2}{2\sigma^2})$$

Which is the function calculated in each neuron, and its partial similarity with respect to σ is:

$$\frac{\partial \varphi}{\partial \sigma} = \varphi'_\sigma(x,\sigma) = \frac{x}{\sigma^3} \exp(-\frac{\|x-\mu\|^2}{2\sigma^2}) \tag{4.46}$$

In this case, the error that minimizes this optimization is the following for *HNN2*:

$$\frac{\partial SSE}{\partial p_j} = \sum_{i=1}^{n} \sum_{k=1}^{h} (\hat{y}_{ki} - y_{ki}) w_{jk} f'_{p_j}(x, p_j)$$

where $\hat{y}_k = Hw_k$.
And for gaussian *RBF*'s we have the following adapted error function:

$$\frac{\partial SSE}{\partial \sigma_j} = \sum_{i=1}^{n} \sum_{k=1}^{h} (\hat{y}_{ki} - y_{ki})\, w_{jk} {\varphi'}_{\sigma_j}(x, \sigma_j)$$

This minimization is the second step of the *Alternate Optimization*. This and the regularization are the re-estimation methods that we incorporate to improve the solution the built parameterization.

4.B.7 Final version

So far we have explained the different functionalities we have implemented in *HNN2*. But now it is necessary to configure the method to get a default version that only requires s_{min} to perform. Some of the functionalities we have implemented require choosing a behaviour and we have chosen default values for them allowing users to use the algorithm in an easy way.

Let's start with the *Leader2* clustering algorithm. From its previous version it maintains the calculation of the L parameter. This says which is the number of instances that have to be grouped before choosing the leader. So, this parameter is calculated by default as: $L = \lfloor n^{\frac{2-s_{min}}{4}} \rfloor$.
In the new version, all the functionalities of the aggregator can be adjusted. The aggregator itself can be chosen, so the default aggregator is the one we have design: "aggregatePartials" (4.B.2). In this aggregator we have to choose some behaviours. In first term, the normalization is activated by default, it performs the arithmetic mean since it uses $q = 1$ and the method to replace the *missing* partial similarity measurements is "fill.zerofive" by default, that is the approach that uses $\frac{1}{2}$ to replace *missing values*.
Regarding partial similarity measures and data types, we have set by default six data types with their corresponding similarity method: binary variables with "binary" (4.3), categoric variables with "overlap" (4.5), ordinal variables with "probabilistic" (4.7), continuous variables with "continuous" (4.8), discrete variables with "continuous" (4.8) and fuzzy variables with "fuzzy.zerofive" (4.13). Some of these similarity functions require parameters to calculate the similarity between two values. These parameters are calculated, if it is possible, during a preprocess attending to the whole data set. For example, the range for "continuous" is calculated as the substraction of the maximum less the minimum value of the variable over the whole data set.

The *HNN2* itself has its own configuration, and it requires also a default setting. The ε value used finding errors or stablishing convergences is here the fourth root of the *machine epsilon*.

In the first layer, the activation function used by default is "functFP", the one we defined above (3.6). It is calculated for each S-neuron independently using the "findP" function, the method we have design to do that (B.2).
Attending to the regularization method, we can choose between the method based on matrix calculations or the one based on the *Singular Value Decomposition*. The first approach is set by default. To perform, it uses three different initial λ, that we have set to $\lambda_i \in \{10^{-6}, 10^{-3}, 1\}$ attending to Orr's work [13]. If convergence is not reached before, we stablish a maximum number of 100 iterations for the regularization loop.
The second loop, which performs the p/σ optimization (*Alternate Optimization*) is set to a maximum of 10 iterations. According to the use of p parameter and the previous activation function, we use the method to recalculate p defined in the previous section (4.B.6). But all this has not importance if the user doesn't activate the *AO* because it is off by default.

That is the configuration for the *HNN2* we have stablished by default.

Some little changes in the previous configuration have to be introduced to perform as the *RBF* network version we have implemented. The first change is to use the aggregator based on the gaussian function for the *Leader2* clustering algorithm. Also, the gaussian radial basis function is used as activation function. Now the algorithm uses σ parameter, so we have set the first heuristic method proposed by Bishop [8] to calculate its value (4.38). As before, we use several σ_i, one per hidden neuron. In the same way, the *AO* uses the method that recalculates σ.

To evaluate both methods we have used a prediction function that calculates the prevision of the learning algorithm. Prevision (Y) and real expected output (T) are compared to return the error measures. Basically, for regression methods it returns two error measures that point out how good the algorithm is for an specific data set. These two errors are the mean squared error:

$$MSE = \frac{1}{t}\sum_{t}\frac{1}{n}\sum_{n}(Y-T)^2 \tag{4.47}$$

and the normalized root mean squared error:

$$NRMSE = \sqrt{\frac{\frac{1}{t}\sum_{k=1}^{t}\frac{1}{n}\sum_{i=1}^{n}\left(Y_{ki}-T_{ki}\right)^2}{\frac{1}{t}\sum_{k=1}^{t}\frac{1}{n}\sum_{i=1}^{n}\left(T_{ki}-\overline{T_k}\right)^2}} \tag{4.48}$$

$$\overline{T} = \frac{1}{n}\sum_{i=1}^{n}T_i$$

The second measure returns a global comparable measure since it is normalized. Classification problems returns also a third measure that represents the accuracy of the method, how many instances have been correctly classified.

5

Experimental results

Once we had finished the development of *HNN2*, we started the experimental settings. These test sets were designed in order to perform a complete practical study of the algorithm. We know its asset, but we have to demonstrate its good performance in a practical way.
In order to achieve that, we have evaluated 10 problems (see their descriptions at appendix C).

The testing was separated in two stages. The first one, the *Fractional Factorial Design*, is used to choose the better values to set some default choises in the configuration of the algorithm. That is, we optimize the default performance of the *HNN2* by choosing the configuration that reports the best results in average over some problems. The second part is a design that compares our *HNN2* with some other algorithms and it is optimized to use two statistical tests that confirm the difference between methods.

Note that here we don't talk about the tests done to verify the right performance of the different functionalities of the algorithm and the final tuned up of the whole *HNN2* because there is no enough space and they are no relevant.

Let's see the experimental settings.

5.A Fractional Factorial Design

In order to choose an standard configuration for this learning method we have developed a Fractional Factorial Design (*FFD*) [22]. There are several reasons that we thought to choose this specific experimental design. The most important one is that we had 5 principal binary questions to solve and combinating them would generate $2^5 = 32$ tests. But, in order to get a more realistic idea, these tests would have to be executed several times. Using a *FFD* this number can be significantly reduced.

Developing this concept, it is necessary to choose carefully a subset (fraction) of the experimental runs (tests) of a full factorial design. This full design performs a test for all the possible combinations of the factors (2^5 tests). The subset for the fractional design is chosen attending to the most important features (factors) of the studied problem. We have 5 factors, so we choose the 2^{5-2}_{III} design specification, which requires 8 runs to work, which is a quarter of the runs required by the full factorial design.
The questions we raised to perform this experimental design are defined as:

1. *Cont.* Similarity measure for continuous variables. We raise the question of using a continuous traditional measure to calculate the similarity of continuous variables or using a fuzzy measure.
   The similarity measure used as continuous measure is the one we called *continuous* (4.8).

The similarity measure used as fuzzy measure is the one we called *fuzzy.zerofive* (4.13). The negative/positive value of the factor are expressed in this way:

$$X_1 \equiv A \equiv \begin{cases} + : \textit{Continuous} \\ - : \textit{Fuzzy} \end{cases}$$

2. *Cat.* Similarity measure for categoric variables. The question now is the choice between two different similarity measures for categoric variables: *overlap* (4.5) or *frequency* (4.6). The negative/positive value of the factor are expressed in this way:

$$X_2 \equiv B \equiv \begin{cases} + : \textit{Overlap} \\ - : \textit{Frequency} \end{cases}$$

3. *Norm.* Normalization. It is an obvious binary question, have we to use the normalization or not?
The negative/positive value of the factor are expressed in this way:

$$X_3 \equiv C \equiv \begin{cases} + : \text{Yes} \\ - : \text{No} \end{cases}$$

4. *Uniq.* Unique smoothness parameter for all the neurons at the first layer. The σ/p parameter can be a single parameter, shared by all the hidden neurons, or a set of parameters, where each hidden neuron has its own parameter. The question, then, is which of the two modes works better.
The negative/positive value of the factor are expressed in this way:

$$X_4 \equiv D \equiv \begin{cases} + : \text{A parameter per hidden neuron.} \\ - : \text{An unique shared parameter.} \end{cases}$$

5. *AO*. *Alternate Optimization*. Is better to use it or not?
The negative/positive value of the factor are expressed in this way:

$$X_5 \equiv E \equiv \begin{cases} + : \text{Yes} \\ - : \text{No} \end{cases}$$

where the order matters, the number of each factor indicates their position inside the confounding structure (table 5.1). This table relates the factors and allows us to reduce the number of runs. Choosing the right order of the factors we have taken into account the following relations:
The normalization (*Norm.*) and the similarity measures (*Cont.* and *Cat.*) perform their main task into the clustering algorithm that initializes the *HNN2*. The (single or many) σ/p parameter is used in the first layer of the *HNN2*, once the centers for hidden neurons have been chosen (the clustering algorithm is finished). Finally, the *AO* "optimizes" the configuration of the network. With this information, we can ensure that the two last factors, in the way they are used in the last stages of the learning process, are affected by the three firsts ones.

Following the 2^{5-2}_{III} design specification and attending to our factor ordering choice, the 8 runs we have to perform are shown at table 5.2. This division assures all the factors are evaluated, but in order to get more realistic results we have to do this 8-tests set several times. Each run performs a 3-fold Cross Validation over the training set (leaving out a test set with the 33% of instances) for choosing the free parameter: s_{min} in *HNN2* and *RBF2*, k in *RBF + K-means*, and

Table 5.1: Map of Factors and Dependences.

FACTOR	DEFINITION	CONFOUNDING STRUCTURE
1	1	1 + 24 + 35 + 12345
2	2	2 + 14 + 345 + 1235
3	3	3 + 15 + 245 + 1234
4	12	4 + 12 + 235 + 1345
5	13	5 + 13 + 234 + 1245

cost in *SVM*. Between 10 and 13 different possible free parameters are taken into account and only the value that returns better results is chosen to get the final results over the test set.
We have performed 3 repetitions of the 8-tests set changing the folds of Cross Validation. All this process has been done using four different data sets: *CRX*, *BANDS*, *HC22* and *HC23*. We chose these four because all of them have many instances, heterogeneous data and *missing values*, that is what we want to fit.

Table 5.2: Map of configurations.

Configuration	X_1	X_2	X_3	X_4	X_5
	Cont.	Cat.	Norm.	Uniq.	*AO*
1	+	+	+	+	+
2	+	+	-	+	-
3	+	-	+	-	+
4	+	-	-	-	-
5	-	+	+	-	-
6	-	+	-	-	+
7	-	-	+	+	-
8	-	-	-	+	+

As we said above, each run performs a *3-fold CV* and the best value is used to get the final result from the test set. Then, when we are evaluating the *FFD*, we get these final results and combine them in the following way:

$$R_{X_i} = \frac{1}{|C_{X_i}|} \sum_{j \in C_{X_i}} c_j - \frac{1}{|\not{C}_{X_i}|} \sum_{j \notin C_{X_i}} c_j \tag{5.1}$$

where C_{X_i} is the set of indexes which groups the configurations (runs) that perform with the positive value of the X_i factor. In this way, the configurations in $\not{C}_{X_i}$ perform the same with the negative value of the X_i factor.
Doing that we get a final value R_{X_i}, that represents the valoration of the factors. Depending on the measure used to do these calculation (error, accuracy...), the positive and the negative value have different meaning.

Applied to our method, these are the equations that evaluate the 5 factors:

$$
\begin{aligned}
R_{X_1} = R_{cont} &= \frac{1}{4}(c_1+c_2+c_3+c_4) - \frac{1}{4}(c_5+c_6+c_7+c_8) \\
R_{X_2} = R_{cat} &= \frac{1}{4}(c_1+c_2+c_5+c_6) - \frac{1}{4}(c_3+c_4+c_7+c_8) \\
R_{X_3} = R_{norm} &= \frac{1}{4}(c_1+c_3+c_5+c_7) - \frac{1}{4}(c_2+c_4+c_6+c_8) \\
R_{X_4} = R_{uniq} &= \frac{1}{4}(c_1+c_2+c_7+c_8) - \frac{1}{4}(c_3+c_4+c_5+c_6) \\
R_{X_5} = R_{AO} &= \frac{1}{4}(c_1+c_3+c_6+c_8) - \frac{1}{4}(c_2+c_4+c_5+c_7)
\end{aligned}
$$

Our prediction method returns: quadractic error, normalized error and accuracy (when there is a classification problem). The last reasoning applied to error or accuracy has different interpretations.
In the one hand, when we want to calculate the configuration using accuracy, we are in front of a maximizing problem. We want the maximum accuracy for our method, so the higher value the better. The function 5.1 is only a substraction of two arithmetic means, the one of positive values of X_i and the one of negative values. The arithmetic mean doesn't change the properties of the percentages, so R_{X_i} is the substraction of the mean accuracy for positive values and the mean for negative values. So, $R_{X_i} < 0$ means that the mean accuracy for negative values of X_i factor is higher than the mean for positive values. Since we are maximizing, the choice is, obviously, the negative value. If $R_{X_i} > 0$ (the mean accuracy for positive values of X_i factor is higher than the mean for negative values) the choice would be the positive value.
On the other hand, but following a similar reasoning, when we want to calculate the configuration using error, we are in front of a minimizing problem. We want the minimum error for our method, so the lower value the better. Now, $R_{X_i} < 0$ means that the mean error for negative values of X_i factor is higher than the mean for positive values. Since we are minimizing, the choice is now the positive value. Otherwise, if $R_{X_i} > 0$ (the mean error for positive values of X_i factor is higher than the mean for negative values) the choice would be the negative value.

5.A.1 *FFD* Experimental Results

The results of this experimental setting gave us the default configuration of the *HNN2* algorithm.

We have 8 configurations and 4 problems, if each problem returns 3 coefficients, we get 96 final values in one repetition. In the table 5.3 we have all of them, but they are the average of the 3 repetitions.

As you can see, the results move several percentage points between the best configurations and the worst ones. A similar behaviour, but in other scale happens with the errors. The extreme cases are the *HC23*, where the difference of accuracy arrives almost until 6%, against *CRX*, where the maximum difference doesn't pass 2%. In terms of configurations, the configuration number 4 and the number 5 stand out because are the best ones for *HC22* and *CRX* (configuration 5), and *HC23* and *BANDS* (configuration 4). If we average the results of the different problems we get the table 5.4.

In this averaged table, the percentage differences vary 3.2%. Configuration number 4 have good results, but in second place now we have configurations 3, 5 and 7 also with good results.

Table 5.3: Averaged response by data set and configuration. There are 3 measures by data set, the percentage of accuracy (*Acc.*), the quadratic error (*QE*) and the normalized error (*NE*).

Name	Measure	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7	Conf. 8
HC22	*Acc*	67.662	68.159	71.144	71.642	72.637	71.144	71.642	68.159
	QE	0.144	0.141	0.141	0.138	0.138	0.135	0.134	0.143
	NE	0.914	0.905	0.904	0.893	0.894	0.885	0.884	0.909
HC23	*Acc*	83.824	84.804	84.314	86.765	82.352	81.863	85.784	80.882
	QE	0.127	0.122	0.118	0.121	0.125	0.13	0.118	0.133
	NE	0.732	0.719	0.707	0.715	0.727	0.742	0.706	0.749
CRX	*Acc*	86.377	86.232	86.957	86.957	87.971	87.246	87.681	87.681
	QE	0.103	0.1	0.1	0.106	0.096	0.101	0.097	0.103
	NE	0.646	0.634	0.635	0.654	0.624	0.637	0.627	0.645
BANDS	*Acc*	70.185	73.148	73.704	75.741	73.333	72.407	71.667	72.037
	QE	0.2	0.187	0.180	0.169	0.184	0.185	0.185	0.187
	NE	0.905	0.874	0.858	0.833	0.867	0.87	0.871	0.876

Table 5.4: General averaged response.

Name	Coef.1	Coef.2	Coef.3	Coef.4	Coef.5	Coef.6	Coef.7	Coef.8
Acc.	77.012	78.086	79.03	80.276	79.074	78.165	79.193	77.19
QE	0.144	0.138	0.135	0.133	0.136	0.138	0.134	0.141
NE	0.799	0.783	0.776	0.774	0.778	0.784	0.772	0.795

But this is a global consideration, the real application of this experimental setting is a local study factor by factor.

Table 5.5: Factors selection.

Factor	*Acc.*		*QE*		*NE*	
Cont.	0.195	+	0.001	-	0.001	-
Cat.	-0.838	-	0.003	-	0.007	-
Norm.	0.148	+	-0.001	+	-0.002	+
Uniq.	-1.266	-	0.004	-	0.009	-
AO	-1.308	-	0.004	-	0.012	-

Analysing this table, that shows the global results of the Fractional Factorial Design of five factors we have studied, we can infer that the configuration with the following values has to be the default configuration:

1. *Cont.* This is the least clear decision. For accuracy, the winner similarity function is *fuzzy.zerofive* (4.13). But in the error measures, the winner is the *continuous* (4.8). Allways, the difference are of very low order. But we have to take a decision, and we choose

continuous, because it wins in error, that is what we are minimizing.

2. *Cat.* Here the *frequency* similarity function (4.6) is chosen unanimously. All the measures point out this, so *frequency* is now the default similarity measure for categoric variables.
3. *Norm.* This is the other factor that has been chosen with a little difference. Allways the use of normalization wins, but by a small difference. Anyway, the positive value that incorporates the normalization functionality to the aggregation is used in the default configuration.
4. *Uniq.* Here, the unique parameter for all the neurons has been the chosen option. This conclusion is one of the most solid conclusion we can infer, attending to the results. They agree between them in choosing this option and even with a larger difference. So, in our default configuration, we use only one σ/p parameter.
5. *AO*. In the same way that for the previous factor, *FFD* determines that is better to perform without *Alternate Optimization*. It is also a solid decision, so we will not use *AO* in the default configuration of the *HNN2*.

Such as Wettschereck and Dietterich defended since some time ago, using several p (or σ), one per neuron, can not be justified in a practical way because it doesn't improve the predictions [11]. They defend the use on a unique p (or σ) and we have arrived to the same conclusion in this Fractional Factorial Design.

Just out of curiousity, we present also a independent *FFD* for each problem. We can see the results at table 5.6.

5.B Final experimental setting

The last experimental setting has been designed to conclude if there is any method really better. We have compared our *HNN2* with the *RBF* network we have designed (*RBF2*) and that uses *Leader2* as initializer. We also compare both with another version of *RBF* networks initialized with *K-means* (*RBFk*)and the *SVM* algorithm.
As you can see, two versions of *RBF* networks are used in this comparison. That is because this kind of network is the *HNN2* natural opponent since the *HNN2* can be seen has a particular case of the *RBF* networks. On the one hand, *RBF2* has been chosen because it is a middle point between the classical *RBF* network and the *HNN2* since it uses also the *Leader2* clustering algorithm as initializer. With this method we are comparing the effect of using heterogeneous (*HNN2*) or continuous (*RBF2*) data. On the other hand, *RBF* with *K-means* gives us the performance of *RBF* networks when they are not initialized with the *Leader2* clustering algorithm but the classical option, *K-means*. Actually, here we compare *RBF* networks with *HNN2* as an specification relationship since *RBF* is an specific case of *HNN2* and here they really use the same learning method code.
Finally, we have chosen the *SVM* with radial kernel (and only with this kernel). That is because, despite the fact that *SVM* doesn't build the model in the same way than *RBF* networks, both build similar structures that could not be differenciated after the learning process. Since both return models with similar architecture and functionality, they may be the expected opponent for each other.
In this project, we don't implement any *SVM* method but we use a *SVM* version already implemented in R. That implies that we will not have any measures that we use to compare the methods, but the measures that already returns this version.

Table 5.6: Factors selection for the each data set.

Problem	Factor	*Acc.*		*QE*		*NE*	
CRX	Cont.	-1.014	-	0.003	-	0.009	-
	Cat.	-0.362	-	-0.001	+	-0.004	+
	Norm.	0.217	+	-0.003	+	-0.01	+
	Uniq.	-0.29	-	0.001	-	0.001	-
	AO	-0.145	-	0.002	-	0.006	-
HC22	Cont.	-1.244	-	0.004	-	0.011	-
	Cat.	-0.746	-	0.001	-	0.002	-
	Norm.	0.995	+	0.001	-	0.001	-
	Uniq.	-2.736	-	0.003	-	0.009	-
	AO	-1.493	-	0.003	-	0.009	-
HC23	Cont.	2.206	+	-0.004	+	-0.013	+
	Cat.	-1.225	-	0.004	-	0.01	-
	Norm.	0.49	+	-0.005	+	-0.013	+
	Uniq.	-0.001	-	0.001	-	0.004	-
	AO	-2.206	-	0.005	-	0.016	-
BANDS	Cont.	0.833	+	-0.001	+	-0.003	+
	Cat.	-1.019	-	0.008	-	0.02	-
	Norm.	-1.111	-	0.005	-	0.012	-
	Uniq.	-2.037	-	0.01	-	0.025	-
	AO	-1.389	-	0.007	-	0.016	-

In order to perform this experimental setting, we have studied several statistical tests. We base our design in the Dietterich's original idea [23]. In this article, the author compares five statistical tests for determining whether one learning algorithm outperforms another. One of these methods was designed by himself, the 5×2 CV paired t test. It uses five different partitions two-fold Cross Validation, which returns ten estimations on test sets. These estimations are combined in the following way to get the $\tilde{t}$ value:

$$\tilde{t} = \frac{p_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}} \tag{5.2}$$

where $p_1^{(1)}$ is the difference between the proportions of the two methods (A,B) we are comparing in the same partition (1) of the same replication (1), $p_1^{(1)} = {p_1^{(1)}}_A - {p_1^{(1)}}_B$. That can be generalized as $p_i^{(j)} = {p_i^{(j)}}_A - {p_i^{(j)}}_B$, with $i \in \{1,\dots,5\} and j \in \{1,2\}$.

In addition, s^2 is the estimated variance $s_i^2 = \left(p_i^{(1)} - \overline{p_i}\right)^2 + \left(p_i^{(2)} - \overline{p_i}\right)^2$, where $\overline{p_i} = \frac{\left(p_i^{(1)}+p_i^{(2)}\right)}{2}$, is the arithmetic mean also with $i \in \{1,\dots,5\}$.

If you start with the hypothesis of both problems having the same error rate, you can refute it with 95% confidence when $\tilde{t} > 2.571$. In this case, the error rate of a method is better than the error rate of the other.

Attending to the configuration of the experimental settings, they chose 2-fold cross validation because it gives large test sets and disjoint training sets. They need a big test because they only

use a single difference $p_1^{(1)}$ in the test. The disjoint training sets give some independence, situation assumed that is not real because in the replications all the instances are reused.
They chose 5 replications of the cross validation because using 5 or more replication increases the risk of Type-I error, and that is which we want to exploit.
They use only one of the observed differences of proportions rather than the mean of all of them because this mean tends to overestimate the true difference, attending to their reasoning. This is a problem generated by the lack of independence between the folds of the cross validation. That would justify the use of only one difference, so they had to make the assumption that the variance estimates s_i are independent of $p_1^{(1)}$ and that $p_1^{(1)}$ is the better difference $p_i^{(j)}$.

But there is an author that differs from Dietterich on this last point, Alpaydin. He defends the use of a f Test where all the differences are combined [24]. This test has been called Combined 5×2 CV f Test and it has the following expression:

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5} s_i^2} \tag{5.3}$$

He defines a statistic computed from the errors on the test set of the two methods adjusted with the training set. They follow the null hypothesis, that is, both methods have the same error rate. If this assumption holds, the statistic obeys a certain distribution.
In this situation, only two kind of errors could happen. If we reject the hypothesis when no difference between the errors exists, we incur a type-I error. Otherwise, if we accept the null hypothesis when a difference exists, we incur a type-II error. Using this last error, he defines the *power* of the test $(1 - Pr[Type - IIerror])$ as the probability of detecting a difference when a difference exists.

Assuming the null hypothesis, that is, assuming that both problems have the same error rate, you can refute it with 95% confidence when $f > 4.74$. Otherwise, the null hypothesis should be accepted.

Two main advantages we have found two this second approach:

1. It combines the 10 statistics calculated for each fold (5×2), which gives more robustness to the method.

2. It increases the robustness but it doesn't suppose any additional cost.

5.B.1 Final Experimental Results

We have compared the four methods that we talked about above:*HNN2*, *RBF2* (with *Leader2*), *RBF+K-means* and *SVM*.
Here you can find the results of this experimental setting.

For each problem, we have a table with the averaged results (Accuracy, Quadratic Error and Normalized Error) of every problem and, then, a second table with the statistic tests (t and f) that compare *HNN2* with the other methods in terms of the different measurements.
Remember that the *SVM* version of R doesn't returns Quadratic Error and Normalized Error for classification problems nor Normalized Error for regression problems.

Let's start with the Credit Aproval problem. We obtain a high accuracy for all the methods, being the *HNN2* the best one with a small difference. The same situation happens with the error,

Table 5.7: Credit Aproval Problem (CRX). Result of the predictions: Averaged Accuracy (*Acc*), Quadratic Error (*QE*) and Normalized Error (*NE*) for each method. The + sign indicates the best method in terms of each observed result.

Method	*Acc*		*QE*		*NE*	
HNN2	85.913	+	0.11	+	0.666	+
RBF2	85.391	-	0.117	-	0.689	-
RBFk	85.188	-	0.116	-	0.685	-
SVM	83.942	-	–	|	–	|

where the *HNN2* is again the best method with a small difference, knowing that we don't have this information for *SVM* (see table 5.7).

Since there are small differences, the statistical tests can not assure that there is a real difference between the results distribution of *HNN2* and the other methods. Only *SVM* is near to pass the *f* test because it has an averaged accuracy almost 2 percentage points less than *HNN2*. But, since the statistical tests take also into account the variance and not only the mean of the results, a real difference can not be proven.

Table 5.8: Credit Aproval Problem (CRX). Result of the tests comparing to the *HNN2* with the other methods. That is done for Accuracy (*Acc*), Quadratic Error (*QE*) and Normalized Error (*NE*), when that is possible. Any number in red color indicates that this combination of method has different distribution.

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	*t*	0.202	0.315	1.199
	f	0.837	1.337	3.276
QE	*t*	0.391	0.952	–
	f	1.652	2.884	–
NE	*t*	0.434	1.101	–
	f	1.693	3.185	–

The results for the German Credit Data problem are similar, but with a lower rate of right predictions. Here, the method that achieves a higher accuracy is the *SVM*, but we don't have error information, so we can not know if the error is lower also. The other three methods, despite the lower accuracy of *HNN2* (in this problem, it seems the worst predictor), the errors demonstrate that the three methods have an almost identical power of prediction (see at table 5.9).

For the CRG problem, the statistical tests results demonstrate what we said above, there is no way to find a real difference between the results of the different methods with this problem 5.10. Any test doesn't stand out because the equality is the general tendency.

In the Cylinder Bands problem results we found larger differences between the four methods. But, the worst method is again the *HNN2*, which have more than 5 percentage points less than the best method, the *SVM*. The error is also significantly higher in *HNN2* (see at table 5.11).

And these last considerations are confirmed with the statistical tests, specifically the *f* test that concludes that there is a significant difference between the accuracy of *SVM* and *HNN2*.

Table 5.9: Result of the predictions German Credit Data Problem (CRG).

Method	*Acc*		*QE*		*NE*	
HNN2	73.399	-	0.174	-	0.909	-
RBF2	74.419	-	0.171	+	0.901	+
RBFk	74.24	-	0.171	+	0.902	-
SVM	74.581	+	–	|	–	|

Table 5.10: Result of the tests German Credit Data Problem (CRG).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	*t*	0.358	0.351	0.285
	f	1.08	1.008	1.09
QE	*t*	1.037	1.009	–
	f	0.83	0.812	–
NE	*t*	1.069	1.029	–
	f	0.84	0.807	–

Table 5.11: Result of the predictions Cylinder Bands Problem (BANDS).

Method	*Acc*		*QE*		*NE*	
HNN2	70.37	-	0.193	-	0.888	-
RBF2	73.074	-	0.172	+	0.839	+
RBFk	72.148	-	0.175	-	0.846	-
SVM	75.593	+	–	|	–	|

This is the only result confirmed as different by the tests, but attending to the error measures, the *RBF*'s are near to be also confirmed as better methods (see at table 5.12).

Table 5.12: Result of the tests Cylinder Bands Problem (BANDS).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	*t*	0.916	0.386	0.638
	f	2.001	1.119	11.1
QE	*t*	2.214	1.993	–
	f	3.741	2.901	–
NE	*t*	2.258	2.031	–
	f	3.792	2.931	–

And the Hepatitis problem is other problem that has bad results with *HNN2*. The results are allways worse than with the other methods, although they are not so bad as in the previous problem. Now, the method with best results is the *RBF2*, but with a minimum difference with the *RBFk*. That can be said because the errors are smaller with the *RBFk* 5.13 despite the better accuracy of the *RBF2*. Anyway, there are so low differences between them.

Table 5.13: Result of the predictions Hepatitis Problem (HEP).

Method	*Acc*		*QE*		*NE*	
HNN2	82.06	-	0.125	-	0.875	-
RBF2	83.864	+	0.118	-	0.848	-
RBFk	83.613	-	0.116	+	0.841	+
SVM	82.444	-	–	|	–	|

The considerations are confirmed again and we have a positive response for the f test comparing the accuracy of the *HNN2* with the one of the *RBF2*. This is not supported by the errors, that have normal responses to the statistical tests, which could indicate that the difference is not so large as it seems (see at table 5.14).

Table 5.14: Result of the tests Hepatitis Problem (HEP).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	t	0.926	0.425	0.816
	f	5.905	1.25	1
QE	t	1.662	0.859	–
	f	2.583	2.806	–
NE	t	1.663	0.799	–
	f	2.679	2.632	–

After two problems that return no so good results when they are treated with *HNN2*, we study the Horse Colic problem. In this case, we have used its 22^{th} variable as class variable. Now the *HNN2* is the method with better results for all the measures 5.15. The differences are not so important, but the accuracy of *HNN2* is 2 percentage points better than *SVM*, for example.

Table 5.15: Result of the predictions Horse Colic Problem -Class variable: 22^{th}- (HC22).

Method	*Acc*		*QE*		*NE*	
HNN2	67.213	+	0.147	+	0.901	+
RBF2	66.667	-	0.149	-	0.907	-
RBFk	66.776	-	0.148	-	0.906	-
SVM	65.246	-	–	|	–	|

But these differences, that seem small, are confirmed by the t test. Attending to this test, the *HNN2* accuracy is really better than the accuracy of *RBF2* in this problem. *RBFk*, with a higher accuracy than *RBF2*, is also near to return a positive response to this test. These problems returns a high value in t test for the *RBF*'s, overall for *RBFk*. That could be due to the dependence of the t test on the $p_1^{(1)}$. You can see all this at table 5.16.

Table 5.16: Result of the tests Horse Colic Problem -Class variable: 22^{th}- (HC22).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	t	2.692	2.05	0.755
	f	1.783	1.193	2.38
QE	t	1.88	2.251	–
	f	0.847	1.058	–
NE	t	1.894	2.26	–
	f	0.855	1.056	–

This behaviour, where the *HNN2* is the best method, is repeated in the other problem based on Horse Colic data set. Now, the class variable is the 23^{th}. In this case, the good results overcome the other methods with a higher evidence. The *HNN2* accuracy is more than 3 percentage points better than the accuracy of the other methods for this problem. Now, this tendency can be observed also in the errors, where the *HNN2* is significantly the method with better results 5.17.

Table 5.17: Result of the predictions Horse Colic Problem -Class variable: 23^{th}- (HC23).

Method	*Acc*		*QE*		*NE*	
HNN2	83.262	+	0.128	+	0.74	+
RBF2	79.994	-	0.152	-	0.808	-
RBFk	79.94	-	0.153	-	0.809	-
SVM	80.056	-	–	|	–	|

And all these considerations are absolutely corroborated with the tests. In terms of accuracy, the tests doesn't confirm any behaviour, but the errors do it. All the tests of errors are positive (not t tests of *RBF2*, but they are near to the edge). That means that *HNN2* is here the best method, out of any doubt (See at table 5.18, where the red color that indicates a positive test response is the most common).

Table 5.18: Result of the tests Horse Colic Problem -Class variable: 23^{th}- (HC23).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	t	0.505	1.37	0.22
	f	1.874	3.053	2.584
QE	t	2.286	2.986	–
	f	15.119	12.786	–
NE	t	2.351	2.755	–
	f	14.602	10.346	–

For the next problem, the Congressional Voting Records problem, the *HNN2* remains being the best method. Here, it doesn't present the best accuracy but the errors. The best accuracy is achieved for the *RBF2*, that is a method with results very close to the *HNN2* ones. The *RBFk* doesn't arrive to the level of the two first methods and the *SVM* presents the worst accuracy 5.21.

Table 5.19: Result of the predictions Voting Records Problem -Original- (HV84).

Method	*Acc*		*QE*		*NE*	
HNN2	95.399	-	0.039	+	0.404	+
RBF2	95.495	+	0.042	-	0.417	-
RBFk	95.357	-	0.064	-	0.49	-
SVM	93.473	-	–	|	–	|

The tests results demonstrate that the differences are not so important. All the values are inside normal values, the only tests that are a little bit high are the accuracy tests of the *SVM*. We saw above they have the largest difference, but despite the difference there is not enough information to confirm it. So, we can assure that here the methods returns similar results 5.20.

Table 5.20: Result of the tests Voting Records Problem -Original- (HV84).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	*t*	0.374	0	1.971
	f	0.862	0.805	2.298
QE	*t*	0.599	0.099	–
	f	1.05	1.036	–
NE	*t*	0.582	0.16	–
	f	1.039	1.067	–

There is another problem based again on the same data set, the Congressional Voting Records. Now, the variables have a different consideration, but the target remains being the same two-classes variable. Now, the best methods seem to be the *RBF*'s, but there is no difference with *HNN2* neither. The changes in the kind of variables don't have any effect in the predictions because the results are very similar.

Table 5.21: Result of the predictions Voting Records Problem -Modified- (HV84b).

Method	*Acc*		*QE*		*NE*	
HNN2	94.755	-	0.044	-	0.431	-
RBF2	95.262	-	0.043	+	0.424	+
RBFk	95.307	+	0.045	-	0.435	-
SVM	94.57	-	–	|	–	|

The tests results for this problem are again so averaged. There is no any positive result, which means that there is a great similarity between all the methods used here applying to this problem. The results of the tests for this problem, which are in the table 5.22, confirm these considerations.

In a similar way, the results of the previous problem which uses the same data set confirmed the same (see table 5.20).

Table 5.22: Result of the tests Voting Records Problem -Modified- (HV84b).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
Acc	*t*	1.104	0.988	0.905
	f	0.759	0.914	1.283
QE	*t*	1.233	0.052	–
	f	1.303	0.536	–
NE	*t*	1.21	0.049	–
	f	1.333	0.525	–

The two problems that we have not commented yet are regression problems. These problems returns only error results, because here there exists no accuracy concept.
The first of these problems is Prostate. The best results are achieved with the *RBF*'s, which return identical results. *SVM* and *HNN2* perform a little bit worse, as you can see at table 5.23.

Table 5.23: Result of the predictions Prostate Problem (PRO).

Method	*QE*		*NE*	
HNN2	0.756	-	0.761	-
RBF2	0.681	+	0.722	-
RBFk	0.681	+	0.721	+
SVM	0.744	-	–	|

The differences between *HNN2* and the other methods can not be confirmed by any statistical test. Despite of the bad results, the high variance of this regression problem does impossible drawing conclusions. See that at table 5.24.

Table 5.24: Result of the tests Prostate Problem (PRO).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
QE	*t*	1.062	1.365	0.78
	f	2.27	2.988	3.551
NE	*t*	0.997	1.285	–
	f	2.253	2.988	–

The last problem we have studied is the second regression problem and it is called Servo. Here, the *HNN2* achieves the best predictions, closely followed by the *RBF2* first and the *RBFk* then. In the last place, the *SVM* performs extremely bad for this problem, doubling the error of the worst method between the other three 5.25.

And the tests confirm the previous considerations. There is an only positive respose corresponding to the comparation with the *SVM*. The reason to get this positive value is the *SVM*

Table 5.25: Result of the predictions Servo Problem (SERVO).

Method	*QE*		*NE*	
HNN2	0.933	+	0.619	+
RBF2	0.955	-	0.627	-
RBFk	0.99	-	0.639	-
SVM	2.226	-	–	|

strange and extremely bad response. For the others, their similar behaviours do that the tests don't find a real difference. See at table 5.26.

Table 5.26: Result of the tests Servo Problem (SERVO).

Obs.	Test	*RBF2*	*RBFk*	*SVM*
QE	*t*	1.212	1.356	2.282
	f	0.601	0.786	14.386
NE	*t*	1.127	1.266	–
	f	0.619	0.841	–

6 Evaluation of results

With the experimental setting results that we have explained above we can draw some ideas about the *HNN2* performance. This is a method that from the beginning has been built putting some hopeful functionalities together. We have thought these functionalities could help to achieve a better fitting since they represent an improvement in the different parts of *HNN*'s regarding previous versions. For example, using a clustering algorithm that allows heterogeneous data improves the input data understanding, or using regularization improves the second layer weights selection.

But these improvements are local interpretations that are not forced to perfom well together. Even if they work like we had expected, there would be a big problem configuring all these functionalities to get really good results. Note that this configuration could change for different problems because that is exactly what allows the algorithm, adaptability to adjust well many problems.

We have built a *HNN* version, the *HNN2* which has a default configuration based on the Fractional Factorial Design we have done over a small set of typical heterogeneous problems. These are problems that fit better the concept of heterogeneous problem we want to adjust. By lack of time, an study for choosing the better configuration has not been done; instead, the default configuration has been used with the 10 problems.

The first experimental setting, the Fractional Factorial Design, was developed for Credit Approval, Horse Colic (using class variable 22 and 23) and Cylinder Bands. The global result was that no *Alternate Optimization* had to be performed in the default configuration, that only one unique p/σ parameter had to be used, using *continuous* (4.8) to treat continuous variables and *frequency* (4.6) to categoric variables, combining the partial similarities with a normalized aggregator.

Problem by problem, we observe different behaviours. For example, attending to accuracy, *CRX* would used *frequency* (4.6) to treat categoric variables, but since we were optimizing regarding *quadratic error* the preferent option attending to *QE* is more important. So, for this problem, the factorial selection is the same that the general selection but using *overlap* (4.5) to treat categorical variables.
Contrarily, analysing *BANDS* we could decide to consider continuous variables as fuzzy variables and not perform partial similarities normalization because the results are so unanimous.
A different case would be *HC23*, which performs better with fuzzy variables and normalization. The selection of *HC22* is so similar to the global selection, but it would discard the partial similarity normalization because despite of being good for accuracy, it's not good for error (as we have seen before in *CRX*).

Attending to the results, there are two factors that have a minimum difference between the two options: A and C (see all the factors at page 56). Both have very similar results, which causes that in table 5.5 we find some numbers closed to 0. That means that in general using an option is the same that using the other one. Despite the small difference, we had to decide a default option and we don't have any reason to contradict this experimental setting, so we had chosen the option with this small advantage.
The decision about factor D is supported by an unanimous behaviour (see at table 5.6) in all the problems, where its use is not recommended. Some authors have defended this conclusion, as Wettschereck and Dietterich, who said that using several p/σ (one per neuron) is not justified because it doesn't improve the predictions [11].
There is a good result to conclude the use of *frequency* partial similarity function with categorical variables. This can be explained for the more extrem behaviour of the *overlap* function, which is the $+$ option of the factor B and returns only 0 or 1, whereas the other option, *frequency*, returns more distributed partial similarities. In this case, the decision is not unanimous between all the problems, because *CRX* would use *overlap*, but there is a wide agreement between the other problems.
Finally there is the factor E, which decides about whether using the *AO* in the default configuration or not. The conclusion is that the algorithm performs better without using it, what is an unanimous decision between all the problems again. That happens probably because it is an unlimited p/σ optimization that could generate an overfitted model. And an overfitted model generates a greater error that goes against what we are optimizing.

Note that this configuration has been chosen attending to some problems, so it is possibly determined by them and a deeper experimental setting should be done to set a more specific default configuration. These are also a subset of decisions, if you want you could find new decisions to be taken that we have no studied by lack of time or less importance. Anyway, each problem will require its own configuration, so the default configuration won't work allways well in the same way we have experimented with Cylinder Bands, for example.

In the case of the final experimental setting, the default configuration of the *HNN2* is compared with the *RBF2*, the *RBF with K-means* and the *SVM* with radial kernel. This comparison is performed attending to ten different problems, which results are used to draw some conclusions.

If we see the final results problem by problem, we observe different situations.

For the Credit Approval problem all the methods achieve an averaged accuracy near to 85%. The best method seems to be the *HNN2*, with the higher accuracy and the lower errors. Despite of having a good difference, the statistic tests don't conclude that the *HNN2* is really better (see at tables 5.7 and 5.8).
German Credit Data is the second problem we have studied achieving an averaged accuracy near to 74% with all the methods. Here, the method with higher accuracy is the *SVM* and, knowing that we have no data about the *SVM* errors, the method with lower errors is the *RBF2*. The differences between methods are so small (table 5.9), so the statistic tests are no conclusive, it is impossible to sure that some method is the best one (table 5.10).
The third data set, Cylinder Bands, is not well-fitted with the default configuration of the *HNN2*. The methods accuracy goes from the 70% of the *HNN2* to the 75% of the *SVM*. This is the better method for accuracy, but attending to the errors, the best method is again the *RBF2*. As you can see at table 5.11, the differences here are higher and the *HNN2* is really worse than other methods. The test results seems to bear this idea out since f test between the *HNN2* and the *SVM* is positive (see at table 5.12). With the *RBF* networks the difference is lower and statistic tests

can not confirm it.
A similar situation is found with the Hepatitis problem. Here, the accuracy achieved by all the methods is near to 83%, specially the *RBF2* that arrives almost to 84%. However, the lower errors are achieved by the *RBF + K-means*, with a minimum difference over the *RBF2* (see at table 5.13). The accuracy is really the only measurement where there exists an important difference with the *HNN2* and this can be demonstrated with the f test between the *HNN2* and the *RBF2* that is positive. For the other methods tests can not find difference (table 5.14).
For the Horse Colic we have found two similar behaviours despite of the two different class variables. The *HC22* is fitted with an accuracy between 65% of the *SVM* and 67% of the *HNN2*. The *HNN2* is the best method again in accuracy, but also in errors. The differences with other methods are not so large, but we have got a positive value for the t test between the *HNN2* and the *RBF2* (see at tables 5.15 and 5.16). In this way, the *HNN2* is also the method that works better with *HC23*, achieving an accuracy of 83%, whereas the other methods are near to 80% of accuracy. The errors are also lower with the *HNN2*, even with an important difference (table 5.17). This behaviour is confirmed by the tests, with a great number of positive responses: comparing the *HNN2* with the *RBF + K-means*, *QE* and *NE* are confirmed by t and f tests to be really lower with our algorithm; comparing the *HNN2* with the *RBF2*, *QE* and *NE* are only confirmed by f tests to be really lower with our algorithm (table 5.18).
There is another data set with two different studied problems, United States Congressional Voting Records. Fitting the *HV84*, the method that does it better is the *HNN2*, with the best averaged errors. This method is closely followed by the *RBF2*, which wins in accuracy with 95.5%, but the others are farther. Anyway, these differences are not so important because any tests returns a positive response (see at tables 5.19 and 5.20). However, the second problem, *HV84b*, is fitted better by the *RBF* networks. The best accuracy is achieved by the *RBF + K-means* (95.3%), and the worst is achieved by the *SVM* (94.5%). The *RBF2* errors are the lowest ones, followed closely by the *HNN2* errors. As you can see at table 5.21, the differences are not so large, so the statistic tests return non-positive responses (table 5.22).
Finally, we have studied two regression problems, so we only dispose of the error measurements. For the first one, Prostate, the *RBF* networks return the best results. Both have almost equal averaged errors, being a little bit larger the *HNN2* errors (see at table 5.23). Anyway, the differences are no too large because the tests don't return any positive response comparing the methods (table 5.24). The second problem, Servo, shows a different behaviour: the lower errors are achieved with the *HNN2*, followed by the *RBF2*. The differences are higher than in the previous problem but only the f test between the *HNN2* and the *SVM* is positive. That is because the *SVM* achieves an extremely bad averaged error.

In this study there are two very close situations that could be in-depth compared.
One of these situations is the Horse Colic data set, that has been studied in two different problems. *HC22* uses the 23th variable as class variable (it has three possible values) and *HC23* uses the 24th (two possible values). Having different class variable, the first layer is allways the same because it is based on a clustering performed over the rest of variables. But the second layer, when supervised learning starts, is built in separated ways. Variable 24 seems to be more predictable than variable 23 because all the methods return higher accuracy for the first one (an average over 83% versus 66% of variable 23). Having different number of possible classes could be a reason to get lower accuracy (if there are more possible classes, the probability of failing rises), but that is not probable because both problems use the same code and algorithm.
Luckily both problems had been studied before with the Fractional Factorial Design and their

best configurations differ (table 5.6) from the default configuration. That could indicate that the obtained results could be better changing the configuration, so in this way some differences could be explained also. Note that the advantage of *HNN2* over the rest of methods is higher in *HC23* than *HC22*.

The second situation is the United States Congressional Voting Records data set, that has been treated also in two different ways. The difference here is that *HV84* considers that the variables are binary whereas the *HV84b* considers them caterogic. The results with all the methods vary a little bit between the two problems, only by the redistribution of instances because they use transformed continuous variables. But the results with the *HNN2* change, and that only can have a reason, considering binary variables is better than considering them categoric.

Analysing the situations where *HNN2* seems to perform worse than other methods, we have found several reasons that despite of using each reason to explain different problems, maybe more than one of these reasons is affecting not only one problem.
The first problem where we observed *HNN2* low performance is *CRG*. Here, attending to the tests results, there is a minimum difference between *HNN2* and the other methods. Actually, we can not say that there is *HNN2* low performance since all the tests return negative responses. If there is difference between the *HNN2* and the other methods but the tests are negative, probably the methods results have a high variance that compensates the average. That means that attending to the instances separation (training/test) the results vary so much. Also there is the uncertainty of knowing if there is another *HNN2* configuration that fits *CRG* better.
A very similar case is the *HV84b* problem. The *HNN2* has worse results with this problem (comparing it with the other methods) but the tests can not confirm a real difference because it is not so large and a high variance can be assumed. The explanation would be the same than above but here we can compare this problem with the *HV84*, that uses the same data set. Attending to the *HV84* results, we can see that the *HNN2* is the method that fits better the problem. Knowing that the difference between *HV84* and *HV84b* is only the type of variable we assume for variables of the data set (which is a configurable point), we can assure that this problem has a bad *HNN2* configuration and that is the reason of its poor fitting.

A third case of bad results with the *HNN2* is *BANDS*. Here the difference with the rest of methods is higher, arriving to be confirmed the difference between the *HNN2* and the *SVM* by a f test. But analysing these results we dispose of an extra information, the *FFD* results for *BANDS* (table 5.6). This experimental setting was so conclusive in this case and determined that its best configuration should have no normalization and treat continuous variables as fuzzy. These two differences regarding the default configuration could justify these poor results because we are aware of that the *HNN2* is not using its best configuration to fit *BANDS*.

Otherwise, the problem associated with the bad results in *HEP* and *PRO* could be made by the small number of instances in their data sets. Both have poor results (worse in *HEP*) due to their high s_{min} variability in the 10 learning runs and, then, also the number of clusters. What happens in this case is that the division on training/test is more important because there is a small number of instances and the training set maybe is not enough to represent the whole problem, so the learning is not complete. Then, the test set returns an important error with a high variance.
There is a third problem which has a data set with less than 200 instances, *SERVO*. But this problem has a more deterministic behaviour because regarding to the *HNN2* executions results, there is no high variability and s_{min} and the number of clusters is allways so similar.
That demonstrate that the result is influenced by the kind of problem and the number of instances. If there is a problem very difficult to be learned, the algorithm will need more instances to fit well

the problem. Of course, there are other reasons because other methods are not so affected by the variability. Probably the default configuration is not the best one to cope with these problems.

As secondary observation of this final experimental setting, we could point out that the *RBF2* is better than the *RBF + K-means*, which means that it is better to initialize a *RBF* network using the *Leader2* clustering algorithm than using the *K-means*.
We point out that attending to the results problem by problem. Most of them have similar results, where one of both *RBF* networks has better results but with a tiny difference. But in the situations where that doesn't happen and a higher difference is found (problems *HV84*, *HV84b* and *SERVO*), the best method is always *RBF2*. Note that both methods use transformed continuous variables and the same neurons model, so the only difference comes from the clustering result that builds the first layer.
Taking into account that algorithms can be reconfigured and probably get better results we only can say that at least the *Leader2* is not worse than the *K-means* to initialize *RBF* networks.

7

Discussion and conclusions

These results demonstrate that the *HNN2* is a good choice when you are looking for a method to fit an heterogeneous data set. If you look into the final experimental setting, that is where there is a real comparison with other learning methods, a half of the problems are better fitted using the *HNN2* with its default configuration. Despite of the rest of problems are better fitted with other methods (this situation has been justified above), the improvement capacity of this algorithm by changing its configuration is demonstrated.
So, despite of being a new method, it performs at least as well as other classical methods. That has more relevance if you note that this method has been built as the result of assembling different functionalities that represent concepts we wanted to our method. We were confidents of achieving a great performance, because these are concepts we thought necessaries to this kind of learning method, but assembling them is not trivial and nobody could sure that it would work well. The *HNN2* has to be consider as an assembly of a set of good ideas that really works well.

This new algorithm uses the similarity concept, which we consider a more intuitive way to stablish how resemblant are two examples. There is a maximum (1) and a minimum (0) of similarity and it is increasing, that is the more similar the higher value. So you can stablish an scale that the distance concept, which is defined in $[0,+\infty]$, doesn't allow you to stablish because there is no a maximum distance (and then, there is no middle and other reference points). In the practice, it is possible to stablish the similarity between two kinds of tables, for example, but it is not possible with distance.

One of the main characteristics of this method is that it is able to treat with original data, without transformations. Without taking into account specific cases, any data transformation entails some knowledge loss because right continuous representation is not allways possible for any data type. The algorithm incorporates a high variaty of data types and functions, but it also allows users to incorporate their own data types and similarity partial functions. So, it is highly customizable using personal configurations.
Performing a complete study for each one of the ten problems we have used in this research project we would probably get ten different configurations. That demonstrates the adaptability of this method, which is a basic characteristic of methods allowing heterogeneuos data. Adaptability is required because treating with heterogeneous data the method is adapted to the changeable data, whereas data is adapted to the method when a classical algorithm is used. The advantage of adapting the method is that we can treat the data in its original codification. The *SVM* requires an optimal way to transform heterogeneous data in its continuous representation. Otherwise, the *HNN2* requires an optimal choice of partial similarity functions that compare two values of a variable in its original state.
People that know about the data set could know a way to compare a variable, but it could not have a continuous immediate representation. But also, if an heterogeneous variable has a continuous

immediate representation, there is at least a way to compare it using a partial similarity function since there exist well known similarity functions for continuous variables. In this way, data transformation can not be considered better than heterogeneous functions.

Attending to experimental results, the two test positive values that confirmed that the *HNN2* performs worse with some problems than other algorithms, both are accuracy tests. That has to be emphasized because this algorithm optimizes the quadratic error, so the accuracy is only an associated measurement that really has less importance than error. In this way, if there is a positive value for accuracy but this bad performance of the algorithm is not confirmed then by a positive value for error, the tests results can not be used as a conclusive argument.
In this way, we can say that the *HNN2* is not worse than the other methods. We have brought natural opponents face to face and our method is probably the best method, and here it just has been used with its default configuration. Maybe it is a method that requires an in-depth configuration to get the optimal results, but if the user disposes of expert knowledge, the *HNN2* is able to take advantage of this knowledge and use it to learn better the problem. Actually, we don't know many methods that take advantage of the expert knowledge as much as *HNN2* does it (many data types, specific partial similarity measures, other configurable parameters: p, q, normalization...). Anyway, the default configuration doesn't returns worse results in average.

Regarding to other *Leader2* version developed in this research project, the supervised version was not taken into account in the experimental settings because it breaks the similarity principles because two instances belonging to the same cluster in the unsupervised version (they are more similar than s_{min}), they could be now in different clusters attending to their class despite of being so similar. This version is only useful when the data is naturally separated, and we know by [2] that the *Leader2* algorithm recognize perfectly this situations without the supervised version. The preliminary practical results demonstrate that it was not useful. The opposite case is the distance version, which is so useful since it is the same algorithm than the similarity version, with performance improvements, but using the distance concept.

We have built a supervised learning method, an artificial neural network that uses simpler data representations and more understandable concepts. That is achieved without loss of performance because the *HNN2* equals or outperforms other classical methods as *RBF* networks or the *SVM*. The *HNN2* guarantees also an interpretable neural network structure, because even the neuron centers are original input data. Inside all this understanding tendency we can fit the lower number of hidden neurons usually associated with the *HNN2*, which is important because complex models are not human interpretable.

7.A Future work

During the development of this thesis we have found some topics that could be researched but we have not done it because of lack of time, size and complexity of the anticipated work or being unrelated with our research.

Some of these topics are:

1. Since the *HNN2* is right now completely written in R language, it is not so fast because that is not a language optimized to programing. R-project allows users to write the more complex parts of the code in other languages (C/C++, for example) to make algorithms faster. That would be a basic tasks to improve the *HNN2* performance.

2. σ regularization. Maybe one of the problems that we have experimented with the *Alternate Optimization* is that σ has been optimized perhaps until falling in overfitting. So, we have thought that some σ regularization method could be used to avoid this over-optimization.

3. Local regularization. We have implemented a global regularization because Orr points out that local regularization is an unnecessary waste of resources for the minimum improvements achieved in practice [13]. But there are methods, one of them defined by Orr himself, that perform regularization using a λ_i different to each weight. That could be tested in the *HNN2*.

4. Calculating p in our proposed heuristic method, it takes the mean similarity as reference behaviour. Another option, a little bit more elaborated would be to use the instances distribution entropy for each cluster.

5. We have worked with an algorithm that transforms classification problems into regression ones to perform. An interesting work could be to avoid this transformation extending the weights regularization, which was the functionality that doesn't allow us to treat classification problems.

6. Incorporate new partial similarity functions, aggregators, heuristic methods to initialize p/σ, etc.

7. New tests should prove the aggregator with different q values, different partial similarity measures, different heuristic methods to initialize p/σ, different initial λ values, etc.

8. Studying the real value of a *RBF* network initialized with the *Leader2* clustering algorithm. In our tests it seems to be better that the version initialized with *K-means*, but that can not be a right conclusion since these tests were not prepared to demonstrate this relationship. The idea of considering *RBF* network a particular case of the *HNN2* could teoretically support this conclusion. So an experimental setting comparing both versions of *RBF* networks could be performed.

9. The *HNN2* default configuration we have proposed has been set attending to the Fractional Factorial Design results and some previous good configurations. A better configuration could be found doing a larger study, including a higher number of problems and new decisions that we have considered well configured. Anyway, using expert knowledge would be the best way and getting it should be a priority to configure each problem independently, that is the correct way.

Bibliography

[1] Ll. Belanche. Similarity-based heterogeneous neuron models. In *Proceedings of* 14^{th} *European Conference on Artificial Intelligence*, Berlin, Germany, 2000.

[2] J. Hernández and Ll. Belanche. Algoritmos de clustering basados en el concepto de líder. Master's thesis, Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya, June 2009. Available at http://upcommons.upc.edu/pfc/handle/2099.1/7667.

[3] J. Hartigan. *Clustering Algorithms*, chapter 3: Quick Partition Algorithms. Wiley, 1975.

[4] Ll. Belanche. *Similarity-based Heterogeneous Neural Networks*. PhD thesis, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2000.

[5] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[6] J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971.

[7] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[8] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[9] Ll. Belanche. A theory for heterogeneous neuron models based on similarity. Technical report, Barcelona, Spain, 2000. Available at http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=427.

[10] Simon Haykin. *Neural Networks, a Comprehensive Foundation*. Prentice Hall, 1999.

[11] Dietrich Wettschereck and Thomas Dietterich. Improving the performance of radial basis function networks by learning center locations. In S. J. Hanson J. E. Moody and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 1133–1140. Morgan Kaufmann Publishers, 1992.

[12] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[13] Mark J. L. Orr. Introduction to radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, 1996. Available at http://www.anc.ed.ac.uk/rbf/papers/intro.ps.gz.

[14] Mark J. L. Orr. Recent advances in radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, 1999. Available at http://www.anc.ed.ac.uk/rbf/papers/recad.ps.gz.

[15] Ross Ihaka Robert Gentleman et al. The R-Project for statistical computing. http://www.r-project.org/.

[16] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the* 15th *International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann Publishers, 1998.

[17] J.J. Valdés Ll. Belanche. *Fuzzy Inputs and Missing Data in Similarity-Based Heterogeneous Neural Networks*, pages 863–873. Mira, Sánchez-Andrés (Eds.), Springer-Verlag, 1999.

[18] C. Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of* 20th *International Conference on Machine Learning*, pages 147–153. AAAI Press, 1998.

[19] A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of* 12th *Conference on Uncertainty in Artificial Intelligence*, pages 397–405. AAAI Press, 2000.

[20] A. Huneiti M. Al-Zoubi, A. Hudaib and B. Hammo. New efficient strategy to accelerate k-means clustering algorithm. *American Journal of Applied Sciences*, 5(9):1247–1250, 2008.

[21] Amaury Lendasse John Lee Nabil Benoudjit, Cédric Archambeau and Michel Verleysen. Width optimization of the gaussian kernels in radial basis function networks. In *European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 425–432. D-side publications, 2002.

[22] National Institute of Standards and Technology. Fractional factorial design. `http://www.itl.nist.gov/div898/handbook/pri/section3/pri334.htm`.

[23] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[24] Ethem Alpaydin. Combined 5×2 cv f test for comparing supervised classification learning algorithms. *Neural Computation*, 11(8):1885–1892, 1999.

[25] A. Frank and A. Asuncion. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, 2010. University of California, Irvine, School of Information and Computer Sciences.

I

Annexes

A

Reasoning the fuzzy similarity functions

A.A Triangle-shaped functions

We have two fuzzy numbers represented by two triangle-shaped functions and we are only interested in the point where they cut. This point is in $y \leq 1$, but if $y < 0$ it is replaced by $y = 0$. This is the similarity value returned by this approach.
Each function has to be built by two straight lines that cut at $(x_{ik}, 1)$ and has a base of length $2 * P$. Having this, we know two reference points for each one of the two lines: $(x_{ik} - P, 0), (x_{ik}, 1)$ and $(x_{ik}, 1), (x_{ik} + P, 0)$ (the first one is increasing and the second one decreasing).
In the first case, the increasing line has the following function:

$$y = a \cdot x + b, \text{ and } (x_{ik} - P, 0), (x_{ik}, 1)$$

$$0 = a \cdot (x_{ik} - P) + b, \text{ and } 1 = a \cdot x_{ik} + b$$

$$a \cdot x_{ik} - a \cdot x_{ik} + a \cdot P = 1$$

$$a = \frac{1}{P}, \text{ and } b = \frac{P - x_{ik}}{P}$$

So the function is:

$$y = \frac{1}{P}x + \frac{P - x_{ik}}{P} \tag{A.1}$$

In the second case, the decreasing line has a different function:

$$y = a \cdot x + b, \text{ and } (x_{ik}, 1), (x_{ik} + P, 0)$$

$$1 = a \cdot x_{ik} + b, \text{ and } 0 = a \cdot (x_{ik} + P) + b$$

$$a \cdot x_{ik} - a \cdot x_{ik} - a \cdot P = 1$$

$$a = -\frac{1}{P}, \text{ and } b = \frac{P + x_{ik}}{P}$$

So the function is:

$$y = -\frac{1}{P}x + \frac{P + x_{ik}}{P} \tag{A.2}$$

Additionally, we have two functions that could be written as:

$$\left.\begin{aligned} y = a_1 \cdot x + b_1 \\ y = a_2 \cdot x + b_2 \end{aligned}\right\} \tag{A.3}$$

where the number 1 indicates that this is the decreasing function and the number 2 indicates the increasing function. If we match them we get:

$$a_1 \cdot x + b_1 = a_2 \cdot x + b_2$$

$$(a_1 - a_2) \cdot x = b_2 - b_1$$

$$x = \frac{b_2 - b_1}{a_1 - a_2}$$

And replacing x in some of the two previous functions:

$$y = \frac{a_1 \cdot (b_2 - b_1)}{a_1 - a_2} + b_1 = \frac{a_1 \cdot b_2 - a_1 \cdot b_1 + a_1 \cdot b_1 - a_2 \cdot b_1}{a_1 - a_2} = \frac{a_1 \cdot b_2 - a_2 \cdot b_1}{a_1 - a_2}$$

Now, we have the point (x, y) where two straigh lines cut, and we are going to do the specific development for two triangle-shaped fuzzy numbers x_{ik}, x_{jk}.
We incorporate two kind of triangle-shaped functions. The first one, where the width of the function doesn't change, cut at the point:

$$x = \frac{\frac{P - x_{jk}}{P} - \frac{P + x_{ik}}{P}}{-\frac{1}{P} - \frac{1}{P}} = \frac{\frac{x_{jk} + x_{ik}}{P}}{\frac{2}{P}} = \frac{x_{ik} + x_{jk}}{2}$$

which is the arithmetic mean. And for y we do:

$$y = \frac{-\frac{1}{P} \cdot \frac{P - x_{jk}}{P} - \frac{1}{P} \cdot \frac{P + x_{ik}}{P}}{-\frac{1}{P} - \frac{1}{P}} = -\frac{\frac{x_{jk} - P}{P^2} - \frac{x_{ik} + P}{P^2}}{\frac{2}{P}} = -\frac{x_{jk} - x_{ik} - 2 \cdot P}{2 \cdot P} = \frac{x_{ik} - x_{jk}}{2 \cdot P} + 1$$

Note that here you have to know which is the higher value to be used at the increasing function leaving the lower one to the decreasing function. But we don't have to know this order. Instead of finding out the order, we introduce a little change at the y function in order to generalize it. At the x function that is not necessary because there is a summation. Here the problem is generated by the substraction at y. The change we introduce is the absolute value of the substraction:

$$y = 1 - \frac{|x_{ik} - x_{jk}|}{2 \cdot P}$$

where the new substraction returns the decreasing behaviour to y, because it will be used as similarity measure.
Using all this information we build the similarity function:

$$s_k(x_{ik}, x_{jk}) \equiv \begin{cases} 1 - \frac{|x_{ik} - x_{jk}|}{2 \cdot P} & \text{if } |x_{ik} - x_{jk}| \leq (2 \cdot P) \\ 0 & \text{otherwise} \end{cases} \tag{A.4}$$

The same base reasoning is used to build the similarity function where the shape of the convex function depends on some percentage P of the value that represents. The problem that introduces this approach is that the width is different at each fuzzy number, so the simplification follows a different way:

$$x = \frac{\frac{x_{jk} \cdot P - x_{jk}}{x_{jk} \cdot P} - \frac{x_{ik} \cdot P + x_{ik}}{x_{ik} \cdot P}}{-\frac{1}{x_{ik} \cdot P} - \frac{1}{x_{jk} \cdot P}} = \frac{\frac{2}{P}}{\frac{x_{ik} + x_{jk}}{x_{ik} \cdot x_{jk} \cdot P}} = \frac{2 \cdot x_{ik} \cdot x_{jk}}{x_{ik} + x_{jk}}$$

which is the harmonic mean. And for y we do:

$$y = \frac{\frac{x_{jk}-x_{jk}\cdot P}{x_{ik}\cdot x_{jk}\cdot P^2} - \frac{x_{ik}+x_{ik}\cdot P}{x_{ik}\cdot x_{jk}\cdot P^2}}{-\frac{1}{x_{jk}\cdot P} - \frac{1}{x_{ik}\cdot P}} = \frac{\frac{x_{jk}-x_{jk}\cdot P - x_{ik} - x_{ik}\cdot P}{x_{ik}\cdot x_{jk}\cdot P^2}}{-\frac{x_{ik}+x_{jk}}{x_{ik}\cdot x_{jk}\cdot P}} = \frac{-x_{jk}+x_{jk}\cdot P + x_{ik} + x_{ik}\cdot P}{(x_{ik}+x_{jk})\cdot P} = \frac{x_{ik}-x_{jk}}{(x_{ik}+x_{jk})\cdot P} + 1$$

The situation is the same that before. Then, we do the same in order to avoid the substraction problems:

$$y = 1 - \frac{|x_{ik}-x_{jk}|}{(x_{ik}+x_{jk})\cdot P}$$

Using all this information we build the similarity function:

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 1 - \frac{|x_{ik}-x_{jk}|}{(x_{ik}+x_{jk})\cdot P} & \text{if } |x_{ik}-x_{jk}| \leq ((x_{ik}+x_{jk})\cdot P) \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

A.B Trapezoid-shaped functions

A trapezoid is a four-sided figure that has a pair paralel and the another pair not paralel. The paralel pair has a large side (bottom) and a short one (upper). In this work we impose that the non-parallel pair of sides have the same degree of inclination. It can be seen as an isosceles triangle that has been cut out disappearing its upper (different) angle. In this way, it can be also seen as an isosceles triangle of *height* $= 1$, that has been separated into two equal parts (two right triangles) and then recombined putting in the middle of the two parts a rectangle of *height* $= 1$. Using this last point of view, the fuzzy trapezoid-shaped similarity function is based on the first fuzzy function we present here A.4.

$$s_k(x_{ik},x_{jk}) \equiv \begin{cases} 1 & \text{if } |x_{ik}-x_{jk}| \leq (2\cdot P_a) \\ 1 - \frac{|x_{ik}-x_{jk}|-(2\cdot P_a)}{2\cdot(P_b-P_a)} & \text{if } |x_{ik}-x_{jk}| > (2\cdot P_a) \text{ and } |x_{ik}-x_{jk}| \leq (2\cdot P_b) \\ 0 & \text{otherwise} \end{cases} \tag{A.6}$$

In this approach P_a is the length of the bottom side and P_b the length of the upper side. In the first term of this similarity function, if the values are so near (less than $2\cdot P_a$) the similarity is the maximum. When the values are farther, $2\cdot P_a$ (the rectangle length) is substracted to use the triangular approach. Finally, if they are so far, there is no similarity between them.

A.C Exponential functions

We have two fuzzy numbers represented by two exponential functions and we are only interested in the point where they cut, that is in $y \in (0,1]$, and it is the similarity value returned by this approach. The exponential function as this appearance:

$$exp(x) = e^{-\frac{(x-\mu)^2}{2\cdot\sigma^2}}$$

Using this function, we derivate the similarity measure that is returned for two values x_{ik}, x_{jk}. We have to generate two exponencial functions with the same width (σ^2) and their centers located at the values $\mu_1 = x_{ik}$ and $\mu_2 = x_{jk}$:

$$exp_1(x) = e^{-\frac{\left(x-x_{ik}\right)^2}{2\cdot\sigma^2}} \quad exp_2(x) = e^{-\frac{\left(x-x_{jk}\right)^2}{2\cdot\sigma^2}}$$

If we match the two exponentials we get:

$$exp_1(x) = exp_2(x)$$

$$e^{-\frac{(x-x_{ik})^2}{2\cdot\sigma^2}} = e^{-\frac{\left(x-x_{jk}\right)^2}{2\cdot\sigma^2}}$$

And simplifying this equality:

$$\frac{(x-x_{ik})^2}{2\cdot\sigma^2} = \frac{\left(x-x_{jk}\right)^2}{2\cdot\sigma^2}$$

$$(x-x_{ik})^2 = \left(x-x_{jk}\right)^2$$

Now, we have to expand the power and simplify again:

$$x^2 - 2xx_{ik} + x_{ik}^2 = x^2 - 2xx_{jk} + x_{jk}^2$$

$$-2xx_{ik} + x_{ik}^2 = -2xx_{jk} + x_{jk}^2$$

And, finally, isolating the x variable:

$$-2x\left(x_{ik} - x_{jk}\right) = x_{jk}^2 - x_{ik}^2$$

$$x = \frac{x_{ik}^2 - x_{jk}^2}{2\left(x_{ik} - x_{jk}\right)} = \frac{\left(x_{ik} + x_{jk}\right)\left(x_{ik} - x_{jk}\right)}{2\left(x_{ik} - x_{jk}\right)} = \frac{\left(x_{ik} + x_{jk}\right)}{2}$$

We get a simplified value of x that is replaced in one of the two exponential functions:

$$exp_1\left(\frac{\left(x_{ik} + x_{jk}\right)}{2}\right) = e^{-\frac{\left(\frac{\left(x_{ik}+x_{jk}\right)}{2} - x_{ik}\right)^2}{2\cdot\sigma^2}}$$

And this is the exponential-based fuzzy similarity function:

$$s_k(x_{ik}, x_{jk}) \equiv e^{-\frac{\left(\frac{\left(x_{ik}+x_{jk}\right)}{2} - x_{ik}\right)^2}{2\cdot\sigma^2}} \tag{A.7}$$

B

Reasoning an heuristic to approximate *p*

We have created a new learning method, the *Heterogeneous Neural Network 2*, and we have to provide answers to all the open questions. One of these questions is the default value p used by the algorithm.

Belanche [9] used $p = 0.1$, because it draws a graphic with a curve not so heavy, that is, an averaged behaviour. We respect this consideration, but we propose a changing p value attending to the clustering results that initialize the *HNN2*.

First of all we define a coefficient that characterizes the clusters, the *compactness index*, a real value $IC_i \in (0,1)$ that is higher the more compact is the cluster i. Defining the concept of compactness for this method, we define two measures:

$l_i = |[\mu_i]|$. It is the number of instances in the cluster i.

$m_i = \bar{s}([\mu_i])$. It is the averaged similarity of the instances of the cluster i with its leader μ_i.

So, based on these two measures, we say that a cluster i is more compact the higher both l_i and m_i are.

We have defined the $IC \in (0,1)$, but p is a real number defined in all the positive numbers ($p \in (0,+\infty)$). Then, we need a function that transforms $r : IC \rightarrow p$, that is, $r : (0,1) \rightarrow (0,+\infty)$. This behaviour is easily found using logarithms, so we define r as:

$$r(z) = -\ln z = \ln \frac{1}{z}$$

Now, we know the transformation needed and just left defining the method to calculate the *compactness index*. We like function $f(x) = \frac{x}{x+n}$ because it is defined in $(0,+\infty) \rightarrow (0,1)$ and the n value allows you to change the growing curve of the function.
We have thought that the better combination of l_i and m_i is multiplying them ($f(l_i * m_i)$). l_i is a natural number ($l_i \in 1,\ldots,h$) and m_i is a similarity measure, so $m_i \in [0,1]$. This multiplication produces a value $x = l_i \cdot m_i$ that is lower or equal than l_i. The higher similarity mean, the closer x and l_i are. This behaviour fits with the definition of compactness we have, both coefficients have to be high to get a high *index*. Right now, we have:

$$IC_i = \frac{l_i \cdot m_i}{l_i \cdot m_i + n}$$

So, the n parameter will be which differenciates the different clusters. If there are two identical clusters (same l_i and m_i) of different clusterings, their IC could be different attending to their clustering. That has to be reflected in n.

Attending to the function f, when $n \to 0, f \to 1$. So, the n has to represent a averaged behaviour $\bar{x}$ that allow the IC to get high values when $x > \bar{x}$ and lower values when $x < \bar{x}$. So we defined $n = \alpha\overline{m}\bar{l}$, where $\overline{m}$ is the average of the similarity of all the instances in the data set with their leader, $\bar{l}$ is the number of instances divided by the number of clusters and α a parameter to fit the relation.
We want the function to get $p_i = 0.1$ when the cluster i is averaged ($m_i = \overline{m}$ and $l_i = \bar{l}$). So, we have $r(IC_i) = 0.1$ and if we spread out the function:

$$-\ln\left(\frac{\overline{m}\bar{l}}{\overline{m}\bar{l} + \alpha\overline{m}\bar{l}}\right) = 0.1$$

$$\ln\left(\frac{\overline{m}\bar{l} + \alpha\overline{m}\bar{l}}{\overline{m}\bar{l}}\right) = 0.1$$

$$\ln(1+\alpha) = 0.1$$

$$1+\alpha = \exp(0.1)$$

And we get the value of α:

$$\alpha = \exp(0.1) - 1$$

Now, using this α value, we can defined the IC as:

$$IC_i = \frac{m_i l_i}{m_i l_i + (\exp(0.1) - 1)\overline{m}\bar{l}} \tag{B.1}$$

And then, the heuristic we propose to calculate p is:

$$p_i = r(IC_i) = r\left(\frac{m_i l_i}{m_i l_i + (\exp(0.1) - 1)\overline{m}\bar{l}}\right) = -\ln\left(\frac{m_i l_i}{m_i l_i + (\exp(0.1) - 1)\overline{m}\bar{l}}\right) \tag{B.2}$$

In the cases where an only p parameter is used, it could be calculated using the mean of the p parameters of all the clusters. That is, when $\hat{p} = \hat{p}_i \forall i, \hat{p} = \overline{p}_i$, where $\overline{p}_i$ is the p_i calculated mean and $\hat{p}$ is the global unique p value.

C

Data sets used in the experimental settings

In order to test our algorithm we have chosen some known problems. These have different data types and dimensions. The only thing they have in common are the heterogeneous data. Many of them are taken from the UCI Repository [25].

Now we will explain them in depth to understand the decisions we have taken over the data. Remember that these decisions are so important in our method because imply an specific similarity function for each variable.

CRX: Credit Approval

This is a data set available at the UCI repository:
`http://archive.ics.uci.edu/ml/datasets/Credit+Approval`.

It was created by Quinlan, from School of Information Technologies in the University of Sydney (Australia). This data set concerns credit card applications.

There are 16 variables and the last one is the class variable. The data set has 690 examples and the 5% of the instances (37) have *missing values*. The class variable has only two possible values (classes), where the first one (+) is the 44.5% of the examples and the second class (-) the other examples (55.5%).

The variables are:

1. Variable 1: Without any other information we only have the possible values: a and b.
   We consider this is a two-value categoric variable.

2. Variable 2: Without any other information, it is a continuous variable.

3. Variable 3: Without any other information, it is a continuous variable.

4. Variable 4: Without any other information we only have the possible values: u, y, l and t.
   We consider this is a categoric variable.

5. Variable 5: Without any other information we only have the possible values: g, p and gg.
   We consider this is a categoric variable.

6. Variable 6: Without any other information we only have the possible values: c, d, cc, i, j, k, m, r, q, w, x, e, aa and ff.
   We consider this is a categoric variable.

7. Variable 7: Without any other information we only have the possible values: v, h, bb, j, n, z, dd, ff and o.
   We consider this is a categoric variable.

8. Variable 8: Without any other information, it is a continuous variable.

9. Variable 9: Without any other information, it is a binary variable. The possible values are: t and f.
   We consider this as binary variable, so we assign value 1 to "t" and 0 to "f".

10. Variable 10: Without any other information, it is a binary variable. The possible values are: t and f.
    We consider this as binary variable, so we assign value 1 to "t" and 0 to "f".

11. Variable 11: Without any other information, it is a continuous variable.

12. Variable 12: Without any other information, it is a binary variable. The possible values are: t and f.
    We consider this as binary variable, so we assign value 1 to "t" and 0 to "f".

13. Variable 13: Without any other information we only have the possible values: g, p and s.
    We consider this is a categoric variable.

14. Variable 14: Without any other information, it is a continuous variable.

15. Variable 15: Without any other information, it is a continuous variable.

CRG: German Credit Data

This is a data set available at the UCI repository:
`http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)`.

It was created by Dr. Hans Hofmann, from Institut fur Statistik und Okonometrie, in the Universitat Hamburg (Germany). This data set concerns credit card applications in this country.

There are 21 variables and the last one is the class variable. The data set has 1000 examples and doesn't have *missing information*. The class variable has only two possible values (classes), where the class 1 is the 70% of the examples and the class 2 the 30%.

The variables are:

1. Variable 1: Status of existing checking account. The possible values are:

   A11 : . . . < 0 DM ; now 2

   A12 : 0 <= . . . < 200 DM ; now 3

   A13 : . . . >= 200 DM ; now 4

   A14 : no checking account; now 1

   We redefine the variable with the changed values and consider it an ordinal variable.

2. Variable 2: Duration in months. Supposing in this field, it has to be the months to return the credit. It is a continuous variable.

3. Variable 3: Credit history. Information about previous credits. The possible values are:

A30 : no credits taken/all credits paid back duly

A31 : all credits at this bank paid back duly

A32 : existing credits paid back duly till now

A33 : delay in paying off in the past

A34 : critical account/other credits existing (not at this bank)

We define it as a categoric variable because we can not find any order.

4. Variable 4: Purpose. Why does the client want the money. The possible values are:

   A40 : car (new)

   A41 : car (used)

   A42 : furniture/equipment

   A43 : radio/television

   A44 : domestic appliances

   A45 : repairs

   A46 : education

   A47 : (vacation - does not exist?)

   A48 : retraining

   A49 : business

   A410 : others

   We define it as a categoric variable because we can not find any order.

5. Variable 5: Credit amount. It has to be the money given to the client. It is a continuous variable.

6. Variable 6: Status of existing checking account. The possible values are:

   A61 : . . . < 100 DM ; now 2

   A62 : 100 <= . . . < 500 DM ; now 3

   A63 : 500 <= . . . < 1000 DM ; now 4

   A64 : .. >= 1000 DM ; now 5

   A65 : unknown/ no savings account; now 1

   We redefine the variable with the changed values and consider it an ordinal variable.

7. Variable 7: Present employment since. The time the client has been working so far. The possible values are:

   A71 : unemployed

   A72 : . . . < 1 year

   A73 : 1 <= . . . < 4 years

   A74 : 4 <= . . . < 7 years

   A75 : . . . >= 7 years

These values are already ordered, so we consider this an ordinal variable.

8. Variable 8: Installment rate in percentage of disposable income. With the information we have, it has to be a continuous variable.

9. Variable 9: Personal status and sex. The possible values are:

   A91 : male : divorced/separated

   A92 : female : divorced/separated/married

   A93 : male : single

   A94 : male : married/widowed

   A95 : female : single

   We define it as a categoric variable because we can not find any order.

10. Variable 10: Other debtors / guarantors. Some people guarantee the client for this credit. The possible values are:

    A101 : none

    A102 : co-applicant

    A103 : guarantor

    We define it as a categoric variable because we can not find any order.

11. Variable 11: Present residence since. It seems to be the time the client has been living in his current house. It is a continuous variable.

12. Variable 12: Property. It seems to be some client's property to mortgage. The possible values are:

    A121 : real estate

    A122 : if not A121 : building society savings agreement/life insurance

    A123 : if not A121/A122 : car or other

    A124 : unknown / no property

    We define it as a categoric variable because we can not find any order.

13. Variable 13: Age in years. It is the client's age. It is a discrete variable, that is a subset of continuous variable.

14. Variable 14: Other installment plans. With the information we have, we only know the possible values:

    A141 : bank

    A142 : stores

    A143 : none

    We define it as a categoric variable because we can not find any order.

15. Variable 15: Housing. Has the client to pay for the residence where he lives? The possible values are:

A151 : rent

A152 : own

A153 : for free

We define it as a categoric variable because we can not find any order.

16. Variable 16: Number of existing credits at this bank. It is a discrete variable, that is a subset of continuous variable.

17. Variable 17: Job. Does he work? The possible values are:

    A171 : unemployed/unskilled - non-resident

    A172 : unskilled - resident

    A173 : skilled employee/official

    A174 : management/self-employed/highly qualified employee/officer

    We define it as a categoric variable because we can not find any order.

18. Variable 18: Number of people being liable to provide maintenance for. It is a discrete variable, that is a subset of continuous variable.

19. Variable 19: Telephone. Does have the client telephone number? The possible values are:

    A191 : none

    A192 : yes, registered under the customers name

    We define it as a two-value categoric variable because there are many other possibilities to incorporate here (mobile phone, fax...).

20. Variable 20: Foreign worker. This is a question with only two possible values:

    A201 : yes

    A202 : no

    It is a binary variable, where we redefine it to "A201"= 1 and "A202"= 0.

HC22/HC23: Horse Colic

This is a data set available at the UCI repository:
`http://archive.ics.uci.edu/ml/datasets/Horse+Colic`.

It was created by Mary McLeish and Matt Cecile, from the Department of Computer Science in the University of Guelph (Guelph, Ontario, Canada). It reflexs the problem of the horse colic, when they can be surgical or the consequences of the illness over the horses. So, each instance is the clinical record of a horse.

This data set has 28 variables and 5 of them could be class variables. There are 368 examples and the 30% of the values are *missing*. In this thesis we have used two problems derived from this data set.
The first one, *HC22*, uses the 23th variable as class variable. This variable has three possible values, that is, three classes. It talks about what happened to the horse and the three opcions are: it is still alive (1, 61.5%), it died (2, 24.3%) or it was euthanized (3, 14.2%).

The second case, *HC23*, uses the 24th variable as class variable. This class talks about if the problem could be surgical, so the possible values (classes) are Yes (1, 63%) or No (2, 37%). Really, in *HC22* there are two less instances because these two examples have a *missing value* in its 23th variable.

From the 28 variables, we only use 21 as predictors. Variables 3 and 28 are not used because they don't have useful information for the learning. The 23 and 24 that are used as class variables. Variables 25, 26 and 27 that are deleted because they are class variables that we don't use, but also because their information is not so explained and could be incongruous.

The variables are:

1. Variable 1: surgery?

   1. Yes, it had surgery
   2. It was treated without surgery

   We consider this as binary variable, so we assign value 1 to "Yes" and 0 to the other possible response.

2. Variable 2: Age

   1. Adult horse
   2. Young (< 6 months)

   We consider this is a two-value categoric variable. It is not binary because easily could be incorporated new possible values (elder, babies. . .).

3. Variable 4: Rectal temperature. It is a real valued variable that measures in degrees celsius the temperature of the horse.
   We consider it a continuous variable.

4. Variable 5: pulse. It is a natural valued variable, with discrete and continuous values. It represents the heart rate in beats per minute of the horse.
   We consider it a discrete variable, that is a subset of continuous variables.

5. Variable 6: respiratory rate. It is again a natural valued variable, with discrete and continuous values. As its name indicates, it represents the respiratory rate.
   We consider it a discrete variable, that is a subset of continuous variables.

6. Variable 7: temperature of extremities. It is an indicator of the peripheral circulation. Reassigning the values, this variable can be considered as an ordinal variable, because it can be ordered.

   1. Normal; now 3
   2. Warm; now 4
   3. Cool; now 2
   4. Cold; now 1

   We redefine the variable with the changed values and consider it an ordinal variable.

7. Variable 8: peripheral pulse. Reassigning the values, this variable can be considered as an ordinal variable, because it can be ordered.

   1. Normal; now 3
   2. Increased; now 4
   3. Reduced; now 2
   4. Absent; now 1

   We redefine the variable with the changed values and consider it an ordinal variable.

8. Variable 9: mucous membranes. It is a measurement of membranes colour. It can have the following values:

   1. normal pink
   2. bright pink
   3. pale pink
   4. pale cyanotic
   5. bright red / injected
   6. dark cyanotic

   We define it as a categoric variable because we can not find any order.

9. Variable 10: capillary refill time. It could have been a continuous variable, but it has been categorized:

   1. < 3 seconds
   2. >= 3 seconds

   We define it as a categoric variable. It is not binary because easily could be incorporated a new possible value (> 5 seconds, for example).

10. Variable 11: pain. It is a subjective measurement of the horse's pain. The possible values are:

    1. alert, no pain
    2. depressed
    3. intermittent mild pain
    4. intermittent severe pain
    5. continuous severe pain

    These values are already ordered, so we consider this an ordinal variable because despite the recomendation of no treating it as ordered variable, the author stablishes a clear order with "the more painful, the more likely…".

11. Variable 12: peristalsis. It indicates the activity in the horse's gut. The possible values are:

    1. hypermotile
    2. normal
    3. hypomotile
    4. absent

These values are already ordered, but in the opposite direction. So we consider this an ordinal variable.

12. Variable 13: abdominal distension. The possible values are:

    1. none
    2. slight
    3. moderate
    4. severe

    These values are already ordered, so we consider this an ordinal variable. The author emphasizes that this should be an important variable.

13. Variable 14: nasogastric tube. It indicates if it loses any gas of the tube. The possible values are:

    1. none
    2. slight
    3. significant

    These values are already ordered, so we consider this an ordinal variable.

14. Variable 15: nasogastric reflux. The possible values are:

    1. none; now 1
    2. > 1 liter; now 3
    3. < 1 liter; now 2

    We redefine the variable with the changed values and consider it an ordinal variable.

15. Variable 16: nasogastric reflux PH. It is a continuous variable with values between 0 and 14.
    We consider it a continuous variable.

16. Variable 17: rectal examination - feces. The possible values are:

    1. normal; now 3
    2. increased; now 4
    3. decreased; now 2
    4. absent; now 1

    We redefine the variable with the changed values and consider it an ordinal variable.

17. Variable 18: abdomen. Its possible values are:

    1. normal
    2. other
    3. firm feces in the large intestine
    4. distended small intestine

   5. distended large intestine

   We define it as a categoric variable because we can not find any order.

18. Variable 19: packed cell volume. It is a continuous variable with values usually between 30 and 50.
    We consider it a continuous variable.

19. Variable 20: total protein. It is a continuous variable with values usually between 6 and 7.5.
    We consider it a continuous variable.

20. Variable 21: abdominocentesis appearance. Its possible values are:

    1. clear
    2. cloudy
    3. serosanguinous

    We define it as a categoric variable because we can not find any order.

21. Variable 22: abdomcentesis total protein. It is a continuous variable.

HEP: Hepatitis

This is a data set available at the UCI repository:
`http://archive.ics.uci.edu/ml/datasets/Hepatitis`.

It was created by G.Gong from the Carnegie-Mellon University (Yugoslavia). It reflexs the problem of the hepatitis in the people that suffer it and if they can die for this reason. So each instances is the medical report of a person.

This data set has 20 variables and the first one is consider the class variable. There are 155 examples and the 5.2% of the values (162) are *missing values*. The class variable has two possible values (classes), live (79.3%) and die (20.7%), and concludes whether the person died or not.

The variables are:

1. Variable 2: Age. It is a real valued variable, with values between 0 and 80. It represents the age of the patient.
   We consider it a continuous variables.

2. Variable 3: Sex. These are the possible values:

   1. male
   2. female

   It is a binary variable because it is impossible to find a person that has any other sex.

3. Variables 4 to 14: Steroid, Antivirals, Fatigue, Malaise, Anorexia, Liver big, Liver firm, Spleen palpable, Spiders, Ascites and Varices. All of them have two possible values: 1 or "no" and, 2 or "yes". They are binary variables, where we redefine them to "yes"= 1 and "no"= 0.

4. Variable 15: Bilirubin. It is a continuous variable with values usually between 0.39 and 4.
   We consider it a continuous variable.

5. Variable 16: Alk phosphate. It is a continuous variable with values usually between 33 and 250.
   We consider it a continuous variable.
6. Variable 17: SGOT. It is a continuous variable with values usually between 13 and 500.
   We consider it a continuous variable.
7. Variable 18: Albumin. It is a continuous variable with values usually between 2 and 6.
   We consider it a continuous variable.
8. Variable 19: Protime. It is a continuous variable with values usually between 10 and 90.
   We consider it a continuous variable.
9. Variable 20: Histology. These are the possible values:
   1. no
   2. yes

   It is a binary variable, where we redefine it to “yes”= 1 and “no”= 0.

BANDS: Cylinder Bands

This is a data set available at the UCI repository:
`http://archive.ics.uci.edu/ml/datasets/Cylinder+Bands`.

It was created by Bob Evans from the RR Donnelley and Sons Co., Gallatin Division (Tennessee, USA).

This data set has 40 variables, including the class variable, but 6 of them are not used. There are 540 examples and the 4.625% of the values (999) are *missing values*. The class variable has two possible values (classes), band (42.3%) and noband (57.7%), and concludes whether there is band or not.

The deleted variables are:

Variable 1: timestamp. It is a numeric value that indicates the moment when it was reculled. It is used as a non-unique example’s code. Without learning relevance.

Variable 2: cylinder number. It is a categoric variable that indicates the cylinder number. There is a 79.4% of different values, so it can be used also as non-unique example’s code. Without learning relevance.

Variable 4: job number. It is a numeric value that indicates the job which is associated with the example. There is a 48.52% of different numbers, so it has not relevant information.

Variable 6: ink color. It is supposed to be a binary variable. In this data set it is a constant variable. Without learning relevance.

Variable 8: blade mfg. It is supposed to be a categoric variable with three different possible values (benton, daetwyler and uddeholm). In this data set, it is a constant variable. Without learning relevance.

Variable 9: cylinder division. It is supposed to be a categoric variable with three different possible values (gallatin, warsaw and mattoon). In this data set, it is a constant variable. Without learning relevance.

The used variables are:

1. Variable 3: customer. It has 71 possible values to classify the costumers. We consider it a categoric variable.
2. Variable 5: grain screened. These are two possible values: no and yes. It is a binary variable, where we redefine it to “yes”= 1 and “no”= 0.
3. Variable 7: proof on ctd ink. These are two possible values: no and yes. It is a binary variable, where we redefine it to “yes”= 1 and “no”= 0.
4. Variable 10: paper type. It has three possible values (uncoated, coated and super) that indicate the type of paper. We consider it a categoric variable.
5. Variable 11: ink type. It has three possible values (uncoated, coated and super) that indicate the type of ink. We consider it a categoric variable.
6. Variable 12: direct steam. These are two possible values: no and yes. It is a binary variable, where we redefine it to “yes”= 1 and “no”= 0.
7. Variable 13: solvent type. It has five possible values (xylol, lactol, naptha, line and other) that indicate the type of solvent. We consider it a categoric variable.
8. Variable 14: type on cylinder. These are two possible values: no and yes. It is a binary variable, where we redefine it to “yes”= 1 and “no”= 0.
9. Variable 15: press type. It has four possible values (Albert70, Motter70, Motter94 and WoodHoe70) that indicate the type of press. We consider it a categoric variable.
10. Variable 16: press. It has eight possible values (802, 813, 815, 816, 821, 824, 827 and 828) that indicate the press. We consider it an ordinal variable, because we can stablish an ordering between the values since they are numeric values.
11. Variable 17: unit number. It is a natural valued variable, with discrete values usually between 1 and 10.
    We consider it a discrete variable, that is a subset of continuous variables.
12. Variable 18: cylinder size. It has three possible values (catalog, spiegel and tabloid) that indicate the size of the cylinder. Without any other information about a possible ordering between the values, we consider it a categoric variable.
13. Variable 19: paper mill location. It has five possible values (canadian, mideuropean, northus, scandanavian and southus) that indicate where the industry that makes the paper is. We consider it a categoric variable.
14. Variable 20: plating tank. It has two possible values (1910 and 1911). It is not a binary variable because easily we could find a new possible value (1912, for instances), so we consider it an ordinal variable, because we can stablish an ordering between the values since they are numeric values.
15. Variable 21: proof cut. It is a continuous variable with values usually between 0 and 100. We consider it a continuous variable.

16. Variable 22: viscosity. It is a continuous variable with values usually between 0 and 100.
    We consider it a continuous variable.
17. Variable 23: caliper. It is a continuous variable with values usually between 0 and 1.0.
    We consider it a continuous variable.
18. Variable 24: ink temperature. It is a continuous variable with values usually between 5 and 30.
    We consider it a continuous variable.
19. Variable 25: humifity. It is a continuous variable with values usually between 5 and 120.
    We consider it a continuous variable.
20. Variable 26: roughness. It is a continuous variable with values usually between 0 and 2.
    We consider it a continuous variable.
21. Variable 27: blade pressure. It is a continuous variable with values usually between 10 and 75.
    We consider it a continuous variable.
22. Variable 28: varnish pct. It is a continuous variable with values usually between 0 and 100.
    We consider it a continuous variable.
23. Variable 29: press speed. It is a continuous variable with values usually between 0 and 4000.
    We consider it a continuous variable.
24. Variable 30: ink pct. It is a continuous variable with values usually between 0 and 100.
    We consider it a continuous variable.
25. Variable 31: solvent pct. It is a continuous variable with values usually between 0 and 100.
    We consider it a continuous variable.
26. Variable 32: ESA Voltage. It is a continuous variable with values usually between 0 and 16.
    We consider it a continuous variable.
27. Variable 33: ESA Amperage. It is a continuous variable with values usually between 0 and 10.
    We consider it a continuous variable.
28. Variable 34: wax. It is a continuous variable with values usually between 0 and 4.0.
    We consider it a continuous variable.
29. Variable 35: hardener. It is a continuous variable with values usually between 0 and 3.0.
    We consider it a continuous variable.
30. Variable 36: roller durometer. It is a continuous variable with values usually between 15 and 120.
    We consider it a continuous variable.
31. Variable 37: current density. It is a continuous variable with values usually between 20 and 50.
    We consider it a continuous variable.

32. Variable 38: anode space ratio. It is a continuous variable with values usually between 70 and 130.
    We consider it a continuous variable.

33. Variable 39: chrome content. It is a continuous variable with values usually between 80 and 120.
    We consider it a continuous variable.

HV84/HV84b: 1984 United States Congressional Voting Records

This is a data set available at the UCI repository:
http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records.

It was created by Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington D.C., 1985. It includes votes for each of the U.S. House of Representatives Congressmen.

This data set has 17 variables, including the class variable. There are 435 examples and the 5.3% of the values (392) are *missing values*. The class variable has two possible values (classes), democrats (61.38%) and republicans (38.62%), and concludes which party votes each person.

The 16 variables are the same kind of variable. They 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 export-administration-act-south-africa. All of them have two possible values: “n” and “y”. They are binary variables, where we redefine them to “y”$= 1$ and “n”$= 0$.

We have defined two problems with this data set. In the second one all the variables are used as categorical variable taking into account some people point of view that said that the *missing values* are not really *missing values*, but a third possible value. Something like “I don’t know”, “I can not answer”…

SERVO: Servo Data

This is a data set available at the UCI repository:
http://archive.ics.uci.edu/ml/datasets/Servo.

It was created by Karl Ulrich in 1986 at the Massachusetts Institute of Technology (MIT). This data set concerns an extremely non-linear phenomenon that is predicting the rise time of a servomechanism in terms of two continuous gain settings and two discrete choices of mechanical linkages.

So, it is a regression problem with 5 variables, including the class variable that vary between 0.131 and 7.1 with mean at 1.39. There are 167 instances and there is no *missing information*.

The variables are:

1. Variable 1: motor. Information about which motor has been used. The possible values are $\in \{A,B,C,D,E\}$, so we define it as a categoric variable because we can not stablish any order.

2. Variable 2: screw. Information about which screws have been used. The possible values are $\in \{A,B,C,D,E\}$, so we define it as a categoric variable because we can not stablish any order.

3. Variable 3: pgain. It is a kind of gain, so it is a numerical value $\in \{3,4,5,6\}$. Despite of being a set of four possible values, an expert has pointed out that it should be considered an continuous variable.

4. Variable 4: vgain. It is a kind of gain, so it is a numerical value $\in \{1,2,3,4,5\}$. Despite of being a set of five possible values, an expert has pointed out that it should be considered an continuous variable.

PRO: Prostate

This is a data set available at the *lasso2* library of R:
`http://www.biostat.jhsph.edu/~ririzarr/Teaching/649/prostate.html`.

This data set come from a study that examined the correlation between the level of prostate specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy.

It is a regression problem with 9 variables, including the class variable (*lpsa*, logarithm of the prostate specific antigen) that vary between -0.431 and 5.583 with mean at 2.478. There are 97 instances and there is no *missing information*.

The variables are:

1. Variable 1: lcavol. Logarithm of the cancer volume. It is a logarithm, so it is a real number. We deal with it using the continuous function.

2. Variable 2: lweight. Logarithm of the postate weight. It is a logarithm, so it is a real number. We deal with it using the continuous function.

3. Variable 3: Age. It is the patient's age. It is a discrete variable, that is a subset of continuous variable. We deal with it using the continuous function.

4. Variable 4: lbph. Logarithm of the benign prostatic hyperplasia amount. It is a logarithm, so it is a real number. We deal with it using the continuous function.

5. Variable 5: svi. Seminal vesicle invasion. It is a two-valued variable $\in \{0,1\}$, and an expert has pointed out that it can be considered a binary variable.

6. Variable 6: lcp. Logarithm of the capsular penetration. It is a logarithm, so it is a real number. We deal with it using the continuous function.

7. Variable 7: gleason. Gleason score. It is a categoric ordered variable and it has four possible values $\in \{6,7,8,9\}$, and an expert has pointed out that it can be considered an ordinal variable. We consider the order of the values is defined by its numerical value, so $6 < 7 < 8 < 9$.

8. Variable 8: pgg45. Percentage Gleason scores 4 or 5. It is a percentage, so it is a real number $\in [0,100]$. We deal with it using the continuous function.

D

Using the *Heterogeneous Neural Network 2*

This a short guide about how to use the algorithm we have developed in R lenguage. It is divided in two sections and in the first one you will find all about the *Leader2* use. In the second section you will find the same for *Heterogeneous Neural Network 2*.

D.A *Leader2*

The *Leader2* version we have developed in this research project has some characteristics that we will explain here. Let's start with the customizable functions that allow users to define their own proposals working with this algorithm.

If you want to incorporate a new partial similarity function, you only has to define these two functions:

```
simFunction <- function(a, b, P)
```

```
simFunction_param <- function(data)
```

where *simFunction* is the name of the function, *a* and *b* are the values to be compared, if the function requires some parameter it is stored at *P* and, finally, *data* is a vector with all the values of this variable in the data set.
The first function is the similarity function itself, so it has to return an only value in $[0,1]$ that indicates how similar are both values (a, b). It can return a *missing value*.
The second function, the one finished in *param*, is a function called by the preprocess function to calculate a default value for the parameter *P*, if it exists and it has not been indicated by the user. So, the function has to return the parameter *P* that will be used after by its partial similarity function.

As we said in the document, the aggregator can be redefined by users. It has to follow this structure:

```
aggregator <- function(inst1, inst2, obj, calc =
              1:length(inst1))
```

where *aggregator* is the function name, *inst1* and *inst2* are the instances to be compared, *obj* is the *Leader2* configuration (it is stablished in the preprocess function), and *calc* indicates the variables that have to be aggregated (by default, all of them).
It calls the partial similarity functions and combines their values to return an only global similarity value. It can not return *missing values*, so an strategy to replace them has to be used. In our default aggregator this choice is customizable also, so users can define their own proposal using:

```
fillNA <- function(vect, obj)
```

where *fillNA* is the name of the function, *vect* is the partial similarity measurements (some of them could be *missing values*) and *obj* is the *Leader2* configuration. It has to return the same vect without *missing values*.

The *Leader2* configuration we have refered above is set in the *preprocess* function:

```
preprocess <- function(data, methods = NULL, types =
                NULL, param = NULL, q = 1, methodsDefault
                = MD, calcSk = TRUE, cjts = list(),
                treatNA = “fill.zerofive”, aggreg =
                “aggregatePartials”)
```

where *data* is a data frame structure with the data set (without class variable). The rest of parameters are:

methods It is a vector with the similarity functions chosen for each variable. By default, it has no value, so the default partial similarity function is used: *continuous* 4.8.

types It is a vector with the types of each variable. By default, it has no value, so the default type is used: *continuous*.

param It is a list with the parameter *P* of the partial similarity functions chosen for each variable. By default, it has no value, so it is calculated using the *param* functions that we saw above.

methodsDefault It is a matrix with the default partial similarity functions asociated with each data type. That is used when *types* are specificated but not *methods*. It is set to *MD*, that we have defined as:

```
MD <- matrix("", 0, 2)
MD <- rbind(MD, c("binary", "binary"))
MD <- rbind(MD, c("categoric", "overlap"))
MD <- rbind(MD, c("ordinal", "probabilistic"))
MD <- rbind(MD, c("continuous", "continuous"))
MD <- rbind(MD, c("discrete", "continuous"))
MD <- rbind(MD, c("fuzzy", "fuzzy.zerofive"))
```

That can be modified using:

```
MD[i,] <- c("dataType", "defaulfSimFunction")
```

Or a new structure can be generating following:

```
newMD <- matrix("", 0, 2)
newMD <- rbind(newMD, c("dataType", "defaulfSimFunction"))
```

where *newMD* is the name of the structure, *dataType* is each one of the data types the user wants to define and *defaulfSimFunction* is its associated partial similarity function.

calcSk It is a boolean value that defines if normalization is used (only for default aggregator). By default, it is *TRUE*.

q It is the *q* value of the aggregator. It is a numeric value, by default 1.

treatNA It is the method to replace missing values. By default, it is *fill.zerofive*.

cjts It is a list with secundary aggregations. It has to be defined as:

```
cjts <- list()
cjts[[i]] <- list()
cjts[[i]]$q
cjts[[i]]$indexs
cjts[[i]]$funct
cjts[[i]]$treatNA
```

where *q* is the *q* value of the aggregator, *indexs* are the variables to be aggregated, *funct* is the aggregator to be used and *treatNA* is the method to replace *missing values*.
Note that this partial aggregations only works in our aggregator.

aggreg It is the aggregator to be used. By default, it is *aggregatePartials*, our proposal.

This preprocess function returns a list with the *Leader2* configuration.
Then, we have the *Leader2* functions themselves. They are:

```
leader2 <- function(data, Smin = 0.5, L = NA, ...)
```

```
leader2.obj <- function(data, Smin = 0.5, L = NA, obj =
               NULL)
```

where *data* is the data set without class variable, *Smin* is the main parameter in our method and *L* says how many instances are required to create a new cluster.
Both functions are the same, but the first one requires also (...) the preprocess function parameters because it calls the preprocess before performing and the second function assumes you have called preprocess function before and you have to pass the *Leader2* configuration in *obj*.
But we have also the supervised versions, which are:

```
leader2.supervised <- function(data, T, Smin = 0.5, L = NA, ...)
```

```
leader2.supervised.obj <- function(data, T, Smin = 0.5, L = NA, obj
                                   = NULL)
```

where all the parameters are the same but incorporating the class variable in *data* and indicating it with the parameter *T*.

Despite of having different names, all these versions work in the same way and all of them return:

nclusters The number of clusters created.

Smin The s_{min} used.

clusters It indicates the instances that work as leaders in the clusters.

assign It indicates which instances belong to each cluster.

obj The *Leader2* configuration.

sim The similarity measurements of each instance with its leader.

rec The order in which the instances of the data set have been chosen.

MSI The similarity measurements of the instances with the leaders.

D.B *Heterogeneous Neural Network 2*

Until now we have seen how to use *Leader2* clustering algorithm, which initializes *HNN2* first layer. Next, we are going to specify the main characteristics to be set up in order to use this algorithm. Let's start again with the customizable functions that allow users to define their own proposals working with *HNN2*.

If you want to incorporate a new behaviour for the width parameter (p/σ), you only has to define these three functions:

```
funct_activation <- function(x, sigma)
```

The activation function, where *x* is the input of the neuron and *sigma* is its width parameter.

```
heuristic_sigma <- function(clust, nSigmas = TRUE)
```

A heuristic to calculate the width parameter, where *clust* is the result of the algorithm that initializes the first layer and *nSigmas* indicates whether several p/σ have to be generated or not.

```
recalculate_sigma <- function(sigma, X, W, WLamb, T, nSigmas,
                          eps, nit)
```

where *sigma* is the width parameter that is a single number or a vector, depending on the *nSigmas*, a boolean that indicates whether several p/σ have to be used or not. Here, *X* is the training set without the class variable, *T* is a vector with the class variable, *W* are the second layer weights and *WLamb* is the component of the error that penalizes the high weights. There are two more parameters related with the optimization itself: *eps* that indicates when convergence is achieved and *nit* that indicates the maximum number of iterations if convergence is not achieved. This function represents the second optimization of the *Alternate Optimization*. This last function optimizes the functions:

```
fErr <- function(sigma, X, W, WLamb, T)
```

```
fdErr <- function(sigma, X, W, WLamb, T)
```

the error function and its derivative in terms of training data.

Note that all these functions can use their paremeters and several global variables: *EPS* that stores the default epsilon, *G* that stores the resemblance between instances and clusters and *Gc* that stores the resemblance between clusters.

The *Heterogeneous Neural Network 2* can be called using some different functioncs:

```
HNN2.linear_regression <- function(data, T, ..., TReg = "matrix",
                          epsilon = EPS, limItG = 10, limIt = 100,
                          findSigma = "findP", fAct = "functFP",
                          nSigmas = TRUE, rSigmas = FALSE, fRSigmas
                          = "recalculateP", iniLambdas = c(1e-6,
                          1e-3, 1))
```

```
HNN2.classification <- function(data, T, ..., TReg = "matrix",
                       epsilon = EPS, limItG = 10, limIt = 100,
                       findSigma = "findP", fAct = "functFP",
                       nSigmas = TRUE, rSigmas = FALSE, fRSigmas
                       = "recalculateP", iniLambdas = c(1e-6,
                       1e-3, 1))
```

where *data* is the data set, *T* indicates the class variable and ... are the parameters to call the *Leader2*. Then *TReg* chooses the method to be used ("matrix" or "svd"), *epsilon* indicates when convergence is achieved and *limIt* and *limItG* indicate the limits of maximum iterations for the inner and the outer loop of the *AO*. Then, *fAct* is the activation function, *findSigma* is the heuristic function to calculate the width parameter, *fRSigmas* indicates the function to recalculate this parameter. *nSigmas* and *rSigmas* are two boolean values that indicate if an unique width parameter has to be used and if it recalculation has to be performed. Finally, *iniLambdas* is a vector with the different values to test as initial values of λ.

All is configured to perform as our default *HNN2* proposal.

When the clustering is performed before you can use its results to call *HNN2* using these functions:

```
HNN2.linear_regression.clust <- function(data, T, clust = NULL, TReg
                                = "matrix", epsilon = EPS, limItG = 10,
                                limIt = 100, findSigma = "findP", fAct
                                = "functFP", nSigmas = TRUE, rSigmas
                                = FALSE, fRSigmas = "recalculateP",
                                iniLambdas = c(1e-6, 1e-3, 1))
```

```
HNN2.classification.clust <- function(data, T, clust = NULL, TReg
                             = "matrix", epsilon = EPS, limItG = 10,
                             limIt = 100, findSigma = "findP", fAct
                             = "functFP", nSigmas = TRUE, rSigmas
                             = FALSE, fRSigmas = "recalculateP",
                             iniLambdas = c(1e-6, 1e-3, 1))
```

where there are no parameters for *Leader2* and *clust* is the previous called *Leader2* result.

The same happens with the *RBF* network version that we have developed paralelly to *HNN2*. It has two functions also:

```
RBF2 <- function(data, T, ..., TReg = "matrix",
        epsilon = EPS, limItG = 10, limIt = 100,
        findSigma = "findHeu1Sigma", fAct =
        "functRBF", nSigmas = TRUE, rSigmas =
        FALSE, fRSigmas = "recalculateSigma",
        iniLambdas = c(1e-6, 1e-3, 1),
        problem="regression")
```

```
RBF2.clust <- function(data, T, clust = NULL,
              TReg = "matrix", epsilon = EPS,
              limItG = 10, limIt = 100, findSigma =
              "findHeu1Sigma", fAct = "functRBF",
              nSigmas = TRUE, rSigmas = FALSE, fRSigmas
              = "recalculateSigma", iniLambdas =
              c(1e-6, 1e-3, 1), problem="regression")
```

They are the same functions but using a different configuration that tries the *RBF* settings. It also incorporates the *problem* parameter, that informs if it is a regression problem or a classification one.

Despite of having different names, all these versions work in the same way and all of them return:

nHN The number of hidden neurons (number of clusters).

nOH The number of output neurons.

alphas Second layer weights.

sigma Width parameter (p/σ).

obj The *Leader2* configuration.

fAct Activation function used.

error Training error.

lambdas Final lambdas.

Once the *HNN2* has been trained, we can test it using the following two functions:

```
ferror <- function(Xt,T,net)
```

```
ferrorNorm <- function(Xt,T,net)
```

where *Xt* is the test set, *T* indicates the class variable and *net* is the *HNN2* that returns the training. The only difference is that the second function returns also a normalized error, when the first one returns the accuracy (only for classification problems) and the quadratic error of the model in *net* with data in *Xt*.