Diplomarbeit

Konzeption und Implementierung eines Neuro-Fuzzy-Datenanalysetools in Java

(Design and Implementation of a Neuro-Fuzzy Data Analysis Tool in Java)

von

cand. inform. Ulrike Nauck

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Aufgabenstellung und Betreuung:
Prof. Dr. R. Kruse

Braunschweig
03.01.1999

Kurzfassung

In dieser Diplomarbeit wird NEFCLASS-J vorgestellt, eine Implementierung des NEFCLASS-Modells. NEFCLASS ist ein Neuro-Fuzzy-Ansatz zur Klassifikation von Daten. NEFCLASS-J wurde in der Programmiersprache Java entwickelt, die auf objekt-orientierten Konzepten beruht und die Unabhängigkeit von einer bestimmten Plattform garantiert. Das Neuro-Fuzzy-Modell NEFCLASS umfaßt Lernverfahren zur Bestimmung der Struktur (Regelbasis) und der Parameter (Fuzzy-Mengen) eines Fuzzy-Klassifikators auf der Grundlage von Daten. Das Ziel des Ansatzes besteht darin, interpretierbare Klassifikatoren zu erzeugen. Daher werden die Lernverfahren mittels Constraints eingeschränkt, die je nach den Bedürfnissen der in Betracht gezogenen Anwendung ausgewählt werden können. NEFCLASS-J ist eine Erweiterung vorangegangener Implementierungen des NEFCLASS-Modells für MS-DOS PC bzw. Unix-Workstations. Die neuen Funktionen des Softwarewerkzeuges umfassen Batch Learning, automatische Kreuzvalidierung, automatische Bestimmung des Umfangs der Regelbasis, Behandlung fehlender Werte und automatische Reduktion (pruning) des Klassifikators, um seine Interpretierbarkeit zu erhöhen. Weiterhin wurde eine vollständig neue Bedienoberfläche entwickelt, die sich in Aussehen und Bedienung an Standardsoftware anlehnt.

Abstract

In this thesis NEFCLASS-J, a new implementation of NEFCLASS - a neuro fuzzy approach for classification of data, is presented. NEFCLASS-J is written in Java, a platform-independent object-oriented programming language.The neuro-fuzzy classification model NEFCLASS offers learning algorithms to create the structure (rule base) and the parameters (fuzzy sets) of a fuzzy classifier from data. The aim of the NEFCLASS approach is to create interpretable classifiers. Therefore the learning algorithms use contraints which can be selected according to the requirements of the application of interest. NEFCLASS-J extends previous implementations of the NEFCLASS model for MS-DOS PC or Unix workstations. The new features offered by the new tool are batch learning, automatic cross validation, automatic determination of the rule base size, handling of missing values, and automatic pruning of a classifer to reduce its size and to increase its interpretability. In addition a complete new graphical user interface was developed that provides a look and feel of standard software applications.

Contents

Introduction 1

1 Neuro-Fuzzy Systems 3

1.1 Classification with Fuzzy Systems 4
1.1.1 Fuzzy Sets 5
1.1.2 Fuzzy Rules 7

1.2 Neural Networks 8
1.2.1 A Generic Model of Neural Networks 9
1.2.2 The Multilayer Perceptron 12
1.2.3 The Backpropagation Algorithm 14

1.3 Neuro-Fuzzy Systems 17

2 The NEFCLASS Model 22

2.1 The Structure of the NEFCLASS Model 22

2.2 Learning a Rule Base - The Algorithm 24

2.3 Training Fuzzy Sets - The Algorithm 27

2.4 Pruning the Rule Base 31

3 Java for NEFCLASS 33

3.1 The NEFCLASS-Tool 33

3.2 The Java Language 34

4 Experiments with NEFCLASS-J 36

4.1 The Wisconsin Brest Cancer (WBC) Data Set 37

4.2 The Experiment 39
4.2.1 Automatically Create a Classifier and then Prune It 39
4.2.2 Use a Certain Number of Rules to Create the Classifier 43
4.2.3 Cross Validation of the Classifier 47

4.3 Comparison of Different Approaches 49

4.4 Conclusion 51

5 How to Use NEFCLASS-J 52
5.1 The Training Data Set 52
5.2 Guided Tour through NEFCLASS-J 53
5.2.1 The Philosophy 57
5.2.2 Create a Project 59

5.2.4 Use Prior Knowledge 81
5.2.5 Pruning the Rule Base 83
5.3 Strategies to Create a Classifier 84
5.4 The Main Menue 84
5.5 The Files of a NEFCLASS Project 89
5.5.1 The Data File 89
5.5.2 The Parameter File 91
5.5.3 The Network File 91
5.5.4 The Training Protocol File 91
5.5.5 The Result File 92

6 The Structure of NEFCLASS-J 93
6.1 The Graphical User Interface 93
6.2 The Data Structure of the NEFCLASS Model 96
6.3 Training Data 99
6.4 Utility Classes 99

7 Conclusion 100

References 102

Introduction

In a time of world wide computerisation and data collection not only computer scientists have the problem to handle data bases. Simultaneously the variety of problems in data analysis increase. The traditional statistical methods are powerfull, but they are often based on assumptions that do not hold for the real world data and the results can be hard to interpret. They come up with a high precision which may not be necessary in any case but can cost a lot. Furthermore there is need for fundamental mathematical knowledge to use these approaches. There are many questions in the scientific and economical world that are wished to be answered with low costs and sufficient precision.

The kind of questions this thesis deals with are classification problems. Classification of customer data, for instance, is an important data analysis problem for modern companies. It is, for example, necessary to know if a customer may be interested in a certain mailing, if certain services should be offered to him, or if there is a danger that he might cancel the contract. The task is to develop a classifier out of a data file. The development-tool should be easy to understand and to use, the development of the classifier should not cost too much effort and the validation of the classifier should be possible. Furthermore the classifier should be interpretable, because it is not only the question to which class a pattern belongs, but also why it belongs to it.

There are *soft computing* methods which take these demands into consideration. The term soft computing was coined by Lotfi A. Zadeh, the founder of fuzzy logic. Soft computing includes approaches to human reasoning that try to make use of the human tolerance for incompleteness, uncertainty, imprecision and fuzziness in decision making processes [NAUCK ET AL 97].

The model used here is the *neuro-fuzzy system*. These systems combine the learning ability of neural networks and the ability of fuzzy systems to handle linguistic rules. These are qualities needed to develop a tool which is able to create a classifier out of data by learning, and on the other hand is able to ensure a certain interpretability with the help of linguistic rules. Furthermore, prior knowledge can be inserted in form of linguistic rules. The neuro-fuzzy approach is described in the first chapter.

The resulting development-tool is NEFCLASS-J, which is short for NEuro Fuzzy CLASSification programmed in Java. There are previous tools like NEFCLASS-X [HOFERICHTER 96 / BODE 97], a tool for Unix systems and NEFCLASS-PC [UNAUCK 97], a system developed in Pascal convenient for MS-DOS/Windows PC. NEFCLASS-J can be seen as an extension of NEFCLASS-PC, but there are some completely new features and the graphical interface changed completely. It is adapted now to the graphic interfaces of Windows and Unix X-Terminals. In Chapter 2 the NEFCLASS model is described on which this tool is based.

In order to get a platform independent NEFCLASS-Tool Java was chosen as the programing language. In Chapter 3 the existing NEFCLASS-Tools are compared to NEFCLASS-J and the benefits of the Java language and the development kit are depicted.

In Chapter 4 some experiments to test the performance of the NEFCLASS-J algorithms are described. For the experiments the 'Wisconsin brest cancer' (WBC) data set is used. A classifier is created where the size of the rule base is determined automatically. This results in a rule base that covers all patterns of the data set. However, this rule base consists of a large number of rules. To reduce the size of the rule base in order to get a more interpretable classifier, an automatic pruning procedure is used. After a promising rule base is found the cross validation procedure allows statements about the quality of the classifier. The results of these experiments are compared to the results of other classifiers.

In Chapter 5 NEFCLASS-J is described in detail. In some instance the philosophy of the tool differs from that of NEFCLASS-PC so that some features may not be recognized by users of the former tool. So references to NEFCLASS-PC are given when necessary. The features are introduced in form of a tutorial. In a guided tour it can be learned that NEFCLASS-J is very easy to use for beginners, because suitable default values are set. After developing a feeling for the data and the tool, changing parameters, editing and pruning the rule base is trained. Furthermore the files created by the NEFCLASS tool are described.

The program structure of NEFCLASS-J is described in Chapter 6. The classes of the graphical user interface, the NEFCLASS model, the training data and utility classes are described and the hierarchy of classes is visualized.

The thesis is completed with the conclusion in Chapter 7.

NEFCLASS-J can be obtained
- by anonymous ftp at fuzzy.cs.uni-magdeburg.de in /pub/nefclass
- via World Wide Web at http://fuzzy.cs.uni-magdeburg.de/nefclass

1

Neuro-Fuzzy Systems

Neuro-fuzzy systems are a combiniation of *neural networks* and *fuzzy systems*. These two models have at first their place in independent areas. The connections to each other are merely marginal but we will see that a combination of both brings benefit for the solution of many problems.

It was the theory of fuzzy sets with which Lotfi A. Zadeh founded 1965 a computational approach to human thinking and behavior [ZADEH 65]. Terms like big, small or approximately are named *fuzzy*. They are sufficient for communication in daily life but not for processing with a machine. It is the model of fuzzy systems which offers a computational representation of these terms. First these systems were successfull in solving control problems. Up to now fuzzy systems are not only used for control problems but also used in many other areas. In this thesis they are used for classification.

Experts can do their job without calculating a mathematical model. They normally get the experience by solving problems. The idea of fuzzy systems is to let the expert specify his actions in form of linguistic rules. These rules are translated into a framework of fuzzy set theory providing a calculus which can simulate the behavior of the expert. The translation into fuzzy set theory is not formalized and arbitrary choices concerning, for example, the shape of membership functions can be made. These uncertainties in the process of building a fuzzy system mostly result in a heuristic tuning process to overcome the initial design errors [NAUCK ET AL 97].

Neural networks are systems that try to make use of some of the known or expected organizing principles of the human brain. They consist of a number of independent, simple processors - the neurons. These neurons communicate with each other via connected weights - the synaptic weights. At first , research in this area was driven bei neurobiological interest but here we will restrict ourselves to the problem of information processing , and do not consider biological aspects. Neural networks can solve difficult problems by using a learning procedure that depends on the neural model and the given problem, but they are black boxes. It is not possible to determine how the solution was found. Furthermore it is not possible to insert prior knowledge to a neural network [NAUCK ET AL 97].

The combination of the learning ability of neural networks and the linguistic rule handling of fuzzy systems results in a self-tuning neuro-fuzzy system which is interpretable and in which prior knowledge can be inserted.

1.1 Classification with Fuzzy Systems

The process of creating a model for the behavior of a human experts is called *cognitive analysis* [KRUSE ET AL. 95]. As just mentioned a method of modelling the behavior of an expert is to compile his knowledge in linguistic rules. As an example the classification of iris flowers is used. There are three classes of iris flowers (Setosa, Virginica, Versicolor). They can be identified with the help of the different length an width of the petal and sepal. So there are four independant variables that form the antecedent of the rule, and three dependent variables that form the consequent of the rule. So the rules are of the form:

if *petal length* is *small* and
petal width is *medium* and
sepal length is *small* and
sepal lenght is *small* **then** class is *Setosa*

The problem now is to describe what small or medium means. Two experts discussing about iris flowers would intuitively know it, but for a computational model these terms must be defined precisely, however, if possible without loosing the vague nature of the terms.

Before we show how this can be done with the help of the fuzzy set concept it should be stressed that fuzzy classifiers are not a replacement for methods like statistics or other forms of machine learning and it does not yield better results. Fuzzy classification offers a different way to achieve the same goal. If a decision is made for a fuzzy classifier usually the following advantages are considered:

- vague knowledge can be used,
- the classifier is interpretable in form of linguistic rules,
- from an applicational point of view the classifier is easy to implement, to use and understand.

But it should also be remembered that fuzzy classifiers that use rules like the one given above are only useful, if direct dependencies of features to class informations are to be modeled. It is implicitly assumed that the features itself are independent from each other. If more complex domains must be modeled, where dependencies or conditional dependencies between all variables of a given problem have to be explicitly represented, then graphical models (dependency networks) like, for instance, Bayesian networks [KRUSE ET AL. 91] or possibilistic networks [KRUSE ET AL. 94] should be preferred. Fuzzy classifiers can be viewed as an alternative to neural networks, regression models, or nearest neighbour classifiers, i the abover-mentioned advantages are of interest [NAUCK/KRUSE 98].

1.1.1 Fuzzy Sets

Fuzzy sets can be used to model terms of every-day language. To define that a person is grey-haired if there are 10.000 grey hairs can not mean that a person with 9.999 grey hairs has his natural color.

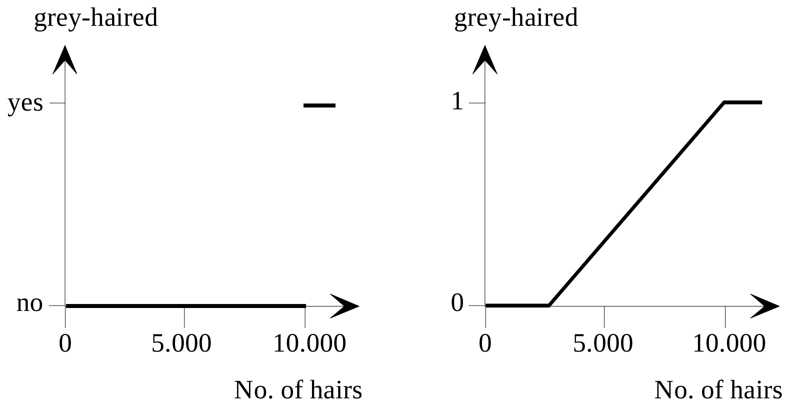


Figure 1.1: In the left figure the crisp and in the right figure the vague modeling of the term *grey-haired* can be seen

In the fuzzy set theory those terms are modeled through vague sets. The whole domain that differs from natural color to grey-haired has to be partitioned into fuzzy sets which can even overlap. An object in the crisp definition is a member of a class or it is not. In the fuzzy set theory it has a membership degree where 1 means full membership and 0 means no membership. A fuzzy set is given by a *membership function* $\mu: \mathbb{R} \rightarrow [0, 1]$ which can be seen as a generalization of the characteristic function of a normal (crisp) set.

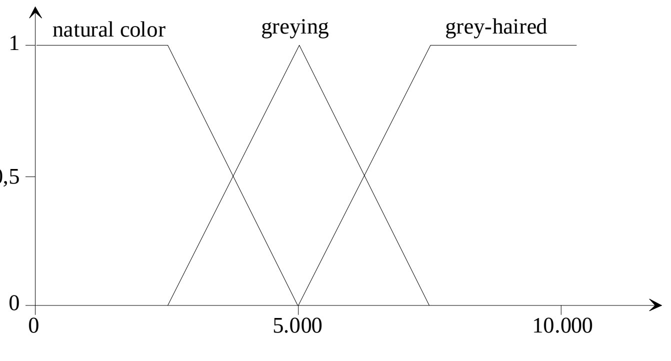


Figure 1.2: A possible fuzzy-partition of the domain ‘greying’

There are three kind of functions that are commonly used to represent fuzzy sets, because they are defined by only a few parameters and they have useful features like convexity and normality [Kruse et al. 94]. These are triangular, trapezoidal and bell-shaped functions.

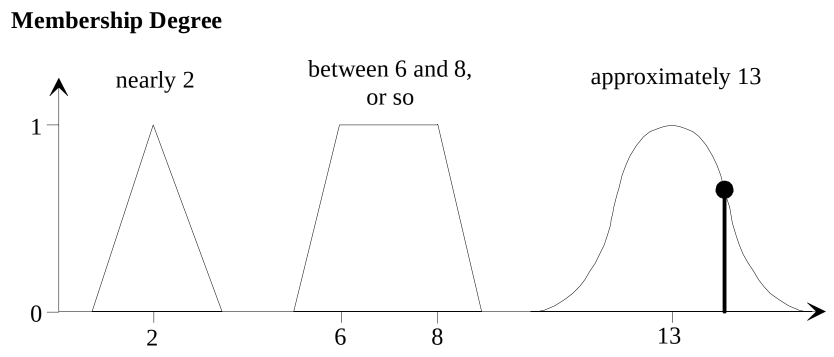


Figure 1.3: Types of fuzzy sets

The next task is to find operators for the intersection and union of fuzzy sets. Functions that fulfil minimum requirements for the intersection operator are called *t-norms*. The *t-conorm* is the dual of the t-norm and t-conorms are the basis for the definition of union operators [KRUSE ET AL 94]. For the NEFCLASS-model the minimum is used:

$$\top_{min}(a, b) = \min \{a, b\}.$$

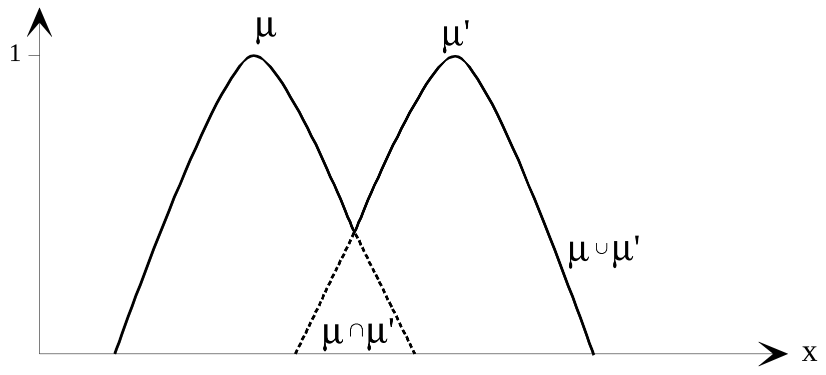


Figure 1.4: Intersection and union of two fuzzy sets

With help of the fuzzy sets as depicted above it is now possible to create fuzzy rules which can be used as a basic construct to model expert knowledge. The intersection operator is used to model the *and* connective and the union operator models the *or* connective.

1.1.2 Fuzzy Rules

Fuzzy rules for solving classification problems haqve the following general form:

$$\mathrm{R_r}: \quad \textbf{If} \;\; x_1 \text{ is } A^{(1)}_{j_1} \text{ and } \ldots \text{ and } x_n \text{ is } A^{(n)}_{j_n} \quad \textbf{then} \;\; (x_1, x_2, \ldots, x_n) \in C_j \,.$$

Where $A^{(1)}_{j_1}, \ldots, A^{(n)}_{j_n}$ are linguistic terms, which are represented by the fuzzy sets $\mu^{(1)}_{j_1}, \ldots, \mu^{(n)}_{j_n}$. $C_j \subseteq \mathbb{R}^n$ is a pattern subset and represents the class *j*. The patterns are input tuples $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. It is assumed that they can be devided into disjunct classes, so that every pattern can be related to a class C_j. Every value x_i of the input tuple is to be partitioned by q_i fuzzy sets $\mu^{(i)}_1, \ldots, \mu^{(i)}_{q_i}$. The classification is defined by a rule base with *k* fuzzy rules $R_1, \ldots, R_k$.

In the beginning of this chapter the example of the iris flower was given. The data set consists of patterns with four input parameters (x_1, x_2, x_3, x_4) given by length and width of the petal and sepal. The domain of every parameter x_i is represented by three fuzzy sets (small = sm, medium = md, large = lg). There are three possible classes (Setosa, Virginica, Versicolor). Of course, different definitions are possible but this depends on the view of the expert. The example from the beginning is now:

if x_1 is *sm* and x_2 is *md* and x_3 is *sm* and x_4 is *sm* **then** *Setosa*

Figure 1.5 shows a graphical representation of a rule base with *k* rules an *n* input parameters.

In contrast to fuzzy control rules where the consequent of a rule is a fuzzy set, a fuzzy classification rule results in a crisp value which is assigned to a class. This value is computed by the t-norm $\top_{\min}$, while the classification result processed over all rules is given by the t-conorm $\perp_{\max}$.

This all seems to be very easy and clear but it should be remembered that for every task a representation has to be found which is not obvious. The choice of the membership functions and the partitioning of the domain are only two problems. The fuzzy sets need not to be symmetric or regularly distributed over the domain.

And there is always an optimisation problem. The more fuzzy sets are used to partition the domains of the variables, the more fuzzy rules can be constructed. Thus the classification can be made arbitrarily exact. On the other hand a fuzzy system with a large rule base is a slow and confusing system. A small number of rules can be leveled out with non-symmetric fuzzy sets. But this reduces the interpretability of the system. So there is the dual problem of performance and interpretability.

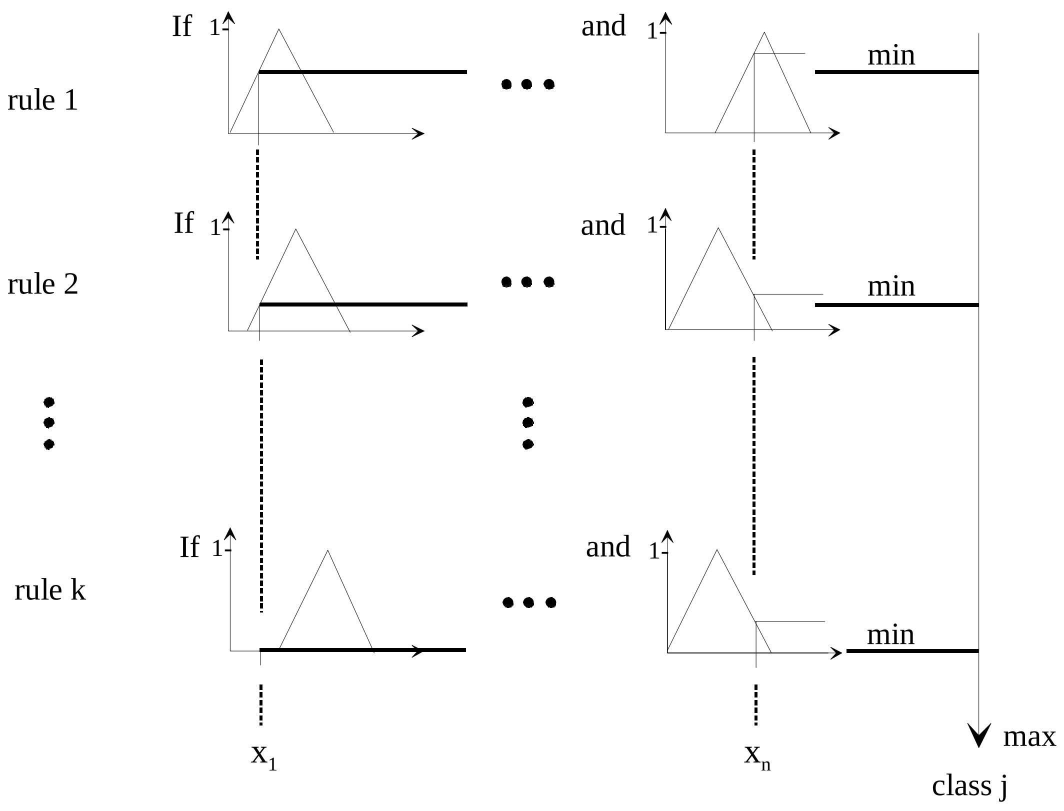


Figure 1.5: Fuzzy rule base and evaluation with minimum and maximum operator

The NEFCLASS-Model adresses these problems. The fuzzy rule base is directly learned from the data. It is possible but not necessary to insert prior knowledge in form of fuzzy rules. Afterwards the fuzzy sets are learned. This cannot solve the described optimization problem completely but it can ease it. Depending on the data constraints given by the user of the system are considered. He has to decide if the classifier should be tuned to a near zero error result or if the priority is on the interpretability of the system and a certain error can be accepted.

1.2 Neural Networks

Research on artificial neural networks started around 1940 and was inspired by interest in the neurophysiological fundamentals of the human brain. It was known that the brain consists of interconnected nerve cells - the neurons - that influence each other by electrical signals. A neuron conducts its signals via its axon that projects from its cell body (soma) [NAUCK ET AL 97]. It receives signals from other neurons over its dentrites which scan the axons of these neurons for signals. The joints of dendrites and axon are named synapses. This is a very little gap where chemical transmitters are activated if the signal of the axon is strong enough. That means that the collection of incomming signals into the neuron reach a certain value. Some synapses inhibit the signal others pass the signal through. This is the way of filtering signals.

It cannot be the task to develop an artificial brain. On one hand there is too little biological knowledge to create a successful model and on the other hand the computer recources are to small to compute only a fraction of the millions and millions of a brain's neurons. So we will restrict ourselves to the problem of information processing, and do not consider biological aspects.

Now a generic model is introduced which lays the foundation of all models for neural networks [NAUCK ET AL. 97]. After this the model used for the NEFCLASS system is depicted, the multilayer perceptron.

1.2.1 A Generic Model of Neural Networks

As depicted in [NAUCK ET AL. 97] an artificial neural network can be generally seen as a formal structure that can be easily described by a set and a few mappings.

Definition 1.1 *A neural network is a tuple* $(U, W, A, O, \mathrm{NET}, \mathrm{ex})$*, where:*

(i) *U is a finite set of processing units (neurons),*

(ii) *W, the network structure, is a mapping from the cartesian product to the set of real values* $W : U \times U \to \mathbb{R}$,

(iii) *A is a mapping that assigns an activation function* $A_u\colon \mathbb{R}^3 \to \mathbb{R}$ to each $u \in U$,

(iv) *O is a mapping that assigns an output function* $O_u\colon \mathbb{R} \to \mathbb{R}$ to each $\mathrm{u} \in U$,

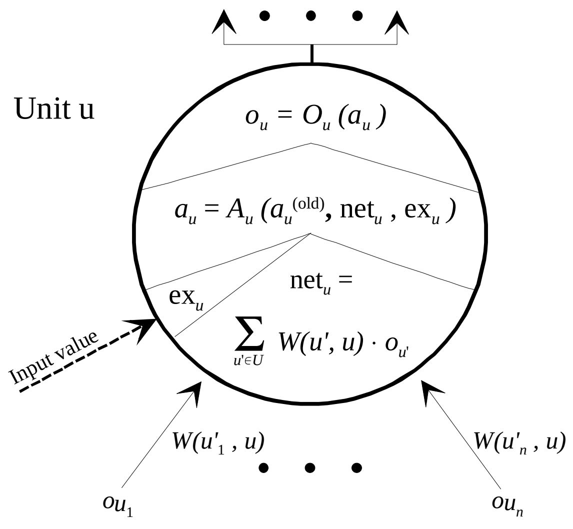
(v) NET is a mapping that assigns a network input function $\mathrm{NET}_u\colon (\mathbb{R} \times \mathbb{R})^U \to \mathbb{R}$ to each $u \in U$, and

(vi) ex *is an internal input function* $\mathrm{ex}\colon U \to \mathbb{R}$, *that assigns to each* $u \in U$ *an external input in the form of a real value* $\mathrm{ex}_u = \mathrm{ex}(u) \in \mathbb{R}$.

This definition describes the static features of a neural network in such a general way that it is a basis for all kind of neural network models. In the following a detailed explanation of the components and parameters is given according to [NAUCK ET AL. 97].

The Processing Units

The *processing units* or *neurons* of a neural network can be viewed as simple processors. Depending on their current state and their current input - both given by real values - they produce an output value and assume a new state. The units process their input values in parallel and independenly of each other. In most neural network models there is one input layer, one or more hidden layers and one output layer. The units of the input and output layers are used to communicate with the environment of the network. The state of the hidden units cannot be directly influenced or observed from the environment.



Unit u
$o_u = O_u(a_u)$
$a_u = A_u(a_u^{(old)}, net_u, ex_u)$
ex_u
$net_u = \sum_{u' \in U} W(u', u) \cdot o_{u'}$
Input value
$W(u'_1, u)$
$W(u'_n, u)$
ou_1
ou_n

Figure 1.6: A processing unit *u* of a neural network

The Network Structure

The mapping *W* is usually called the *network structure*. It can be displayed as a weighted directed graph, where the nodes represent network units, and the weighted edges are weighted communication links. The network structure is the basis of the communication of the processing units. The output of one unit becomes the input of other units.The weighted connections enable each unit to consider the output values of the other units to a large or small extend, or to disregard them alltogether. We say:

the connection between u’ and u

does not exist	if $W(u', u) = 0$,
is called excitatory	if $W(u', u) > 0$,
is called inhibitory	if $W(u', u) < 0$.

W determines the structure of the neural network. By setting certain weights to zero we can obtain a layered network where units are not connected to each other, if they lay in the same layer or in layers that are not adjacent to each other. An importent type of neural network is the multilayer perceptron which is also the basis for the NEFCLASS-Model (see figure 1.8).

The mapping W can be modified in a so-called *learning mode*. During this phase one tries to determine W with help of a learning rule in such a way that the neural network produces certain output patterns in the presence of certain input patterns.

Output Functions

Each processing unit may posess its own output function O_u, which transforms the activation into an output value. In most neural network models the activation and the output values of a unit are identical. So each unit uses the identity as an output function.

Propagation Functions

The mapping NET assigns to each unit u of the neural network a network input function NET_u, which is also called propagation function or transfer function. Also in this case generally only one kind of function is used. It is the weighted sum of the output of all network units.

External Input Functions

An external input function ex provides the connection to the external environment for a neural network. It creates an external input for the network, which reacts by changing the activation of its units to produce an output. Usually only a subset of the network units can receive input values, i.e. ex is only defined for this subset of input units. In the multilayer perceptron these units build a layer that means they are not connectd. As activation function the identity is chosen, so the activation only depends from ex_u

Activation Functions

The mapping A makes it possible to assign to each unit of the network its own activation function. However, all units normally use the same function. An activation function A_u calculates the current activation a_u of a unit $u \in U$. In order to model a kind of "memory" or to simulate a process of "slowly forgetting" the activation function may depend on the former activation. Moreover it can depend on the network input and the external input. In most neural network models one of the activation functions shown in figure 1.7 is used.

For the multilayer perceptron the activation function of the input layer is the identity. For the hidden layers and the output layer the sigmoid function is used because the learning algorithm needs a continuously differentiable function.

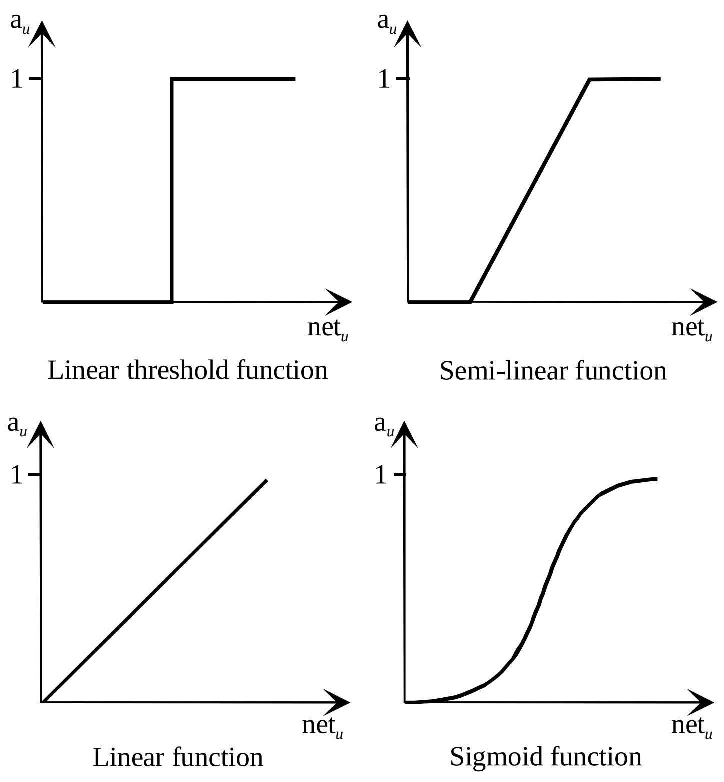


Figure 1.7: Activation functions

It was just mentioned that there exist many different neural network models. The most important is the multilayer perceptron which is also used for the NEFCLASS model. The following chapter shows the structure of this network as well as the learning process is described.

1.2.2 The Multilayer Perceptron

The multilayer perceptron is an extension of the simple perceptron introduced by Frank Rosenblatt in 1958 [ROSENBLATT 58]. A detailed description can be read in [NAUCK ET AL. 97]. The multilayer perceptron is a neural network that consists of an input layer, one or more hidden layers and an output layer. The units of the input layer do not perform any computation, they just pass on their input values. The output layer can consist of more than one unit. In classification tasks one for each class is needed. Figure 1.8 shows a multilayer perceptron using the description of a unit given in Figure 1.6.

The formal definition of a multilayer perceptron reads in [NAUCK ET AL. 97] as follows:

Definition 1.2 *A multilayer perceptron is a neural network* *MLP = (U, W, A, O,* NET, ex*), that has the following properties:*

(i) $U = U_1 \cup ... \cup U_n$, *is a set of processing units (neurons) where* $n \geq 3$ *is assumed. Furthermore,* $U_i \neq \emptyset$ *for all* $i \in \{1, ..., n\}$ and $U_i \cap U_j = \emptyset$ *for* $i \neq j$. U_1 *is called the input layer and* U_n *the output layer. The* U_i *with* $1 < i < n$ *are called hidden layers*.

(ii) *The network structure is given by the function* $W : U \times U \to \mathbb{R}$. *There only exist connections between consecutive layers.*
Thus, $(u' \in U_i \wedge W(u', u) \neq 0) \Rightarrow u \in U_{i+1}$ *for all* $i \in \{1, ..., n-1\}$.

(iii) *A assigns an activation function* $A_u : \mathbb{R} \to [0, 1]$ *to each unit* u $\in U$ *to calculate the activation* a_u *with*

$$a_u = A_u(\mathrm{ex}(u)) = \mathrm{ex}(u)$$

for all $u \in U_1$ *and*

$$a_u = A_u(net_u) = f(net_u)$$

for all $u \in U_i, i \in \{2, ..., n\}$, *where all units use the same non-linear function* $f : \mathbb{R} \to [0, 1]$.

(iv) *O assigns an output function* $O_u : \mathbb{R} \to [0, 1]$ *to each unit* $u \in U$ *to calculate the output* o_u *with* $o_u = O_u(a_u) = a_u$ *for all* $u \in U$.

(v) NET *assingns a network input (propagation) function* $\mathrm{NET}_u : (\mathbb{R} \times \mathbb{R})^{U_{(i-1)}} \to \mathbb{R}$ *to each* $u \in U_i$, $(2 \leq i \leq n)$ *to compute the network input* net_u *with*

$$\mathrm{net}_u = \sum_{u' \in U_{i-1}} W(u', u) \cdot o_{u'} + \theta_u$$

$\theta_u \in \mathbb{R}$ *is the bias of the unit u.*

(vi) $\mathrm{ex} : U_1 \to [0, 1]$ *assigns an external input* $\mathrm{ex}_u = \mathrm{ex}(u)$ *to each input unit* $u \in \mathrm{U}_1$.

A multilayer perceptron uses real-valued units and therefore, a continous activation function is necessary. The learning algorithm for the multilayer perceptron requires the activation function to be diffentiable.

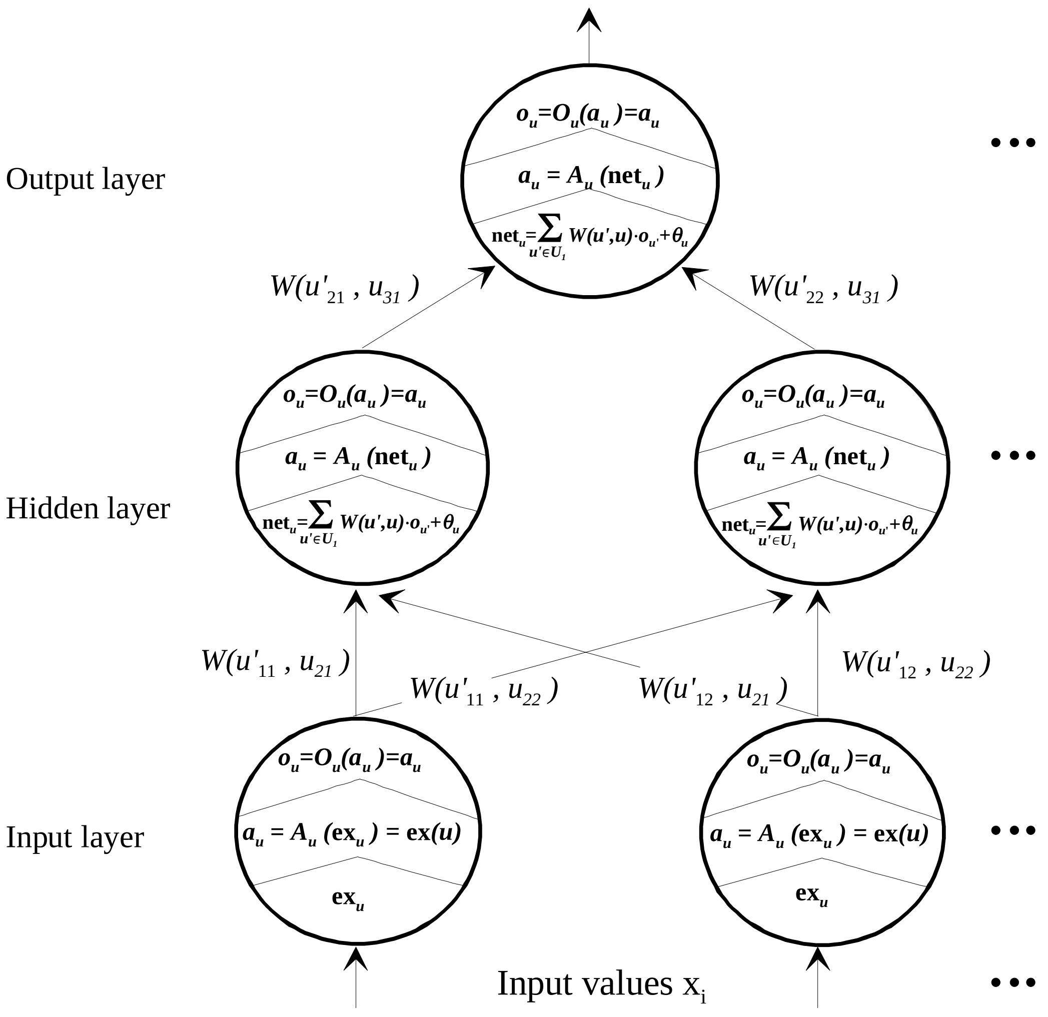


Figure 1.8: A multilayer perceptron

1.2.3 The Backpropagation Algorithm

Up to now only the structure of neural networks was considered. A very important part of neural networks are the algorithms that allow to change the weights *W*. The goal of the so-called learning algorithms is to determine *W* in such a way that the network produces certain output values when certain input values are given. The network should react in a reasonable way, when new, unknown values are presented. To achieve this goal the provided known input patterns are propagated through the network. The determined outputs are then are compared to the target output patterns. Then *W* is changed in a way that the network comes closer to the target output, when the same pattern is propagated again [NAUCK ET AL. 97].

In every step of processing, the input patterns that means the data influence the output values. Data with unknown results determine a free learning problem. The task is to receive similar output values for similar input values. If the result of the data is known the learning problem is called fixed. To express whether two inputs or outputs are similar or close to each other we need to define a similarity or error measure. This measure depends on the type of network and learning algorithm. In [NAUCK ET AL. 97] a learning algorithm is defined as:

Definition 1.3 *A learning algorithm is a procedure that changes the structure W of a neural network using a learning problem. The learning algorithm is successful, if the network solves the learning problem after the application of this procedure, or if the error of the network is less than a given error bound. Otherwise the learning algorithm has failed. A learning algorithm that uses a free learning problem is called an unsupervised learning algorithm. If a fixed learning problem is used, the procedure is called a supervised learning algorithm.*

For the NEFCLASS-Model a fixed learning problem is used so a supervised learning algorithm has to be defined. There are two problems to be solved for the definition of a learning algorithm for multilayer perceptrons. At first an output error for every unit of the hidden layer has to be calculated. Remember that these units cannot be observed from the environment. Second it has to be fixed in which way the changing of the weights should happen.

In 1986 Rumelhart, Hinton and Williams introduced in [RUMELHART/MCCLELLAND86] an algorithm which achieves to do this. This method is named *backpropagation* because the output error is propagated back through the network.

This is the way to solve the first problem and it needs four steps. In the first step the input values are propagated through the network till the output values are calculated. The second step is to calculate an error with help of an error function. To distribute this error value backwards through the network is the third step. The error values are weighted in the same way as the input values in the first step. These weighted errors reach the units and are accumulated in step four. As a result every unit gets the partial amount of the total error that it caused.

The error measure that has to be minimized for each input/output pair is defined in [NAUCK ET AL. 97].

Definition 1.4 *Let $\tilde{L}$ be a fixed learning problem, and $l \in \tilde{L}$ a pair of patterns that have to be learned.*

(i) An error measure for a supervised learning algorithm is a mapping

$$e : \mathbb{R}^{U_O} \times \mathbb{R}^{U_O} \rightarrow \mathbb{R}_0^+$$

such that for all $a, b \in \mathbb{R}^{U_O}$

$$e(a, b) = 0 \Leftrightarrow a = b$$

holds. The error $e^{(l)}$, that is caused by a neural network when processing the pair of patterns $l \in \tilde{L}$ is given by $e^{(l)} = e(\mathbf{t}^{(l)}, \mathbf{o}^{(l)})$, where $\mathbf{o}^{(l)}$ is the output (vector) of the network to the input patterns $\mathbf{i}^{(l)}$, and $\mathbf{t}^{(l)}$ is the target pattern given by the learning problem.

(ii) The error $\varepsilon_u^{(l)}$ of an output unit $u \in U_O$ when processing the input pattern $l \in \tilde{L}$ is the differnce between the output value $t_u^{(l)}$ detrmined by the learning problem and the output value $o_u^{(l)}$:

$$e_u^{(l)} = (t_u^{(l)} - o_u^{(l)})$$

With the definition of the error measure some questions have to be considered. To prohibit that negative and positive deviations neutralize each other the square of the Euclidean distance is used:

$$e^{(l)} = \sum_{u \in U_O} (t_u^{(l)} - o_u^{(l)})^2.$$

If instead of the square the absolute value is used, then another possible error measure is obtained which does not overrate large and underrate small deviations.

The second problem in defining a learning algorithm is to determine a method for changing the weights in a manner that the error can be minimized. The common supervised learning algorithms approximate a gradient descent in W and try to reduce the global error to zero this way. However, such a learning procedure cannot allways guarantee convergence, becauce it is equivalent to a local heuristic search procedure. The algorithm follows the gradient in the error surface defined on W, starting at a point given by the initialization of W and the learning problem $\tilde{L}$. The algorithm stops when it reaches a local minimum. If this local minimum is not also the global minimum, i.e. the error limit ε is not reached, then the learning process failed.

If the gradient decent is not matched closely enough, then the learning algorithm can oscillate. In such a situation the procedure aproches a minimum, but overshoots the mark.

Then a new approach from another direction takes place which leads to the same result, and so on. In this case the learning procedure also fails because it will forever iterate.

These problems are tried to solve with help of the learning rate η. The value η indicates the step width used for following the gradient. The multidimensional error surface may contain a lot of local minima and sudden strong changes in direction that have to be followed in order to reach a minimum. If the value chosen for η is too large, the error of the network can suddenly rise again. The reason might be for instance that while descending into a narrow gap of the “error mountains” the algorithm is continously jumping back and forth the walls of the gap [NAUCK ET AL. 97].

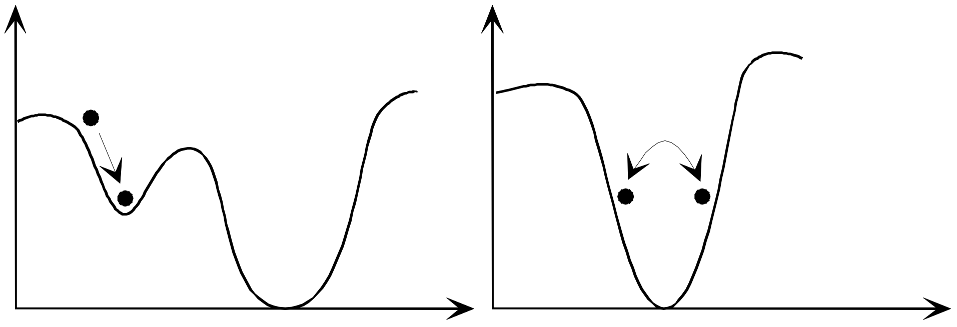


Figure 1.9: The backpropagation procedure can be trapped in local minima (left), or oscillate because the learning rate is too large (right)

As just explained a continously differntiable activation function is needed. A linear function is not used because the propagation through the network is equivalent to a multiplication of a tuple with a matrix which would unter mathematical consideration lead to a onelayered network. Thus a sigmoid function is used. The formal definition of the backpropagation algorithm is given in [NAUCK ET AL. 97].

1.3 Neuro-Fuzzy Systems

We just discussed the problem of defining and tunig a fuzzy system and found out that it would be fine to have learning opportunities for this. Neural networks offer this learning opportunity but on the other hand the are very successful in classification problems. So why should there fuzzyness be integrated? Why not only use neural networks? Let us go into it further with an overview of advantages and disadvantages of fuzzy systems and neural networks [NAUCK ET AL. 97].

neural network	**fuzzy system**
advantages	
● no mathematical process model required ● no rule-based knowledge required ● different learning algorithms available	● no mathematical process model required ● prior (rule-based) knowledge can be used ● simple interpretation and implementation
disadvantages	
● black box ● rules cannot be extracted (usually) ● determine heuristic parameters ● adaption to modified environment can be difficult and relearning may be necessary ● prior knowledge cannot be used (learning from scratch) ● no guarantee that learning converges	● rules must be available ● cannot learn ● no formal methods for tuning ● semantical problems in interpreting tuned system ● adaption to modified environment can be difficult ● tuning may be not successful

Table 1: Comparing neural networks and fuzzy systems

Neuro-fuzzy systems are created to overcome the disadvantages of neural networks and fuzzy systems. The term is usually used for every kind of combination of neural networks and fuzzy systems. One approach is to combine both in such a way that learning algotithms are used to determine parameters of fuzzy systems. This means that the main intention of a neuro-fuzzy approach is to create or improve a fuzzy system automatically by means of neural network methods. An even more important aspect is that the system should always be interpretable in terms of fuzzy if-then rules, because it is based on a fuzzy system reflecting vague knowledge. In a word: the task is to overcome the disadvantages without loosing the advantages.

On the other hand a fuzzy neural network is a neural network that uses fuzzy methods to learn faster or perform better. In this case the improvement of the neural network is the main intention. An interpretation in terms of fuzzy rules is neither important nor possible here, because the system is based on a neural network with black box characteristics. In [NAUCK ET AL. 97] the following taxonomie to describe the different combinations of neural networks and fuzzy systems is used:

Fuzzy neural networks
Fuzzy methods are used to enhance the learning capabilities or the performance of a neural network. This kind of approaches is not to be confused with neuro-fuzzy approaches.

Concurrent "neural/fuzzy systems"
A neural network, and a fuzzy system are working together on the same task, but without influencing each other, i.e. neither system is used to determine the parameters of the other. Usually the neural network processes the input to, or postprocesses the outputs from the fuzzy system.These kinds of models are strictly speaking neither neuro-fuzzy approaches nor fuzzy neural networks.

Cooperative neuro-fuzzy models
A neural network is used to determine the parameters (rule, rule weights and / or fuzzy sets) of a fuzzy system. After the learning phase, the fuzzy system works without the neural network. These are simple forms of neuro-fuzzy systems, and the simplest form - determining rule weights by neural learning algorithms - is widely used in commercial fuzzy development tools, even though semantical problems can arise.

Hybrid neuro-fuzzy models
Modern neuro-fuzzy approaches are of this form. A neural network and a fuzzy system are combined into one homogeneous architecture. The system may be interpreted either as a special neural network with fuzzy parameters, or as a fuzzy system implemented in a parallel distributed form. There a lot of different models described in [NAUCK ET AL. 97]. Some of these approaches are reinforcement learning types that are especially suited for control tasks and others are multi-purpose models, which use supervised learning, and can be used for data analysis, like the NEFCLASS approach.

As we are here only interested in hybrid neuro-fuzzy systems, in the further descriptions we restrict ourselves to informations needed as a basis for this approach. In the following. [NAUCK/KRUSE 98] a definition is given that shall be used here to specify what neuro-fuzzy system means in this thesis:

1. A neuro-fuzzy system is a fuzzy system trained by a (heuristical) learning algorithm (usually) derived from neural networks.
2. A neuro-fuzzy system can be represented by a feed-forward neural network architecture. However, this is not a prerequisite to training, it is merely a convenience to visualise the structure and the flow of data.
3. A neuro-fuzzy system can always be interpreted in terms of fuzzy if-then rules.
4. A neuro-fuzzy system‘s training procedure takes the semantics of the underlaying fuzzy model into account to preserve the linguistic interpretability of the model.
5. A neuro-fuzzy systems performs (special cases of) function approximation. It has nothing to do with fuzzy logic in the narrow sense. i.e. generalized logical rules [KRUSE ET AL. 94].

The second list item tells us something about the viewpoints that can be taken. The hybrid approach is to interpret the rule base of the fuzzy system in terms of a neural network. The fuzzy sets can be seen as weights and the input and output variables and the rules can be interpreted as neurons. This way a fuzzy system can be interpreted as a special neural network. The learning algorithm works by modifying the structure and/or the parameters that means the inclusion or deletion of neurons or adaption of the weights.

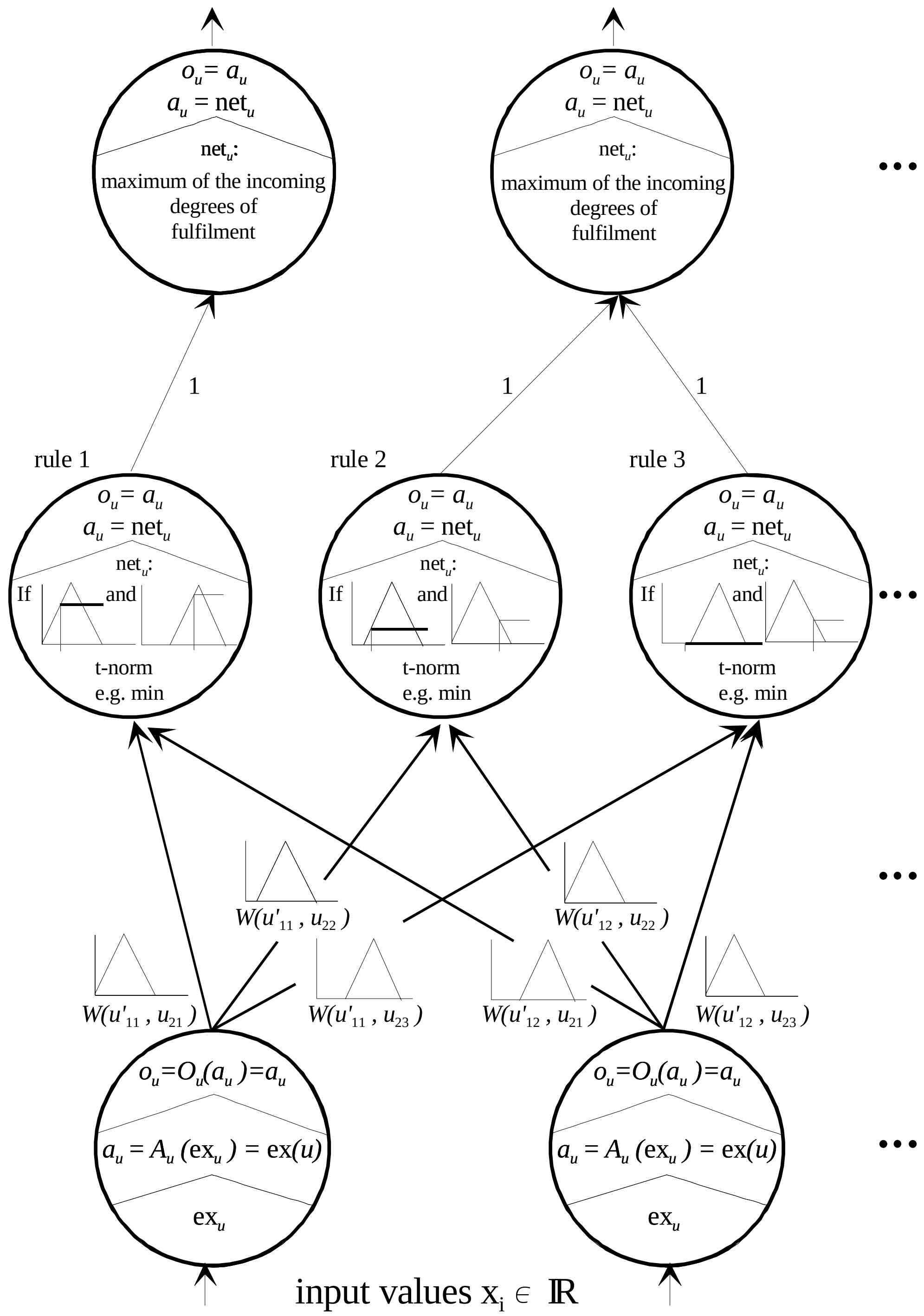


Figure 1.10: A neural network view of a neuro-fuzzy classifier

It is an important aspect that the changes caused by the learning process can be interpreted in terms of neural networks as well as in terms of fuzzy systems. The black box behavior of neural networks is avoided and a successful learning process can be seen as an increase of explicit knowledge which is represented in the rule base.

The task is now to think about how to create such a neuro-fuzzy system, in which way the rules base is built, and what kind of learning algorithm is used for that. It should be possible that the rules can be learned from sratch based on training data. In [NAUCK ET AL. 97] three possibilities of creating a rule base are considered.

- The system starts without rules, and creates new rules until the learning problem is solved. Creation of a new rule is triggered by a training pattern which is not sufficiently covered by the current rulebase [BERENJI / KHEDKAR 93, NAUCK / KRUSE 95, TSCHICHOLD GÜRMAN 95]. This approach can lead to large rule bases if the membership functions are not appropriately chosen.
- The system starts with all rules that can be created due to the partitioning of the variables, and deletes insufficient rules from the rule base [NAUCK / KRUSE 93]. For this procedure an evaluation of the performance of individual rules is needed. This approach encounters complexitiy problems when applied to problems with a large number of variables. Inconsistent rule bases are avoided, because a consistency test is part of the evaluation. It is possible to obtain rule bases with too few rules by this procedure.
- The system starts with a (possibly randomly chosen) rule base with a fixed number of rules. During learning rules are replaced [SULZBERGER ET AL.93], while the consistency of the rule base has to be checked at each step. The drawback is the fixed number of rules. Additionally an evaluation scheme for rule deletion, and a data analysis procedure for acquiring new rules, must be implemented. If this is not done, then the learning is equivalent to stochastic search. In the event of degration in performance, replacements may possibly have to be cancelled [SULZBERGER ET AL.93].

Another problem to think about is the application of the learning algorithm to the system. While using the neural network view the fourth point of the definition is easily forgotten which requires that the neuro-fuzzy system‘s training procedure takes the semantics of the underlaying fuzzy model into account to preserve the linguistic interpretbility of the model.

We also just described that learning algorithms are usually gradient descent methods. But they cannot be applied directly to a fuzzy system, because the functions used to realize the inference process are usually not differentiable. There are two solutions for this problem [NAUCK/KRUSE 98b]:

- replace the functions used in the fuzzy system (like min, max and membership functions) by differentiable functions, or:
- do not use a gradient-based neural learning algorithm but a better suited procedure.

The problem with the first solution is that the interpretability may be reduced. The NEFCLASS model uses the second possibility, a special learning algorithm which will be introduced in the next chapter.

2

The NEFCLASS Model

In this chapter the neuro-fuzzy method NEFCLASS is discussed. This method is an linguistic approach to construct fuzzy systems from data by applying a heuristic data-driven learning algorithm that computes local parameter modifications. There are several other methods which are able to do this but the focus of the NEFCLASS model lies in the interpretability of the developed classifier. The constraints of the learning algorithm allow the user to interactively influence the training process.

So the main goal of NEFCLASS is to create a readable classifier that also provides an acceptable accuracy. However, the user has to be aware, that readability and accuracy do not go together. An interpretable fuzzy system should display the following features:

- few meaningful rules with few variables in their antecedents,
- few meaningful sets for each variable,
- there are no rule weights,
- identical linguistic terms are represented by identical fuzzy sets,
- only normal fuzzy sets are used, or even better fuzzy numbers or fuzzy intervals.

These features pose a lot of restrictions on the way how a fuzzy system can be created from training data. If high performance of a fuzzy system is the main goal, then it is necessary to fit the system to the data very accurately. This approach, however, usually yields fuzzy systems that do not display the above-mentionend features and are in fact black-box models. A user has therefore to decide, what is more important - accuracy or readability. NEFCLASS provides means to ensure the readability of the solution by giving the user complete control over the learning process. It should also be stressed that interpretable solutions can usually not be obtained without the user‘s cooperation. The user must decide whether the solution‘s readability is *sufficient* or not, and must ready to influence the learning process when necessary. NEFCLASS must be seen as a tool that supports users in finding readable fuzzy classifiers. It is *not* an automatic classifier creator where data is fed in and a solution pops out. It is necessary that the user *works* with this tool (with *work* being the operative word) [NAUCK/KRUSE 98]. For this reason only fast learning strategies are used to give the user the possibility to interact with the tool.

2.1 The Structure of the NEFCLASS-Model

As just mentioned in the last chapter it is possible to view a neuro-fuzzy system as a special three layered feedforward neural network where

- the first layer represents the input variables that means the pattern tuples,
- the hidden layer represents fuzzy rules,
- the third layer represents the output variables that means one unit for every class,
- the units use t-norms and t-conorms as activation functions,
- the fuzzy sets are encoded as (fuzzy) connection weights.

Figure 1.10 and Figure 2.1 show this neural network structure which is often used to demonstrate the parallel structure and the data flow through the model, both for learning (backward path) and classification (forward path). Furthermore it is more easy to compare NEFCLASS to other fuzzy classication approaches if this representation is chosen. But it also should be remembered again that this is only one possible visualisation. This system is *not* a neural network. It is a hybrid neuro-fuzzy system which is an integrated system. If only the neural network view is considered the advantages of the fuzzy view are lost and these are the features NEFCLASS focuses on.

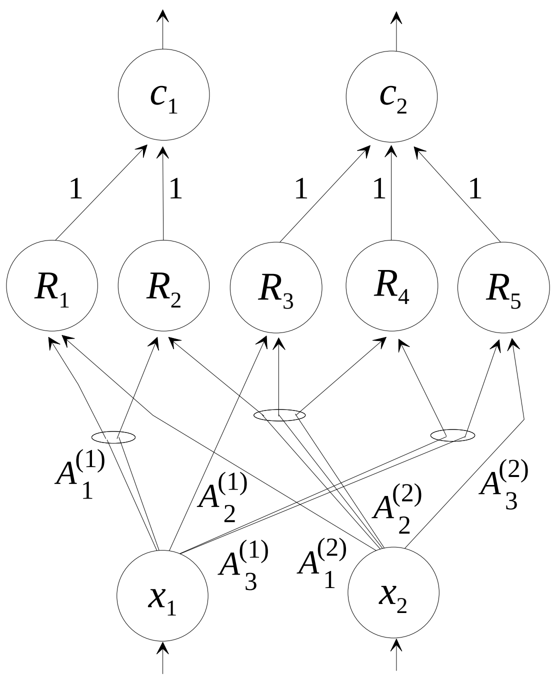


Figure 2.1: The architecture of the NEFCLASS model

A more simple version of Figure 1.10 is Figure 2.1. The R_k represent the rules and $A_j^{(i)}$ is the weight $W(x_i\ R_k)$ where index j selects the fuzzy sets of the partition. NEFCLASS uses shared weights on some of the connections (in Figure 2.1 this is shown by elipses drawn around the connections). This way it is made sure that for each linguistic value (e.g. "x_1 is *positive big*") there is only one representation as a fuzzy set. It cannot happen that two fuzzy sets that are identical at the beginning of the learning process develop differently, and so the semantics of the rule base encoded in the structure of the network is not affected [NAUCK / KRUSE 98]. Connections that share a weight always come from the same input unit because a label (e.g. “positive big”) should have the same meaning whenever it is used for a certain variable but need not to have the same meaning for all variables.

$W(R_k, c_m)$ is the connection from the rule R_k to the output unit c_m. For semantical reasons, i.e. to avoid weighted rules these connections are fixed at either 0 (connection does not exist) or 1 (connection exists). Each rule unit is connected to exactly one output unit [NAUCK / KRUSE 98b]. The output activation is computed by a maximum operation instead of a weighted sum.

We have to take care now that we do not get confused with the term “learning”. On one hand we talk of “learning a network from data” or “from scratch” what means that the rule base is learned, and - it depends on the view - a fuzzy system or a neural network is built.

On the other hand there is the learning process which is also named the “training of the network”. This is the real learning process where the weights are changed such that in the end a classifier is created.

So in the first learning step the *structure* of the classifier is created that fits the database as good as possible, and already has some qualities the user requires (few number of rules and fuzzy sets). In the second learning step the *classifier* is completed by determining the parameters of the system in an iterative training process to improve the accuracy without loosing the interpretability. These two steps will be described now in the next subsections.

2.2 Learning a Rule Base - The Algorithm

A NEFCLASS system can be built from partial knowledge about the patterns, and can be refined by learning, or it can begin with an empty rule base that is filled by creating rules from the training data. For each input variable the user must decide how many fuzzy sets are to be used to partition the domain of the respective variable. By this the granularity for each variable and the linguistic terms that can be used by the classifier are given. For some variable the user might prefer to distinguish just between small and large, for some other variables a finer partition may be useful. The user must also specify a value of $k_{\max}$, i.e. the maximum number of rule nodes that may be created in the hidden layer. For each class there must be at least one rule. It is also possible to let NEFCLASS find a suitable value for $k_{\max}$ by itself [NAUCK / KRUSE 98b].

In the following it is assumed that triangular membership functions described by three parameters are used:

$$\mu : \mathbb{R}^n \rightarrow \mu(x) = \begin{cases} \dfrac{x-a}{b-a} & \text{if } x \in [a,b), \\ \dfrac{c-x}{c-b} & \text{if } x \in [b,c], \\ 0 & \text{otherwise.} \end{cases}$$

In addition, the left and the right-most membership functions for each variable can be shouldered, i.e. the triangle becomes a half trapezoid.

Consider a NEFCLASS system with:
- n input units $x_1, ..., x_n$,
- $k \leq k_{max}$ initial rule units $R_1, ..., R_k$ (prior knowledge, k = 0 what means no prior knowledge is given),
- m output units $c_1, ..., c_m$,
- a learning set $\tilde{L} = \{(\boldsymbol{p}_1, \boldsymbol{t}_1), ..., (\boldsymbol{p}_s, \boldsymbol{t}_s)\}$ of s patterns, each consisting of an input pattern $\boldsymbol{p} \in \mathbb{R}^n$, and a target pattern $\boldsymbol{t} \in \{0, 1\}^{m.}$

Assume that NEFCLASS is initialized with $k \leq k_{max}$ fuzzy rules. The rule base of NEFCLASS is completed by finding for each pattern $\boldsymbol{p}$ a combination of fuzzy sets that yields the highest degree of membership for each value p_i. This combination of fuzzy sets is the antecedent of a prospective rule. If such an antecedent does not already exist, it is stored in a list. A suitable consequent for each antecedent is determined by adding up the degrees of fulfilment for all patterns separately for each class. The consequent is set to that class label that obtains the largest sum. After each training pattern was processed once, we obtain a rule base of k' rules. If $k' > k_{max}$, only the best k_{max} rules (‘best’ rule learning) or the best k_{max}/m rules for each class (‘best per class’rule learning) are kept, all other rules are deleted from the rule base. The best rules are determined by computing performance values for each rule. If a rule correctly classifies a pattern, its degree of fulfilment is added to its performance value, if not, the degree of fulfilment is subtracted. The performance values can be computed on the fly, such that rule learning is completed after a single sweep through the training set. In Figure 2.2 the rule learning algorithm is given in pseudo code.

From the two described ways to create a rule base for a NEFCLASS system the ‘best per class’ option should be selected, when one supposes that the patterns are distributed in an equal number of clusters per class. This strategy is especially suitable, if there are classes with much less patterns than other classes, because it guarantees that each class is covered with rules independently from the distribution of patterns. ‘Best’ rule learning is suitable, when there are classes, which have to be represented by a larger number of rules than other classes [NAUCK / KRUSE 98b].

```
For each pattern (p, t) of $\tilde{L}$ do
   begin
      For each input feature do
         find $\mu_{ji}^{(i)}$ such that $\mu_{ji}^{(i)}(p_i) = \max_{j \in \{1,\dots,q_i\}} \{\mu_j^{(i)}(p_i)\}$;
      Create antecedent $A = (\mu_{ji}^{(1)}, \dots, \mu_{jn}^{(n)})$;
      If $A$ is not in list of antecedents
         then add antecedent A to list of antecedents;
   end;
For each pattern (p, t) of $\tilde{L}$ do
   For each antecedent $A_j$ do
      begin
         $c$ = class index of p given by t;
         $C_j(c) = C_j(c) + A_j(\mathbf{p})$          (* add degree of fulfilment *)
      end;
For each antecedent $A_j$ do
   begin
      $c = \operatorname{argmax}_{i \in \{1,\dots,m\}} \{C_j(i)\}$;
      create rule $R_j$ with antecedent $A_j$ and consequent $c$;
      add $R_j$ to the list of rule base candidates;
      $performance_j = C_j(c) - \sum_{i \in \{1,\dots,m\},\ i \neq c} C_j(i)$
   end;
If "best" rule learning
   then For $i = 1$ to $k_{\max}$ do
           begin
              $R = \operatorname{argmax}_{R_j} \{performance_j\}$
              add $R$ to rule base;
              delete $R$ from list of rule candidates;
           end
   else If "best per class" rule learning
      then For each class $c$ do
              For $i = 1$ to $\frac{k_{\max}}{m}$ do
                 begin
                    $R = \operatorname{argmax}_{R_j, consequent_j = c} \{performance_j\}$
                    add $R$ to rule base;
                    delete $R$ from list of rule candidates;
                 end;
```

Figure 2.2: The rule learning algorithm in pseudo code

The learning algorithm can be visualized in a grid structure. Figure 2.3 shows how rules are selected from a grid structure in feature space that is given by the fuzzy sets of the individual variables. In this case the system is allowed to create three rules, therefore there are unclassified patterns.

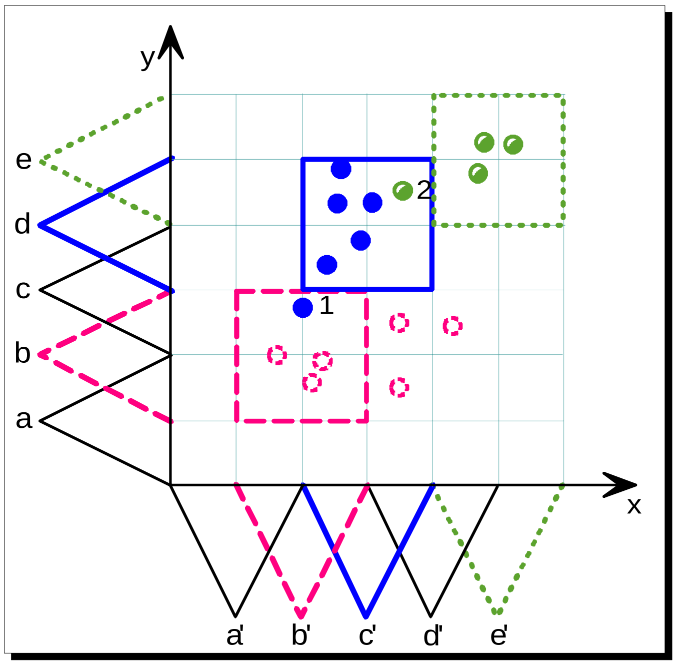


Figure 2.3: Classification after rule learning with NEFCLASS

One can see that the classification result is not bad, but improvements are desired. Pattern 1 and pattern 2 are misclassified and three patterns are not classified. To shift and modify the fuzzy sets would help:

- Pattern 1 is correctly classified if
    - fuzzy set *b* is a bit smaller from the top,
    - fuzzy set *d* is a bit wider to the bottom,
    - fuzzy set *c'* is a bit wider to the left.
- Pattern 2 is correctly classified if
    - fuzzy set *c'* is a bit smaller from the right,
    - fuzzy set *e'* is a bit wider to the left.
- The unclassified patterns are correctly classified if
    - fuzzy set *b'* is a bit wider to the right.

2.3 Training Fuzzy Sets - The Algorithm

The supervised learning algorithm of NEFCLASS to adapt its fuzzy sets runs cyclically through the learning set $\tilde{L}$ until a given end criterion is met, e.g. if a number of admissible misclassifications is reached, or if the error cannot be decreased further, etc.

After a pattern is propagated, the error is determined for each output unit. Based on this error, for each active rule unit it is decided whether its degree of fulfilment should be larger or smaller. The membership function that is responsible for the degree of fulfilment is identified and only this fuzzy set is adapted accordingly. A fuzzy set is only modified, if this does not violate the constraints specified by the user. Typical constraints are for example:

- fuzzy sets must overlap to a fixed degree,
- fuzzy sets must not pass each other (i.e. exchange their relative positions),
- fuzzy sets must stay symmetrical,
- membership degrees must add upto 1.0,
- etc.

Users can select one or more constraints depending on their needs. Constraints like these help to obtain an interpretable rule base, but may cause al loss of performance in classification [NAUCK / KRUSE 98b]. In figure 2.4 the learning algorithm is given in pseudocode. Here only the algorithm for triangular membership functions is given. The necessary changes for triangular or bell shaped functions are straight forward and can be found in the source code of the implementation.

```
repeat
   propagate the next pattern (p, t);
   for each output unit $c_i$ do
      $ec_i$ = $t_i$ - activation($c_i$);
   for each rule unit $R$ with activation($R$) > 0 do
      begin
         $e_R = activation(R) \cdot (1 - activation(R)) \cdot \sum_{c_i} (W(R,c) \cdot ec_i)$
         $j = \operatorname{argmin}_{i \in \{1,\ldots,n\}} \{W(x_i, R)(p_i)\}$
         $\mu = W(x_j, R)$
         (* $a_\mu$, $b_\mu$ and $c_\mu$ are the parameters of the fuzzy set $\mu$ *)
         $\delta_b = \sigma \cdot e_R \cdot (c_\mu - a_\mu) \cdot \operatorname{sgn}(p_i - b_\mu)$;
         $\delta_a = -\sigma \cdot e_R \cdot (c_\mu - a_\mu) + \delta_b$;
         $\delta_c = \sigma \cdot e_R \, (c_\mu - a_\mu) + \delta_b$;
         modify $\mu$ with $\delta_a$, $\delta_b$, $\delta_c$, without violating the constraints for $\mu$;
   end;
until end criterion;
```

Figure 2.4: The fuzzy set learning algorithm in pseudo code [NAUCK / KRUSE 98b]

The learning procedure for the fuzzy sets is a simple heuristics. It results in shifting the membership functions and in making their supports larger or smaller (see Figure 2.5). By changing only the fuzzy set that delivered the smallest membership degree for the current pattern, the changes are kept as small as possible.

The sum in the computation of e_R (line 7 in Figure 2.4) is not really necessary, because each rule unit is connected to only one output unit (i.e. there is just one $W(R, c) \neq 0$). But it makes the model more flexible, because it would be possible to also use adaptive rule weights. Although the implementation of the NEFCLASS model allows to use rule weights, it is recommended not to use them in order to keep the semantics of a NEFCLASS system. It is not clear what a weighted fuzzy rule is supposed to mean. Rule weights are often superfluous, because they can be represented as changes in the membership functions [NAUCK/KRUSE 98c]. In [NAUCK ET AL. 97] it is reported that rule weights are not necessary to obtain good classification results. However, without rule weights a NEFCLASS system usually cannot produce exact output values of 0 or 1 due to the mathematics involved. For the same reason the learning procedure cannot reach an error value of zero, and therefore the change in error is usually used as a stop criterion for the learnig algorithm.

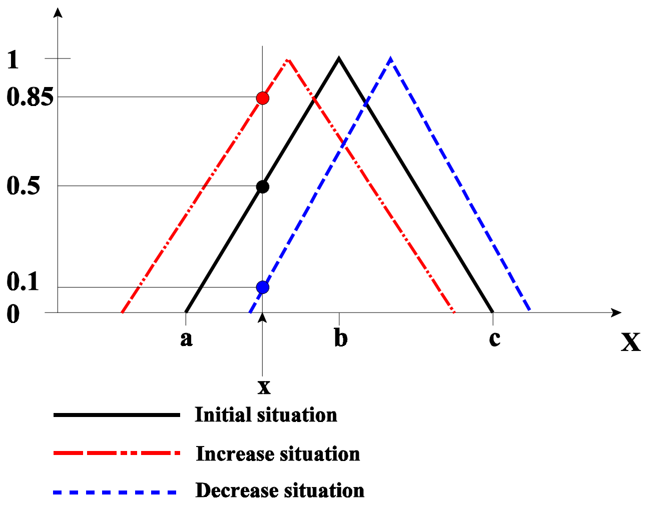


Figure 2.5: Adaption of fuzzy sets

The adaption of the fuzzy sets is carried out by simply changing the parameters of its membership function in a way that the membership degree for the current feature value is increased or decreased respectively.

Figure 2.6 displays the situation after the learning algorithm for the membership functions was applied to improve the classification result. Here no constraints were used to restrict the learning process. As can be seen the resulting fuzzy partitions are no longer nicely interpretable. Such a result is an indication to repeat the learning process with other parameters. In this case it would have been better to allow the system to create four rules and to use constraints for training the membership functions. This example is to illustrate that it is important for the user to work interactively with approaches like NEFCLASS to obtain readable solutions [NAUCK / KRUSE 98].

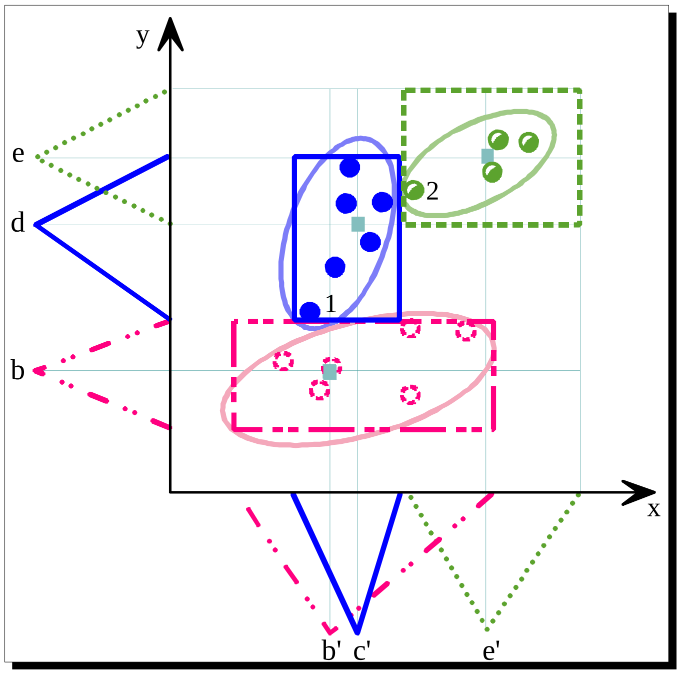


Figure 2.6: Classification after learning fuzzy sets

Compared to neural networks, NEFCLASS uses a much simpler learning strategy. There is no vector quantisation involved to find rules (clusters), and the membership functions are not trained by gradient descent. Fuzzy rule creation can be seen as a selection from an initially given virtual rule base, specified by a fuzzy partition of the input domain.

From the viewpoint of the NEFCLASS architecture and the flow of data, the fuzzy sets are trained by a backpropagation-like algorithm. We use the term backpropagation to denote the idea of a learning procedure, not a special implementation in form of an algorithm. Backpropagation means to compute an output error and to propagate it backwards through the architecture from the output units towards the input units. This error signal is used to locally change parameters. Neural networks often implement backpropagation by gradient descent. NEFCLASS does not compute any gradient information. It uses a much simpler heuristic instead. In addition, the adaptivity of a NEFCLASS system is restricted compared to neural networks. This restriction is due to the initially given fuzzy partitions, which define the form and maximal number of clusters, and by the constraints that do not admit certain changes in the fuzzy sets [NAUCK / KRUSE 98].

2.4 Prunining the Rule Base

The learning algorithm of the NEFCLASS-Model provides good results for many classification problems. However, a good interpretation of the learning result cannot always be guaranteed, especially for high-dimensional problems. Because interpretation is one reason for using a fuzzy classifier in the first place, there is a need to enhance the learning algorithms of a neuro-fuzzy system with techniques for simplifying the obtained result [NAUCK/KRUSE 97]. Four pruning strategies are used to improve the interpretability of the classifier.

- Delete linguisitc terms from the antecedent of a rule under certain ascpects:

    - A variable is not important for the classification.

    The task is to find out if there is an input variable that is not necessary for the classification. For this the correlations of the input variables with the class information is used. Variables that have a low correlation are tested whether they can be deleted from the antecedet or not. Under a statistical view it is dubious to use the correlation, but it is only used to find out a sequence of testing the variables and is not used to delete a variable.

    The variable with the lowest correlation is the first to be tested. It is deleted from the antecedents of all rules, a consistency check of the rule base follows and the fuzzy sets are trained. If this improves the classification result this classifier is taken as the current one and the next variable with the lowest correlation is tested. If a test fails and no improvement was reached the last classifier that improved the classification result is the outcome of this pruning method.

    - A variable is not important for the degree of fulfilment of a rule.

    For each rule the linguistic term is identified whose membership degree is identical to the degree of fulfilment of the rule in the least number of cases. The rule with the smallest number is selected and the identified term is removed from its antecedent, and a consistency check is done. After this the fuzzy sets are trained and if the classifer can be improved it is kept and this pruning step is repeated. If the classifier cannot be improved the previous one is restored. The last classifier that improved the classification result is the outcome of this pruning method.

    - A term uses fuzzy set with a very large support

    If a variable is partitioned by more than two fuzzy sets, sometimes the support for one or more of them can become quite large during learning. This can be seen as evidence, that such a fuzzy set is superfluous.

    The fuzzy sets of every variable are sorted by their width. The term with the widest fuzzy set of all is deleted from all rules. Only in rules where it is the last term of the antecedent it is not deleted. A consistency check of the rule base follows and the fuzzy sets are trained. If this improves the classification result this classifier is taken as the current one and the fuzzy set with the next biggest width is examined. This process stops when the classification result is not improved anymore. The last classifier that improved the classification result is the outcome of this pruning method.

- Delete rules that never or very rarely provide the maximum degree of fulfilment for the class given by their consequent.

  All patterns are presented to the rules. For every rule it is counted how often the classification is correct and it produces the maximum degree of fulfilment for the correct class. The rule with the smallest number is deleted and the fuzzy sets are trained. If this improves the classification result this classifier is taken as the current one and the rule with the next lowest number is deleted. This process stops when a test fails and no improvement was reached or if no rule can be deleted because there is only one for each class. The last improved classifier is the result of this pruning method.

3

Java for NEFCLASS

Up to now there existed two NEFCLASS implementations which are named NEFCLASS-PC and NEFCLASS-X. To explain why a third implementation is developed now in Java both tools are described very shortly now. In a second subsection the advantages of the Java language are described.

3.1 The NEFCLASS Tools

In [UNauck 97] NEFCLASS-PC is desribed as an interactive simulation software to develop, train and test a neuro-fuzzy system for classification. It is written in Turbo-Pascal. NEFCLASS-PC 2.04 is the fifth released version of the neuro-fuzzy classification software for MS-DOS PC using an 80286 processor or better. This description shows that the implementation is a bit "old fashioned". The interface has nothing of a todays standard that for example the Windows interface offers. But this version was very successful. There were about 5.000 downloads from the Internet. For this reason the wish for a UNIX version and some more features arose.

The UNIX version NEFCLASS-X was developed in 1996. It consists of a C++ program that implements the NEFCLASS approach and a separate user interface written in TCL/TK. The software was written for UNIX environments, but it is possible to run it under Windows since TCL/TK is available for these platforms. Additional to the features of NEFCLASS-PC it implements pruning strategies for reducing the rule base which have to be supervised by the user.

It was the problem to update two implementations and to take care that both versions are equivalent. With the upcoming of Java which promises a platform independent development of software tools it was decided to create a third implementation. This is NEFCLASS-J. NEFCLASS-J is developed under "Visual Cafe for Java" an application development tool of Symantec Ltd.

Some new features of NEFCLASS-J the other implementations do not have are:

- Batch learning instead of online learning.
  With online learning the weight changes are executed after propagation of each pattern. So in dependency of how the patterns are listed in the data file the learning algorithm will go different ways in the error surface to find a minimum. This need not to be bad

because this randomly can lead to a ‘good’ minimum. However, the wish is to create a classifier which is not influenced by random events during the training process. Therefore the aim is to reach independency from the data. This is the reason for batch learning. The changes are accumulated and after all patterns are propagated the execution of the weight changes takes place. The result is not necessarily better but it is made sure that the quality of the classifier is not only high because the data were presented in a suitable condition by chance.

- Automaticaly determination of the number of rules.
  This feature is explicitly described in the fifth chapter.
- Cross validation of the classifier.
  This feature is explicitly described in the fifth chapter.
- Handling of missing values.
  The membership degree is set to ‘one’ for all fuzzy sets of a variable if a value is missing in the data.

3.2 The Java Language

Java developed by Sun Microsystems, was designed for creating applets and applications for Internets, intranets, and any other heterogeneous, distributed network. This language offers the following powerful features, as described in the Java white papers published by Sun [SYMANTEC 97]:

- **Simple**. Java is similar to C and C++, which many programmers are already familiar with. Some of the more difficult features of C++, including operator overloading, pointers and pointer arithmetic, multiple inheritance, and extensive automatic coercions, were omitted to make programming with java eadier. The Java automatic garbage collection feature reduces bugs by automatically freeing unused memory.
- **Small.** The Java virtual machine is relatively small in size, so it can be downloaded over the Internet and run on computers with little available memory. Many operating systems will include Java in the future.
- **Object-oriented**. Java mimics the object orientation of C++ and includes extensions from Objective C for dynamic method resolution. Some advantages of object-oriented programming include the following:
  - Code is encapsulated in objects, which have a public interface and a private implementation, so one can rapidly develop prototypes and group code into manageable chunks - even for very complex systems.
  - Objects can inherit the characteristics of other objects and override inherited characteristics, so one can easily reuse code, make code more compact, and fix or update code in one place, which saves time and reduces bugs.

- **Network-ready**. Creating network connections is easier in Java for C or C++ because Java has built-in routines for dealing with TCP/IP, including HTTP and FTP. These routines make it as easy to open and access objects over the network through URLs as it is to access a local file system.
- **Robust**. Java eliminates problems early by requiring declarations, using static typing, having the compiler perform type check, and not supporting pointers, which can result in overwriting memory or corrupting data.
- **Secure**.Because there are no pointers, Java applications cannot access data structures or private data that they do not have access to. This prevents most viruses from taking hold. Applets, when run within a Web browser on a local computer, cannot read or write the disk, execute programs on this computer, or connect to any other computers except the server they were downloaded from.
- **Architecture-neutral and portable**. The Java compiler generates an architecture-neutral object file format and bytecode instructions, so Java can run on any computer that has a Java runtime system. Bytecodes are instructions that are similar to machine code, but are not platform specific. During execution, the Java virtual machine either interprets the bytecode or converts them to machine code. Creating sepatate applications for different computer platforms is no longer and issue.
- **High-performance**. Java bytecodes can be translated on the fly to native machine instructions - for example , by a Java-enabled browser. Linking is faster than for C or C++. Once the Java bytecodes are converted to machine code by a Just-In-Time compiler in a Java virtual machine, the performance is comparable to that of C or C++.
- **Multithreaded**. Java code can deal with multiple things happening at once with sophisticated synchronization primitives that are integrated into the language, which makes them easier to use and more robust. Multithreading improves interactive responsiveness and real-time behavior, so is critical to high-performance Java applets because applet execution must continue while various image and binary files are being retrieved from the Web servers. In addition, the ability to control the execution of multiple concurrent threads is crucial for deploying real-world Web-applications.
- **Dynamic**. New module plug-ins can be added to a Java application with minimal overhead. Java can look up a class definition at runtime from its name.

For the development of the NEFCLASS implementation Symantec “Visual Cafe Professional Development Edition” was used. It is an integrated development tool. Interfaces can be designed graphically by mouseclicks and the code is generated automatically. This code is descripted with comments what helps to integrate the underlying algorithms such that the visual tool and the Java code always match. The author found it comfortable to use and it supported to learn the Java language. The handling is comfortable and intuitive. With help of an Interaction Wizard it is possible to create component connections very easily. The appropriate method is created automatically which can be changed afterwards. This way one is very easily enabled to create a complex application where all features are designed and then improve it step by step.

4

Experiments with NEFCLASS-J

In this chapter some experiments to test the performance of the NEFCLASS-J algorithms are described. The results of these experiments are compared to the results of other classifiers. For the experiments the ‘Wisconsin brest cancer’ (WBC) data set is used.

In comparison to NEFCLASS-X the use of the pruning methods changed. There the user has to select a pruning method, and then has to initiate every pruning step the by himself. He has to determine the control parameters for a method and to edit the rule base. The user has the total control of the pruning process but it is uncomfortable because it needs time and the user needs some knowledge about the pruning methods. With NEFCLASS-J the pruning process happens automatically if the user decides to start the process. There is no need to know anything about the methods but on the other hand the process cannot be controlled. After every pruning step the fuzzy sets are trained and every method implements an automatic restore if a pruning step is not successful. All four pruning methods are automatically used one after the other.

- The first method is to find out variables that are not important for the classification result and can be deleted from the antecedents of all rules.
- With the second method rules are deleted that never or rarely provide the maximum degree of fulfilment for the class given by their consequent.
- The third method looks for variables that are not important for the computation of the degree of fulfilment of a rule.
- The fourth method deletes terms with fuzzy sets with a very large support.

The philosophy of implementing the pruning feature differs from ‘total control’ in NEFCLASS-X to ‘absolutely no control’ in NFCLASS-J. For an update version of NEFCLASS-J it is planned to give back some control to the user. Then the methods will be offered to the user so that he can select how often a certain method should be used.

After that the parameter settings for a classifier are found it is time to think about the reliability and the validity of this classifier. The reliability for the classifier is given because the process of classification is deterministic. So the same data will always produce the same result.

The classifier is automatically created from the data set, with the task that this classifier should be able to classify any data of this certain type. Only a limited number of patterns can be used to create the classifier and the quality of the classification result will always depend on these patterns used for creation. If these training patterns do not represent the set of patterns that can occur in reality the classifier will never be able to classify all possible patterns. The same problem occurs, of course, if the data of different classes overlap such that there are ambigous patterns. This is the problem of modeling real word problems and finding the right features to describe the patterns.

So let us assume that we created a good pattern set. The next problem is that only a part of it can be used for creating the classifier. The other part has to be used as unseen data containing class information to verify the classification result. Here we also have to take care that the selected part of the patterns represents the whole set. As this is a tedious task, this partitioning of the pattern set should happen automatically. A method where the whole pattern set is used to create the classifier, and where several validation steps are done before, in order to find out a measure for the quality of the resulting classifier, is called *cross validation*.

For cross validation the pattern set is randomly devided into a number of stratified samples. The first part is taken to create the classifier and the rest of the pattern set is used to test the classifier. The result of this test is an error value. Then the next part is taken to create the classifier and the rest is used for testing, and so on. In every run the whole pattern set is used, but the mixture of training and test data always changes. We get as many error values as there are partitions of the pattern set. From this a mean error value and a confidence interval is calculated which are an estimation of the error of the classifier created from the whole data set.

4.1 The Wisconsin Brest Cancer (WBC) Data Set

The WBC data set is available from the machine learning repository at ftp://ftp.ics.uci.edu./pub/machine-learning-databases. It is a database that was provided by W.H. Wolberg from the University of Wisconsin Hospitals, Madison. The data set contains 683 cases. In the following some informations and statistics about the WBC data set are described.

Number of input features	9
Names of input features	clumb_thickness uniformity_of _cell_size uniformity_of_cell_shape marginal_adhesion single_epithelial_cell_size bare_nuclei bland_chromatin normal_nucleoli mitoses
Minimum value	1.00

Maximum value	10.00
Defined minimum value	0.00
Defined maximum value	11.00
Number of classes:	2
Names of classes	malign benign
Number of cases:	683

No.	Features	Mean	Std. Deviation
1	clumb_thickness	4.44	2.82
2	uniformity_of _cell_size	3.15	3.07
3	uniformity_of_cell_shape	3.22	2.99
4	marginal_adhesion	2.83	2.86
5	single_epithelial_cell_size	3.23	2.22
6	bare_nuclei	3.54	3.64
7	bland_chromatin	3.45	2.45
8	normal_nucleoli	2.87	3.05
9	mitoses	1.60	1.73

malign: 239 cases (encoded as class 0)
benign: 444 cases (encoded as class 1)

Correlation Table

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**class**
1	1.00	0.64	0.65	0.49	0.52	0.59	0.55	0.53	0.35	-0.71
2		1.00	0.91	0.71	0.75	0.69	0.76	0.72	0.46	-0.82
3			1.00	0.69	0.72	0.71	0.74	0.72	0.44	-0.82
4				1.00	0.59	0.67	0.67	0.60	0.42	-0.71
5					1.00	0.59	0.62	0.63	0.48	-0.69
6						1.00	0.68	0.58	0.34	-0.82
7							1.00	0.67	0.35	-0.76
8								1.00	0.43	-0.72
9									1.00	-0.42

4.2 The Experiment

The goal of this experiment is to test the pruning process and to do a cross validation in order to get some information about the validity of the classifier the classifier that can be compared with other methods that are discussed in [NAUCK/KRUSE 98b]. The experiment is devided into three parts.

4.2.1 Automatically Create a Classifier and then Prune It

The classifier is created with the ‘automatically determine the rule base’ option in order to find a rulebase that covers the whole data set. After training the classifier the pruning feature automatically performs four pruning methods one after the other.

The Parameter Settings	
Training data file	wbc.dat
Number of fuzzy sets	2
Type of fuzzy sets	triangular
Aggregation function	maximum
Size of the rule base	automatically determined
Rule learning procedure	best per class
Fuzzy set constraints	- keep relative order - always overlap
Rule weights	not used
Learning rate	1
Validation	no validation
Stop control	- Maximum number of epochs = 500 - Minimum number of epochs = 0 - Number of epoches after optimum = 100 - Admissible classification errors = 0

Create the Classifier

The main menu entry ‘Classifier|Create Classifier’ is used. The following box shows an excerpt of the log file.

```
% NEFCLASS log file file created by NEFCLASS-J 1.0 (c) Ulrike Nauck, Braunschweig, 1999
% This file was created at January 1, 1999 10:04:58 PM GMT+00:00

Training data: C:\VisualCafePDE\projects\VC_NEFCLASS\wbc.dat
Training for at most 500 cycles and for at least 0 cycles.
Continue for 100 cycles after a local optimum was found.
Learning will stop at 0 misclassifications.
There is no validation.
Parameter: LearningRate = 1.0, Fuzzy set constraints:
keep order, must overlap,
rule weights are not used.
The number of rules will be determined automatically.
Each variable uses 2 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.

Starting the training process using 100.0% of all cases.
Searching for rules in the training data...
Performance on training data (100.0% of all cases):
...
135 possible rules found. Now determine the optimal consequents.
...
Performances of Rule Candidates per Class
...
Selection of consequents is complete.
135 rules found in the data:
...
This rule base covers all patterns.
Training Fuzzy Sets.

         Current Result                                              Best Result
Cycle    wrong          error                        cycle    wrong          error
1                27                   118.371901              1              27
                                                                             118.371901
...              ...                  ...                                    ...      ...
                                                                                      ...

Training fuzzy sets stopped after epoch 128
Best classifier was found in cycle 27 (30 misclassfications, error = 62.112250)
Restoring best solution.

Performance on training data (100.0% of all cases):
683 patterns, 30 misclassifications (error = 62.453891)
```

- After rule learning the rule base ends up with
  - 55 rules of the kind:

if	clumb_thickness is small	and
	uniformity_of _cell_size is small	and
	uniformity_of_cell_shape is small	and
	marginal_adhesion is small	and
	single_epithelial_cell_size is small	and
	bare_nuclei is small	and
	bland_chromatin is small	and
	normal_nucleoli is small	and
	mitoses is small	**then** benign

- After fuzzy set training the classification result is:
  - 30 misclassifications
  - error = 62.453891

Prune the Rule Base of the Classifier

A classifier is considered to be “better” than another classifier if the number of misclassifications or the error value are smaller. The number of misclassifications determine the classification performance, and the error (sum of squared differences between targets and outputs) measures the ambiguity of the classifications. The pruning process reduces the number of rules to five. The number of misclassifications are the same (30). The following box shows an excerpt of the log file.

```
Rule learning will not be invoked.
Each variable uses 2 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.
This is an automatic pruning cycle.

Starting to prune the classifier with 55 rules.
Performance on training data (683 patterns) before pruning:
30 misclassifications (error = 62.453891)
...
Prune variables by correlation...
Prune rules by classification frequency...
Prune linguistic terms by minimum frequency...
Prune linguistic terms by fuzzy set support.
Further pruning cannot improve the classfier.
Final pruning result:
The rule base contains 5 rules.
Performance on training data (683 patterns):
32 misclassifications (error = 53.190468).
```

- After pruning the rule base ends up with 5 rules as follows:
  - if uniformity_of _cell_size is small and uniformity_of_cell_shape is small and bare_nuclei is small and then benign
  - if uniformity_of _cell_size is large and uniformity_of_cell_shape is small and bare_nuclei is large and then malign
  - if uniformity_of _cell_size is small and uniformity_of_cell_shape is small and bare_nuclei is large and then malign
  - if uniformity_of _cell_size is large and uniformity_of_cell_shape is large and bare_nuclei is small and then malign
  - if uniformity_of _cell_size is large and uniformity_of_cell_shape is large and bare_nuclei is large and then malign
- The classification result is:
  - 32 misclassifications
  - error = 53.190468

Performance on Training Data

	Predicted Class							
	0		**1**		**n.c.**		**sum**	
0	**224**	32.80%	15	2.20%	0	0.00%	239	34.99%
1	17	2.49%	**427**	62.52%	0	0.00%	444	65.01%
sum	241	35.29%	442	64.71%	0	0.00%	683	100.00%

0:	malign	Correct:	**651**	(95.31%)
1:	benign	Misclassified:	32	(4.69%)
n.c.:	not classified			

Conclusion

The pruning methods reduced the number of rules and not all variables are used now. So the rules became shorter. The error value was lowered from 62.45 to 53.19. On the other hand the number of misclassifications increased. But now the rule base can be interpreted well. So with a next experiment it is tried to improve this result.

4.2.2 Use a Certain Number of Rules to Create the Classifier

We learned from the last experiment that the rulebase can be reduced to 5. We will therefore try to create a classifier that is limited in its number of rules. We use the ‘best per class’ option so that the number of rules should be 6. This experiment is devided into two parts because the pruning methods do not reset the fuzzy sets after each step. So in the first step we create and the prune the classifier and in a second step we reset the fuzzy sets, train them and then prune the rule base again in order to reduce the influence of the fuzzy set training on the pruning process.

The Parameter Settings	
The same as in experiment 4.2.1	
Size of the rule base	6
The same as in experiment 4.2.1	

Create the Classifier

The main menu entry ‘Classifier|Create Classifier’ is used. The following box shows an excerpt of the log file.

```
Training data: C:\VisualCafePDE\projects\VC_NEFCLASS\wbc.dat
Training for at most 500 cycles and for at least 0 cycles.
Continue for 100 cycles after a local optimum was found.
Learning will stop at 0 misclassifications.
There is no validation.
Parameter: LearningRate = 1.0, Fuzzy set constraints:
keep order, must overlap,
rule weights are not used.
The rule base will consist of 6 rules using the best 3 rules for each class.
Each variable uses 2 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.
...
WARNING: This rule base covers only 81% of all training patterns,
but this may improve during fuzzy set learning.
Training Fuzzy Sets.

        Current Result                                  Best Result
Cycle   wrong          error                    cycle   wrong          error
1              148         226.057851           1              148              22
                                                                                6 .
                                                                                05
                                                                                78
                                                                                51
...            ...         ...                                         ...      ...
                                                                                ...
Training fuzzy sets stopped after epoch 500
Best classifier was found in cycle 499 (27 misclassfications, error = 108.876753)
Restoring best solution.
```

- After rule learnig the rule base ends up with 6 rules
- After fuzzy set training the classification result is:
  - 27 misclassfications
  - error = 108.876753

Prune the Rule Base of the Classifier

The main menu entry ‘Classifier|Prune Classifier’ is used. The pruning process recudces the number of rules to two. The following box shows an excerpt of the log file

```
Fuzzy sets will be trained.
This is an automatic pruning cycle.

Starting to prune the classifier with 6 rules.
Performance on training data (683 patterns) before pruning:
27 misclassifications (error = 108.876753)
...
Prune variables by correlation...
Prune rules by classification frequency...
Prune linguistic terms by minimum frequency
Pruning could not improve the classifier.
The previous version is restored.
Prune linguistic terms by fuzzy set support.
Further pruning cannot improve the classfier.

Final pruning result:
The rule base contains 2 rules.

Performance on training data (683 patterns):
25 misclassifications (error = 146.242560).
```

- After pruning the rule base ends up with two rules:
  - if clumb_thickness is large and uniformity_of _cell_size is small and uniformity_of_cell_shape is small and bare_nuclei is small and bland_chromatin is small and mitoses is small and then benign
  - if clumb_thickness is large and uniformity_of _cell_size is large and uniformity_of_cell_shape is large and bare_nuclei is large and bland_chromatin is large and mitoses is large and then malign

- The classification result is
  - 25 misclassifications
  - error = 146.242560

Conclusion

The number of rules and the number of misclassifications is really good but the error value increased a lot. This means that the classification became less crisp. Furthermore it would be desirable to get shorter rules. So this experiment is continued with a reset of the fuzzy sets and training and pruning again.

Reset and Train the Fuzzy Sets

The main menu entry ‘Classifier|Reset Fuzzy Sets’ and ‘Classifier|Train fuzzy Sets Only’ are used. The following box shows an excerpt of the log file.

```
Training data: C:\VisualCafePDE\projects\VC_NEFCLASS\wbc.dat
Training for at most 500 cycles and for at least 0 cycles.
Continue for 100 cycles after a local optimum was found.
Learning will stop at 0 misclassifications.
There is no validation.
Parameter: LearningRate = 1.0, Fuzzy set constraints:
keep order, must overlap,
rule weights are not used.
There are 2 initial rules:
...
Rule learning will not be invoked.
Each variable uses 2 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.

Starting the training process using 100.0% of all cases.
Training Fuzzy Sets.

        Current Result                                  Best Result
Cycle   wrong       error                       cycle   wrong       error
1            431          566.603306            1            431          56
                                                                          6 .
                                                                          60
                                                                          33
                                                                          06
...          ...               ...                                  ...   ...
                                                                          ...
Training fuzzy sets stopped after epoch 500
Performance on training data (100.0% of all cases):
683 patterns, 25 misclassifications (error = 165.652780)
```

- After fuzzy set training the classification result is:
    - 25 misclassifications
    - error = 165.652780

Prune the Rule Base of the Classifier

The main menu entry ‘Classifier|Prune Classifier’ is used. The pruning process recudces the number of variables to two in the first rule and to three in the second rule. The following box shows an excerpt of the log file

```
Fuzzy sets will be trained.
This is an automatic pruning cycle.

Starting to prune the classifier with 2 rules.
Performance on training data (683 patterns) before pruning:
25 misclassifications (error = 165.652780)
...
Prune variables by correlation...
Prune linguistic terms by minimum frequency...
Prune linguistic terms by fuzzy set support.
Further pruning cannot improve the classfier.

Final pruning result:
The rule base contains 2 rules.
...
Performance on training data (683 patterns):
24 misclassifications (error = 64.956353).
```

- After the pruning process the rule base ends up with 2 rules:
  - if uniformity_of_cell_size is small and bare_nuclei is small and then benign
  - if uniformity_of_cell_size is large and uniformity_of_cell_shape is large and bare_nuclei is large and then malign
- The classification result is:
  - 24 misclassifications
  - error = 64.956353

Performance on Training Data

	Predicted Class							
	0		**1**		**n.c.**		**sum**	
0	**226**	33.09%	13	1.90%	0	0.00%	239	34.99%
1	11	1.61%	**433**	63.40%	0	0.00%	444	65.01%
sum	237	34.70%	446	65.30%	0	0.00%	683	100.00%

0:	malign	Correct:	**659**	(96.49%)
1:	benign	Misclassified:	24	(3.51%)
n.c.:	not classified			

Conclusion

This parameter setting offers a good performance. But up to now the classifier is created with the whole data set and the performance is tested by using the same data. So there is a need for a validation procedure in order to find out an error probability for the classifier.

4.2.3 Cross Validation of the Classifier

A 10-fold cross validation is used. We are interested in the validation of the fuzzy set learning. We do not validate the structure learning process because we think that we found a promising rule base during the last two experiments, i.e. we want to know how well a classifier with this rule base will perform.

The Parameter Settings	
Training data file	wbc.dat
Number of fuzzy sets	2
Type of fuzzy sets	triangular
Aggregation function	maximum
Size of the rule base	6 (irrelevant, because rule base is fixed)
Rule learning procedure	best per class (not used)
Fuzzy set constraints	- keep relative order - always overlap
Rule weights	not used
Learning rate	1
Validation	10-fold cross validation
Stop control	- Maximum number of epochs = 500 - Minimum number of epochs = 0 - Number of epoches after optimum = 100 - Admissible classification errors = 0

Train the Fuzzy Sets with Cross Validation

For the validation the fuzzy sets must be reset and the ‘Train Fuzzy Sets Only’ command of the ‘Classifier’ menu has to be used. An excerpt of the log file is given below.

```
Training data: C:\VisualCafePDE\projects\VC_NEFCLASS\wbc.dat
Validation mode is 10-fold cross validation.
Parameter: LearningRate = 1.0, Fuzzy set constraints:
keep order, must overlap,
rule weights are not used.
There are 2 initial rules: ...
Rule learning will not be invoked.
Each variable uses 2 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.
Starting the training process using 10-fold cross validation.
Validation cycle 1 of 10
Training Fuzzy Sets.

        Current Result                                   Best Result
Cycle   wrong          error                      cycle  wrong          error
1               13                19.985655              1              13
                                                                        19.985655
...             ...               ...                                   ...     ...
                                                                                ...
Training fuzzy sets stopped after epoch 336
Best classifier was found in cycle 235 (3 misclassfications, error = 9.516956)
Restoring best solution.

Result of validaton cycle 1:
Training data: 614 patterns, 22 misclassifications (error = 54.433769)
Validation data: 69 patterns, 3 misclassifications (error = 9.516956)
Result of validaton cycle 2:
Training data: 614 patterns, 26 misclassifications (error = 59.920585)
Validation data: 69 patterns, 1 misclassifications (error = 4.312431)
Result of validaton cycle 3:
Training data: 614 patterns, 23 misclassifications (error = 56.715695)
Validation data: 69 patterns, 2 misclassifications (error = 7.237653)
Result of validaton cycle 4:
Training data: 615 patterns, 28 misclassifications (error = 64.975795)
Validation data: 68 patterns, 1 misclassifications (error = 4.463641)
Result of validaton cycle 5:
Training data: 615 patterns, 24 misclassifications (error = 57.010478)
Validation data: 68 patterns, 3 misclassifications (error = 7.427357)
Result of validaton cycle 6:
Training data: 615 patterns, 27 misclassifications (error = 63.789943)
Validation data: 68 patterns, 3 misclassifications (error = 5.379538)
Result of validaton cycle 7:
Training data: 615 patterns, 23 misclassifications (error = 57.333187)
Validation data: 68 patterns, 4 misclassifications (error = 7.028670)
Result of validaton cycle 8:
Training data: 615 patterns, 26 misclassifications (error = 63.663097)
Validation data: 68 patterns, 2 misclassifications (error = 5.015220)
Result of validaton cycle 9:
Training data: 615 patterns, 25 misclassifications (error = 61.532933)
Validation data: 68 patterns, 2 misclassifications (error = 7.874834)
Result of validaton cycle 10:
Training data: 615 patterns, 21 misclassifications (error = 57.834692)
Validation data: 68 patterns, 4 misclassifications (error = 5.605990)
```

- After the final cycle of the validaton process where the whole data set is used the result is:
  - 25 misclassfications
  - error = 64.150063

Performance on Training Data

	Predicted Class							
	0		**1**		**n.c**		**sum**	
0	**225**	32.94%	14	2.05%	0	0.00%	239	34.99%
1	11	1.61%	**433**	63.40%	0	0.00%	444	65.01%
sum	236	34.55%	447	65.45%	0	0.00%	683	100.00%

0:	malign	Correct:	**658**	(96.34%)
1:	benign	Misclassified:	25	(3.66%)
n.c.:	not classified			

Result of the Cross Validation	
Number of patterns	683

Estimated number of misclassifications per pattern for unseen data:	mean = 0.036637
	standard deviation = 0.015091
	n = 10

99% confidence interval for the error estimation:	0.036637 ± 0.012979
Estimated error value	3,7% ± 1,3%

4.3 Discussion of the Result and Comparison to other Approaches

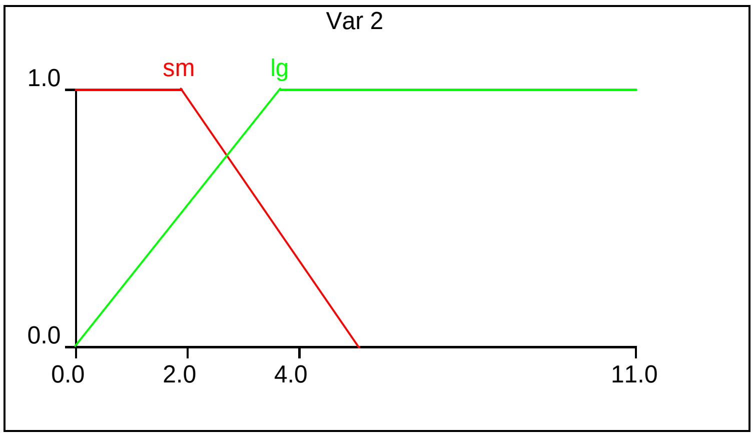
The classifier resulting from the creation process described in the experiments produces classification results wich can be interpreted very well because the rule base consists of only two short rules. Under this constraint an estimated error value of 3,7% ± 1,3% is a really good classification result.

The Rules

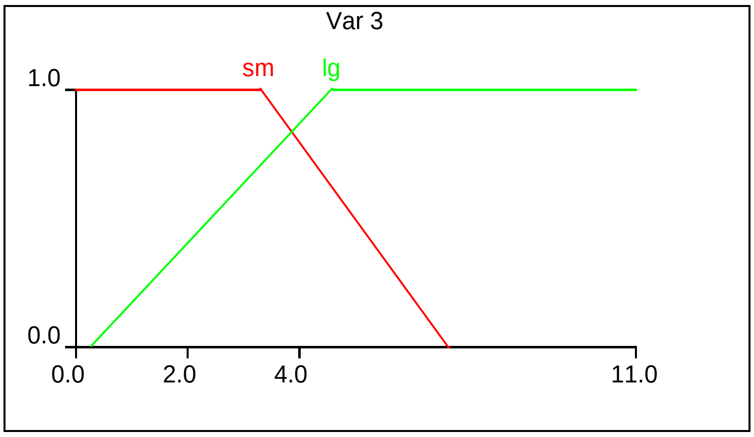
if uniformity_of_cell_size is small and
bare_nuclei is small **then** benign

if uniformity_of_cell_size is large and
uniformity_of_cell_shape is large and
bare_nuclei is large **then** malign

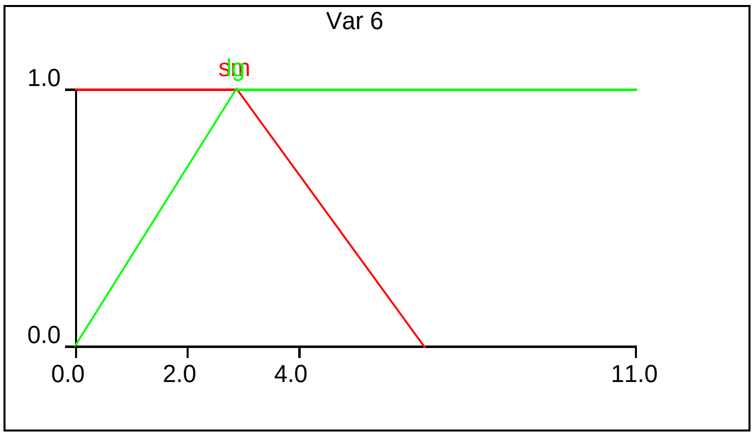
The Fuzzy Sets



Var 2: uniformity_of_cell_size
sm: small
lg: large



Var 3: uniformity_of_cell_shape
sm: small
lg: large



Var 6: bare_nuclei
sm: small
lg: large

The following table [NAUCK/KRUSE 98b] compares NEFCLASS-J to results obtained with other approaches. The classification performance on unseen data is comparable and the classifier is very compact. The error estimates given in the table are either obtained from 1-leave out cross validation, 10 fold cross validation as used in the experiments described above, or from testing the solution once by holding out 50% of the data for a test set.

Model	**Tool**	**Remarks**	**Error (%)**	**Validation**
Discriminant analysis	SPSS	Linear Model 9 variables	3.95	1-leave-out
Multilayer Perceptron	SNNS	Four hidden units, Rprop	5.18	50% test set
Decision tree	C4.5	31 nodes pruned	4.9	10-fold
Rules from decision tree	C45. rules	8 rules using 1-3 variables	4.6	10-fold
NEFCLASS	NEFCLASS-X	2 rules using 5-6 variables	4.94	10-fold
NEFCLASS	NEFCLASS-J	2 rules using 2-3 variables	3.7	10-fold

4.4 Conclusion

For applying a neuro-fuzzy strategy one important aspect should be considered: for whatever reason a fuzzy system to solve a problem is chosen, it cannot be because an exact solution is needed. Fuzzy systems are used to exploit the tolerance for imprecise solutions. Fuzzy systems should be used because they are easy to implement, easy to handle and easy to understand. A learning algorithm to create a fuzzy system from data also should have these features. Under this view the learning and pruning strategies of NEFCLASS are simple and fast heuristics.

The new NEFCLASS tool was presented using the well-known Wisconsin Brest Cancer data set. The new philosophy in pruning the rule base automatically was successfull, so users who have no experience with pruning methods can create a compact interpretable classifier.

The validation option that offers cross validation as well as single test enables the user to compare the quality of the classification results very easily to other approaches. No further computation with statistic tools is needed.

It is also shown that NEFCLASS-J is not a tool for automatic creation of a fuzzy classifer. It supports the user, but it cannot do all the work because a precise and interpretable fuzzy classifier can hardly be found by an automatic learning process. There will always be the trade-off between readability and precision.

5

How to use NEFCLASS-J

In this chapter a ‘guided tour’ through the program is given. The data set used for this is the ‘Iris’ data set. The interface and all of the parameters will be explained. A classifier can be developed, trained and validated without any changes of parameters. But as depicted in the fourth chapter, the user should *work* with the program. With help of the parameters the system is constrained so that a classifier can be developed that fits the problem. But nevertheless the hardcopy pictures of the interface show the default values for the parameters. Some tests how the changing of parameters influences the system are made in Chapter 4.

This chapter of the thesis is not only a description of the GUI of NEFCLASS-J but shall also function as a tutorial which will be released on the Internet. Therefore the reader is adressed directly in the following and quite some effort was made to write a text that can be understood by non-experts, too. The following pages also make use of color. NEFCLASS-J uses distinct colors to help the user to recognize the different dialogs and frames more easily. This chapter uses this color code to stress the relationship to certain parts of the tool.

In addition to the guided tour where the parameters are introduced by an example there is a picture of the main menu of the program. There the page numbers are given where the respective menu item is explained. For information about the installation of NEFCLASS-J please refer to the file readme.txt which is distributed together with the program.

5.1 The Trainig Data Set

The training set is the same as used in the manual of NEFCLASS-PC. So for users of this implementation it might be easier to follow the new outfit of NEFCLASS-J. Also the guided tour will look very much like the descriptions in this manual [UNauck 97].

NEFCLASS-J learns from training data, which must be provided in a pattern file. This manual uses the Iris data which is also distributed in combination with NEFCLASS-J as a concrete example for the explanations. Iris data is perhaps the best known database to be found in the pattern recognition literature. Fisher’s paper [FISHER 36] is a classic in the field and is referenced frequently to this day [DUDA&HART 73]. The three types of Iris flowers can be classified by the length and width of their sepals and petals. So the attribute information is:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
    - Iris Setosa
    - Iris Versicolour
    - Iris Virginica

The data set contains 3 classes of 50 instances each, where each class refers to a type of Iris plant. These are 150 cases of 4 numeric predictive attributes and the class coded as a binary vector of 3 components. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. Here some statistics on the data set are given:

	Min	**Max**	**Mean**	**SD**	**Class Correlation**
sepal length	43	79	584	83	0.7826
sepal width	20	44	305	43	-0.4194
petal length	10	69	376	176	0.9490 (high!)
petal width	1	25	120	76	0.9565 (high!)
No missing attribute values					

Table 5.1: Statisic information about the ‘iris’ data set

In the NEFCLASS-PC implementation there was need to devide the data file into two files, one for training and one for testing. This is not necessary anymore. The new validation feature offers an automatic random division of the file.

5.2 Guided Tour through NEFCLASS-J

Normally a user has a certain problem if he decides to use a program like NEFCLASS. In most cases there is data base, other classification are just tried out and now the question arises if there are other solutions which are faster, better (in accuracy or interpretability) or even easier to handle. So let us specify these tasks.

The Problem

➪ There is data to be classified.
➪ For some reason you decided to do a classification with a neuro-fuzzy system.
➪ You want to know how successful the classifier is, so you want to know the performance of the neuro-fuzzy system.
➪ You want to modify the classifier in order to find out if the performance can be improved.
➪ In the case you have prior knowledge about the data, you want to use it for the creation of the classifier

The Needs

- A data base consisting of patterns with class information,
- A software tool like NEFCLASS-J
  - to create, train, and validate the classifier,
  - to give you graphical and textual displays for interpreting the results,
  - to do some statistics to get prior knowledge about the data.
  -

The following figue shows the structure of the NEFCLASS main menu. In there the page numbers for the descriptions of the features are written next to each menu entry. This is a kind of index for the case that you are looking for some special descriptions.

The tutorial is devided into five parts that deal with the following topics:

1 The Philosophy

2 Create a Project

- Open the project specification dialog
- Open a data file for training
- Edit the labels
- View some statistics
- Create the rule base
- View the rule base
- Learn the fuzzy sets and view the error
- View the fuzzy sets
- Save and close the project

3 Try to Improve the Classifier by Changing the Parameters

- Open a project
- Change the number and type of fuzzy sets
- Change the aggregation function
- Select the interpretation of classification result
- Determine the size of the rule base
- Change the rule learning procedure
- Relearn the rule base
- Determine the constraints for fuzzy sets
- Determine constraints for rule weights
- Determine a learning rate
- Determine the validation procedure
- Determine the parameters for the stop-learning control

4 Use Prior Knowledge

- The rule editor

5 Pruning the Rule Base

- Manual pruning
- Automatic pruning
- Semi-automatic pruning

NEFCLASS-J

Project

- New Project Ctrl+N — pp ▪57, p. 59, **84**
- Open Project Ctrl+O — p ▪67, **84**
- Edit Project Ctrl+E — pp ▪57, p. **84**
- Save Ctrl+S — p ▪67, **84**
- Save As Ctrl+Shift+S — p ▪**85**
- Close Project Ctrl.+Shift+C — p ▪67, **85**

- Load Training Data — p. 61, **85**
- Load Application Data — p. 57, **85**

- Exit NEFCLASS

Classifier

- Create Classifier — pp ▪68, p. 82, **85**
- Create Rule Base Only — p ▪63, **85**
- Train Fuzzy Sets Only — p ▪65, **85**

- Prune Classifier — p. 83, **85**
- Create Classifier and Prune It — p. **85**
- Create Pruned Classifer — p. **85**

- Restore Fuzzy Sets — p. **87**
- Reset Fuzzy Sets — p. 82, **87**

- Classify Training Data — p. **87**
- Classify Appl. Data — p. **87**

- Stop Training Process — p ▪**87**

View

- Error — p. **88**
- Fuzzy Sets — p. 66, **88**
- Statistics Training Data — p. 62-63, **88**
- Statistics Appl. Data — p. **88**

- Hide All
- Show All

Rules

- Edit — p. 64, 82, **88**
- Restore Rule Base — p. **88**
- Export — p. **88**

Help

- Help Topics... — p. **88**
- About ... — p. **88**

5.2.1 The Philosophy

The handling of the created NEFCLASS-System is organized in projects. Whenever a stage of development is reached that should be saved, this can be done as a project. For the creation of a classifier the following steps have to be done:

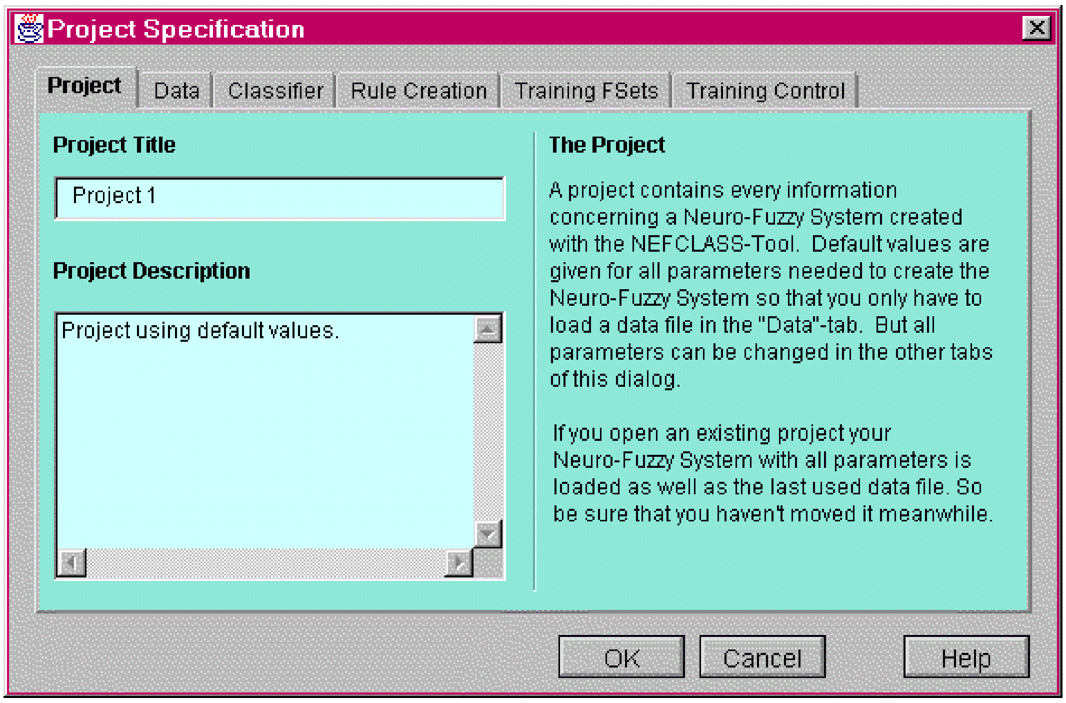
- Parameters to define and constrain the classifiers have to be set. Some of them (e.g. learning rate, aggregation function, stop control, learning procedure) can be set without a data file to be opened. Others like number and type of fuzzy sets depend on the patterns. So the next step is to
- open a data file for training. After the definition of all parameters (default values are set) the
- training of the classifier can be started
  - in two single steps, first the rule learning and in a second step the training of fuzzy sets or
  - in one step.

In every state it is possible to save the project. If a data file is loaded the file name is saved in the project so that it will be loaded automatically when the project is opened again. So be careful with moving your data files. If the classifier is just created additional to the project file a classifier file is saved. The project file is supposed to have the extension '.prj'. It contains the parameter definitions used to create the classifier. The classifier file is supposed to have the extension '.cls'. It contains the rule base and all other informations to describe the classifier. The user only needs to save the project and the files will be created automatically. These files are in ASCII code and can be viewed with any editor. It is possible to handle one project at a time only. So a project has to be closed before another one can be opened.

Furthermore it is possible to load a second data file which can used for testing the network or even classifying unknown data. It can be loaded by 'Project|Open Application Data'. Although it is sortet into the Project menu it is not neatly connected to it. Another project can be loaded to classifiy this data. It is not saved with any project as it is done with the training data.

The data files cannot be closed explicitely. They are simply replaced when a new file is loaded. NEFCLASS will do a consistency check if the new data file fits the classifier. If not a message is given that the project must be closed and a new one has to be created. With closing the project the data files are also closed.

The handling of a project is controlled by a dialog which is opend when in the main menu 'Project|New Project', 'Project|Edit Project', or 'Project|Open Project' is selected. In this dialog all parameters can be set.



Project Title

Project 1

Project Description

Project using default values.

The Project

A project contains every information concerning a Neuro-Fuzzy System created with the NEFCLASS-Tool. Default values are given for all parameters needed to create the Neuro-Fuzzy System so that you only have to load a data file in the "Data"-tab. But all parameters can be changed in the other tabs of this dialog.

If you open an existing project your Neuro-Fuzzy System with all parameters is loaded as well as the last used data file. So be sure that you haven't moved it meanwhile.

The six tabs of the dialog have different colours. If further dialogs are opened concerening the matter of a tab the same colour for this dialog is used. This way a connection to the different steps of creating a classifier is given. The topics of the tabs and the assigned colours are:

- **Project** (emerald green)
  - features of the tab:
  - project name can be set
  - project description can be set
- **Data** (blue)
  - features of the tab:
  - load data file for training
  - invoke dialog for changing variable and class names
  - invoke window for statistics
  - connected dialogs:
  - progress dialog for file loading
- **Classifier** (yellow)
  - features of the tab:
  - set the numbers and types of fuzzy sets
  - set the aggregation function
  - edit the rule base
  - connected dialogs:
  - view the fuzzy sets

- **Rule Creation** (orange)
  - features of the tab:
  - set the size of the rule base
  - select the rule learning procedure
  - choose if the the rule base need to be relearned
  - connected dialogs:
  - view the learning process in a progress dialog

- **Training FSets** (green)
  - features of the tab:
  - set the constraints for the fuzzy sets
  - set the constraints for the rule weights
  - set the learning rate
  - connected dialogs:
  - view the training progress, errors are drawn in a graph

- **Training Control** (pink)
  - features of the tab:
  - set the validation procedure
  - set the and the stop control parameters

The project specification dialog has to be closed before other dialogs or the main menu is enabled again (except the dialogs invoked from one of the tabs). This may sometimes be uncomfortable but it helps to do a consistency check of the parameters. If some settings are not valid the dialog cannot be closed and the user has to look for the problem. This way inconsistent projects are avoided.

In the following tutorial you will be guided through the process of creating and improving a classifier. The description of the exercises is written in boxes. For these boxes the same code of colours is used as for the windows, dialogs and tabs in the program. E.g. if an exercise deals with the handling of training data the describing boxes are blue.

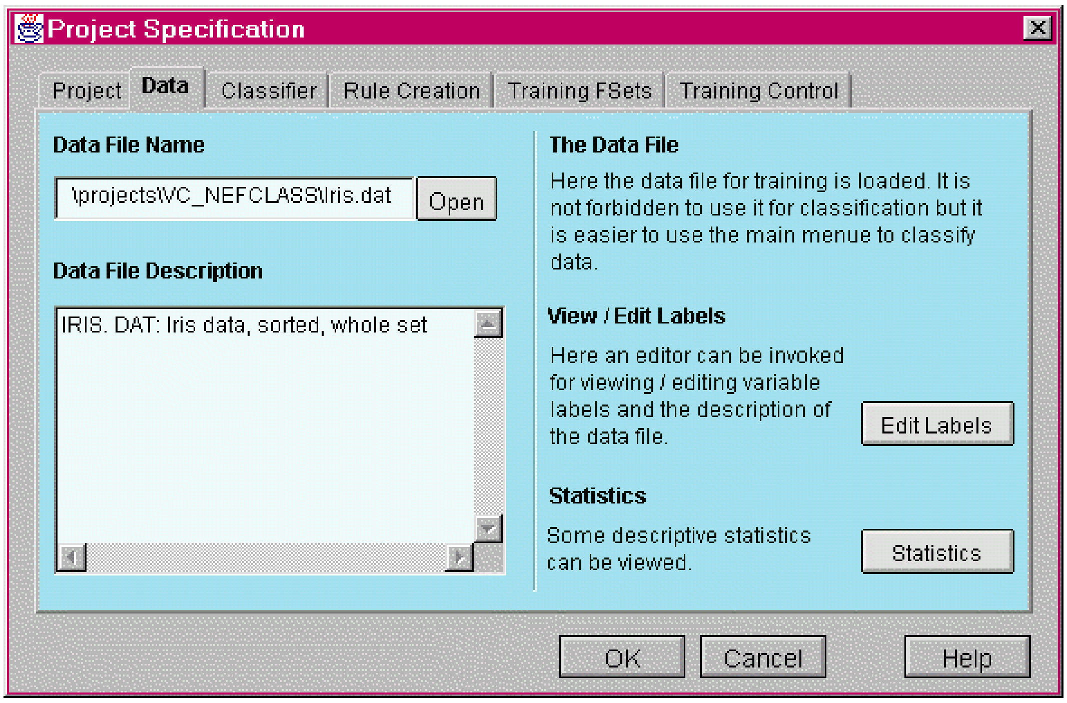
5.2.2 Create a Project

In this part of the guided tour a classifier using default values is created. The only change that will be trained here is the renaming of the variable and class labels.

- **Open the 'Project Specification' Dialog**

  In this dialog every parameter of the classifier can be set. When creating a new project this dialog is opened automatically.
  - Use 'Project|New Project' in the main menu to invoke the 'Project Specification' dialog.
  - In the *Project* tab insert as a title "Project1" and
  - if you want you can write a description.

- **Open a Data File for Training**
  - choose the *Data* tab



Data File Name

\projects\VC_NEFCLASS\Iris.dat

Data File Description

IRIS. DAT: Iris data, sorted, whole set

The Data File

Here the data file for training is loaded. It is not forbidden to use it for classification but it is easier to use the main menue to classify data.

View / Edit Labels

Here an editor can be invoked for viewing / editing variable labels and the description of the data file.

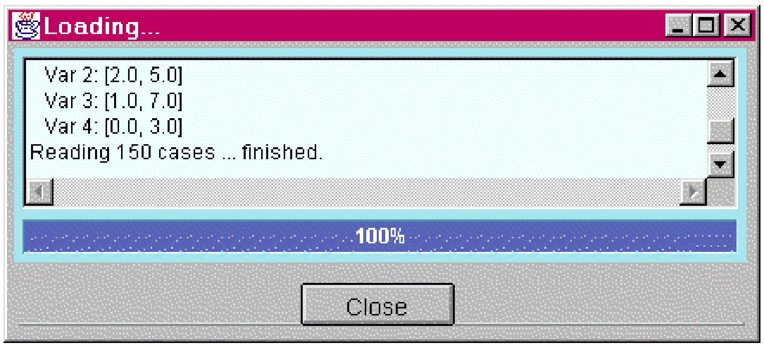
Statistics

Some descriptive statistics can be viewed.

  - select the 'Open' button
  - select the file 'iris.dat' from the directory to where you stored it.

File Name for Training Data

If the chosen file is not formatted in NEFCLASS style a message box opens with this information. The file name can*not* be written in the text field in the left of the button, it is only meant for display. The progress of opening the file is shown in a message box where also some informations about the data are displayed.



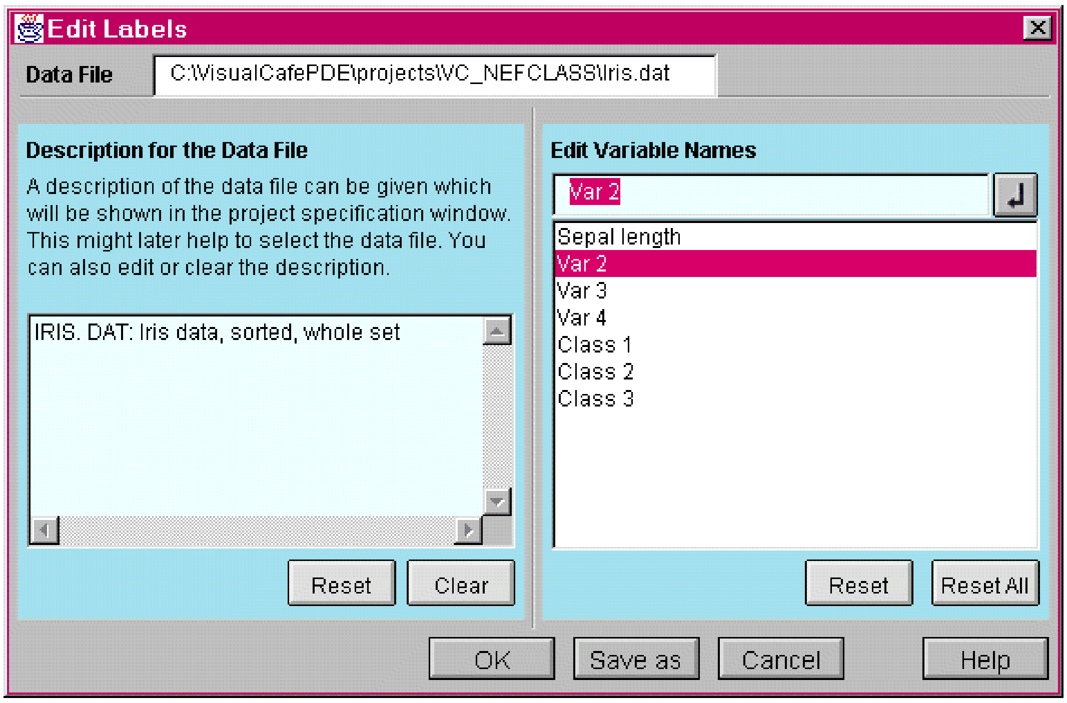
```
  Var 2: [2.0, 5.0]
  Var 3: [1.0, 7.0]
  Var 4: [0.0, 3.0]
Reading 150 cases ... finished.
```

Another possibility to load a data file is to choose 'Project|Load Training Data'. An open dialog will also be invoked. But this is only possible when the project specification dialog is closed. After the file is opended the filename and if existing a description will be inserted into the *Data* tab of the 'Project Specification' dialog when opened the next time.

- **Edit the Labels**
  - Choose the 'Edit Labels' button.

Dialog for Changing Variable and Class Names

Another dialog will be invoked where the names of the variables and the classes can be changed. These new names are saved in the data file and used from now on. If a data file has no labels yet default names are used.



C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat

A description of the data file can be given which will be shown in the project specification window. This might later help to select the data file. You can also edit or clear the description.

IRIS. DAT: Iris data, sorted, whole set

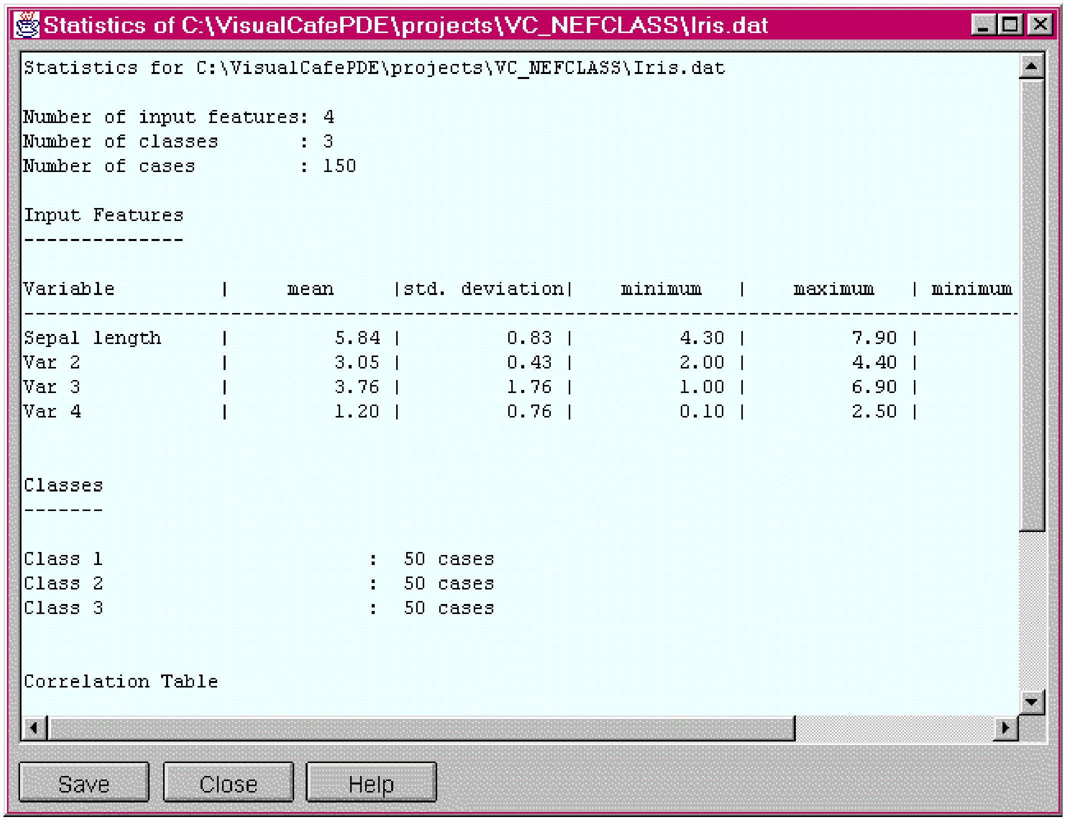
Var 2

Sepal length
Var 2
Var 3
Var 4
Class 1
Class 2
Class 3

In the text area on the left a description of the data file (the name is displayed in the first line) can be given. On the right a list of names appear. First the variable names are listed and then the class names. In the example default names had been given and the first name (sepal length) was just changed. With clicking on the next name to be changed it appears in the line above the list. There the changes can be written. A click on the 'enter' button next to this line will insert the new name into the list. While you do not leave this dialog with ok, you can reset the changes. The 'save as' button gives you the chance to create a new data file if you do not want the original data to be changed.

- So click the first line of the list (Var 1) and change it to 'sepal length'. You can do more if you want to. The data is described in section 5.2 of this chapter.
- Leave the dialog with 'OK'.

- **View Some Statistics**
  - Click the 'Statistics' button of the *Data* tab.


Statistics of C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat
Save
Close
Help

```
Statistics for C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat

Number of input features: 4
Number of classes       : 3
Number of cases         : 150

Input Features
--------------

Variable          |     mean     |std. deviation|    minimum    |     maximum    | minimum
---------------------------------------------------------------------------------------------
Sepal length      |         5.84 |          0.83 |          4.30 |          7.90 |
Var 2             |         3.05 |          0.43 |          2.00 |          4.40 |
Var 3             |         3.76 |          1.76 |          1.00 |          6.90 |
Var 4             |         1.20 |          0.76 |          0.10 |          2.50 |


Classes
-------

Class 1                        :   50 cases
Class 2                        :   50 cases
Class 3                        :   50 cases


Correlation Table
```

View Statistics

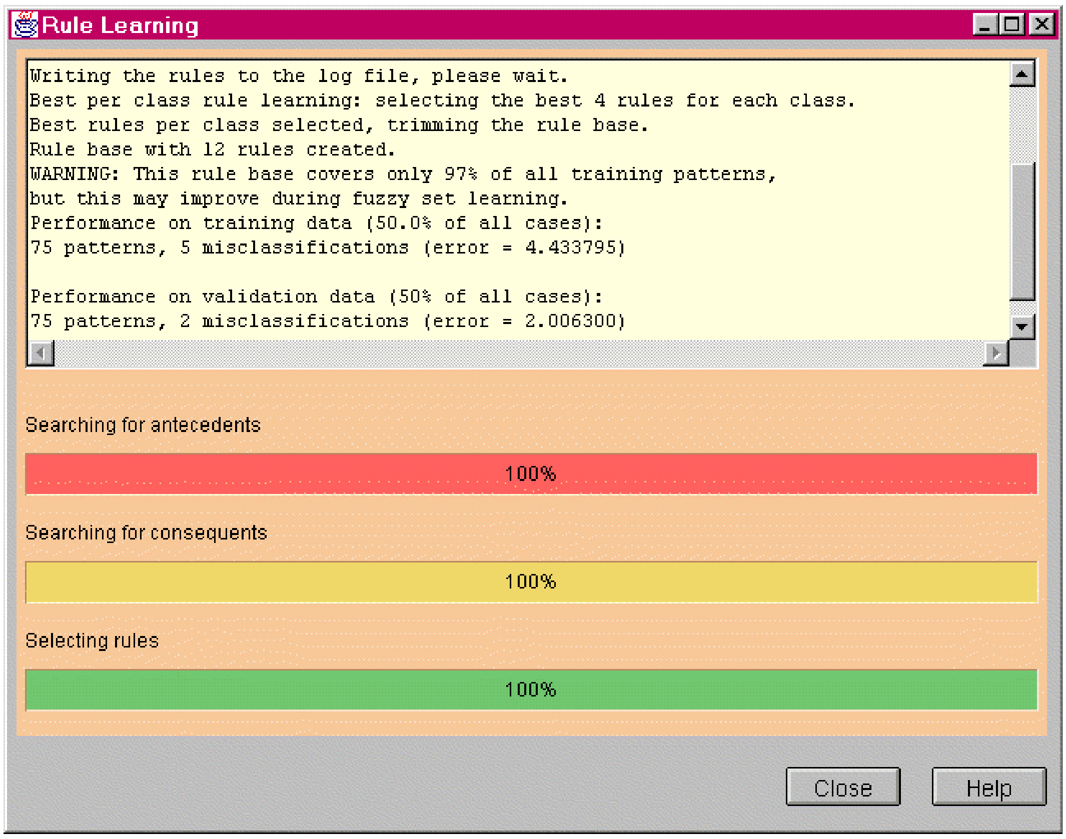
A new window opens and some descriptive statistics and correlations are displayed. As you can see the changes you did with the labels are transferred. The text can be edited in this window and also be saved in an ASCII file. The extension '.txt' will be suggested for the file name you give in the save dialog.

This is only one possibility to view statistics. If a data file is loaded the statistics window can also be invoked by 'View|Statistics Training Data'.

- **Create the Rule Base**
  - Close the project specification dialog with 'OK'.
  - Select the 'Classifier|Create Rule Base Only' item of the main menu.

Learn the Rule Base

An orange window opens (orange concerning rules) where classification results are displayed.



```
Writing the rules to the log file, please wait.
Best per class rule learning: selecting the best 4 rules for each class.
Best rules per class selected, trimming the rule base.
Rule base with 12 rules created.
WARNING: This rule base covers only 97% of all training patterns,
but this may improve during fuzzy set learning.
Performance on training data (50.0% of all cases):
75 patterns, 5 misclassifications (error = 4.433795)

Performance on validation data (50% of all cases):
75 patterns, 2 misclassifications (error = 2.006300)
```

Searching for antecedents

100%

Searching for consequents

100%

Selecting rules

100%

The text field gives information about the creation of rules, the trimming of the rule base, the rule learning procedure used for this, number of patterns, misclassifications and so on.
Furthermore three bars display the progress of
- searching for antecedents,
- searching for consequent and
- selecting rules

  - Close the window now or minimize it. It is possible to keep it open but you will not recognize much when more windows are opened.

- **View the Rule Base**.
  - Select ‘Rules|Edit’ from the main menu.

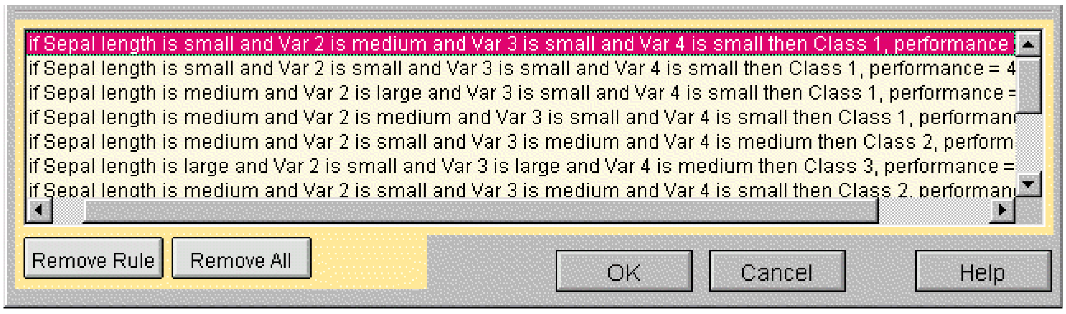
View the Rule Base

A yellow dialog is invoked.Here the color yellow (classifier) is chosen instead of orange (rules) because the dialog is normally used to edit the rule base and insert prior knowledge in form of rules. So it is for defining the classifier before the learning process. But for to edit a rule base it has to be displayed and so we can use this dialog for a glance on the rules.

Rule numbers precede the rules. The rules are followed by:
- rule weights written in brackets if used (should not be done normally) and
- the performance of the rule.

The default number of rules specified in the project was ‘10’, but we also selected ‘best per class’ as rule learning procedure. So the number is rounded up to 12 rules.

This dialog has to be closed before anything else can be done.

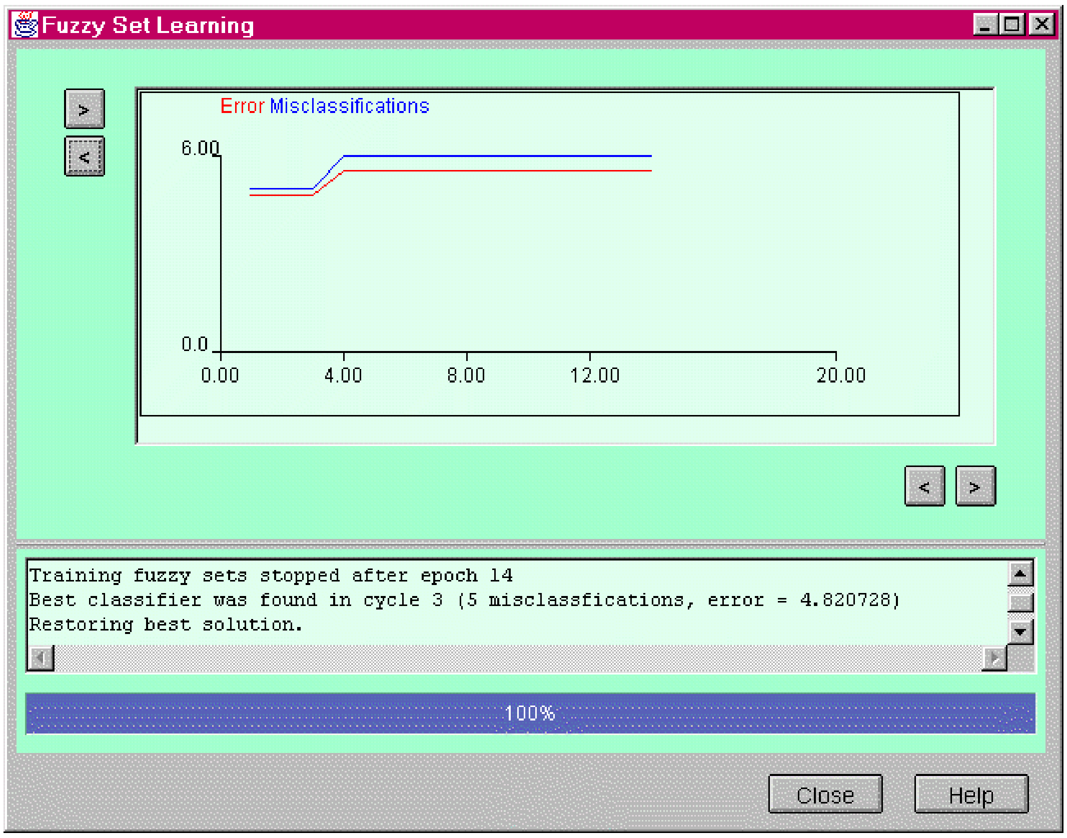

```
if Sepal length is small and Var 2 is medium and Var 3 is small and Var 4 is small then Class 1, performance
if Sepal length is small and Var 2 is small and Var 3 is small and Var 4 is small then Class 1, performance = 4
if Sepal length is medium and Var 2 is large and Var 3 is small and Var 4 is small then Class 1, performance =
if Sepal length is medium and Var 2 is medium and Var 3 is small and Var 4 is small then Class 1, performan
if Sepal length is medium and Var 2 is small and Var 3 is medium and Var 4 is medium then Class 2, perform
if Sepal length is large and Var 2 is small and Var 3 is large and Var 4 is medium then Class 3, performance =
if Sepal length is medium and Var 2 is small and Var 3 is medium and Var 4 is small then Class 2, performan
```

- **Learn the Fuzzy Sets and the View the Error**
  - Select the ‘Project|Train Fuzzy Sets Only’ menu item

Learn the Fuzzy Sets and View the Progress

A green window (concerning fuzzy set learning) is openend while the fuzzy sets are learned.

In the upper part the error and the misclassifications are displayed. The buttons in the left and under the display can be used for changing the the scale of the graph. The text field shows information about the learning process, misclassifications and the error.



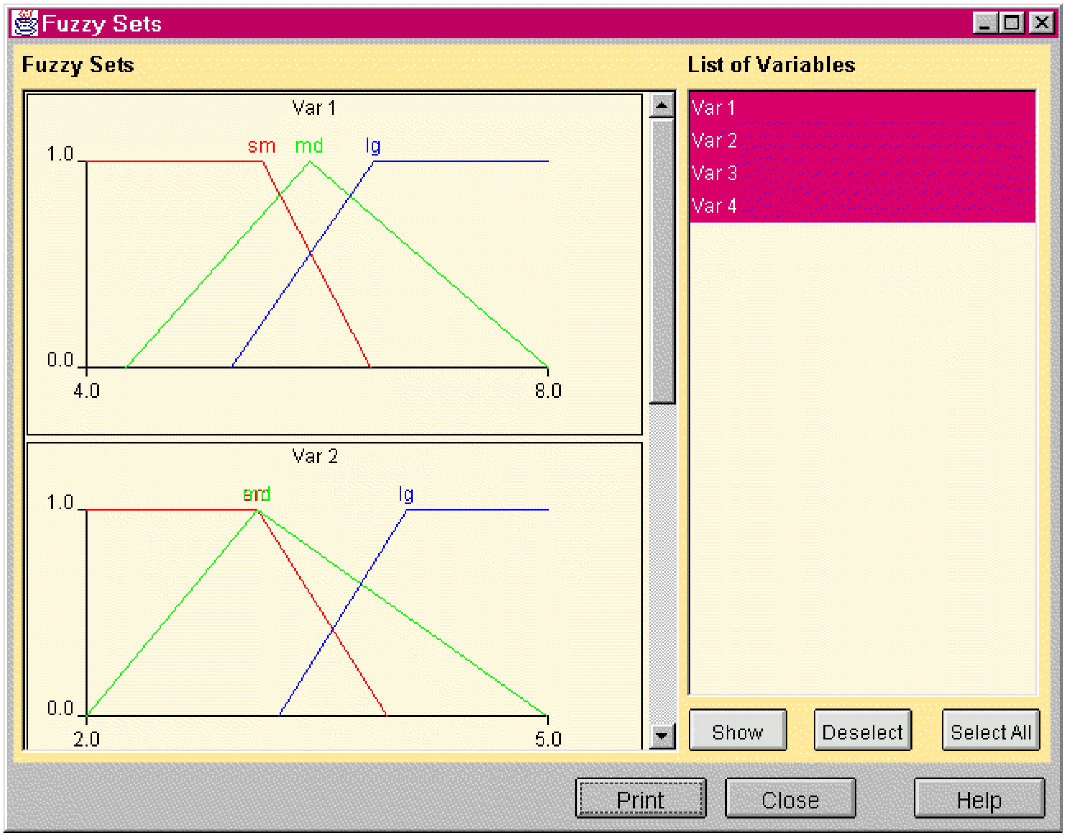
```
Training fuzzy sets stopped after epoch 14
Best classifier was found in cycle 3 (5 misclassfications, error = 4.820728)
Restoring best solution.
```

- close or minimize the window.

- **View the Fuzzy Sets.**
  - select ‘View|Fuzzy Sets’ from the main menu.

View the Fuzzy Sets

The fuzzy sets can be viewed now. This is a yellow vindow again because the fuzzy sets define the classifier. The fuzzy sets can be viewed once a data file for training is loaded. There are default values of the project which define the fuzzy partitions so that a graphical display can be created.



As we will see in the next subsection the default value for the number of fuzzy sets is ‘3’ and the default type is ‘triangular’. So we can view in this window one graph for every variable where the domain is devided into 3 triangular fuzzy sets. Because of the learning process they are not symmetric and evenly distributed anymore.

  - If you are only interested in some variables you can select them from the list on the right
  - The graphs shown on the left can be printed (3 graphs per page).

- **Save and Close the Project**.

> **Save and Close the Project**.
>
> A project can be saved by the ‘Project|Save’ or ‘Project|Save As’ menu option. If an unnamed project should be saved by the option Project|Save the ‘project title’ given in the *Project* tab of this dialog is taken as the file name for the project. The project and the class file (if existing) are automatically created with extensions ‘.prj’ and ‘.cls’. If no title is given a ‘save as’ dialog will be opended so that a project name can be given.
> These files are ASCII files and can be viewd with any editor.

  - select ‘Project|Save’
  - select ‘Project|Close Project’

5.2.3 Try to Improve the Classifier by Changing the Parameters

In the following all parameters which can be set in the ‘project specification’ dialog are described but only some parameters will be changed.

- **Open a Project**
  - select ‘Project|Open Project’ from the main menu.
  - select ‘Project1’.prj

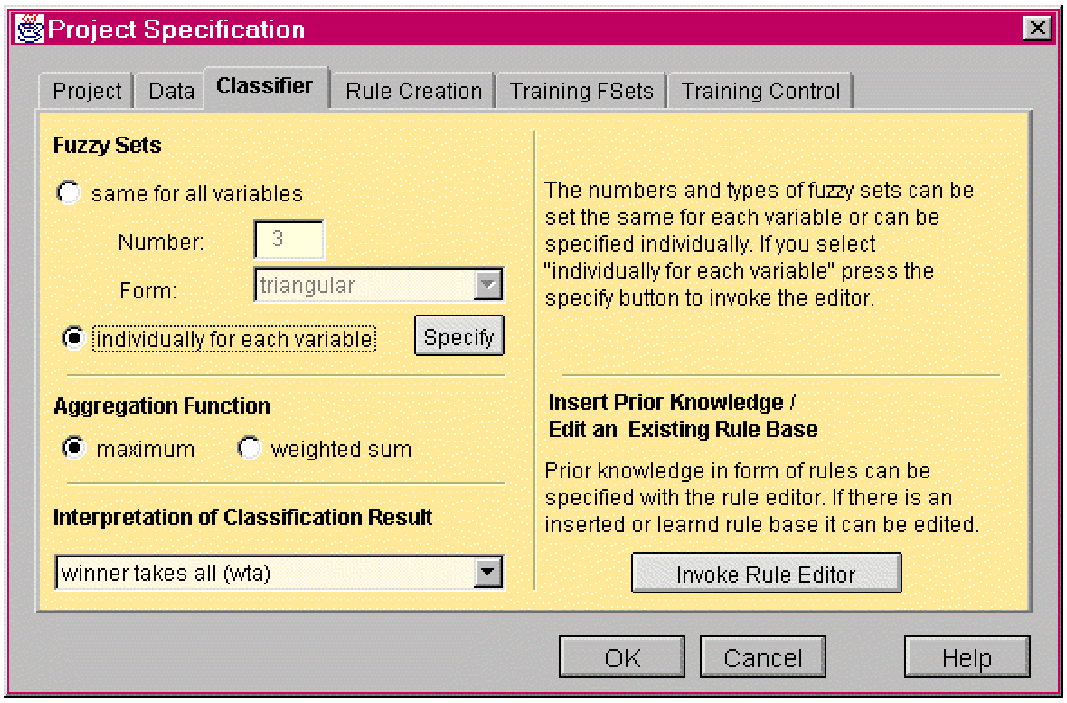
> **Open a Project**
>
> The ‘project specification’ dialog is invoked: The training data file that was loaded during the first part of this tutorial is just loaded.

  - select the *Data* tab
  - view the file name and description given there.

> If there is no file name given something went wrong in the first section of the tutorial.

  - if no file name is given please open a training data file now.

○ select the *Classifier* tab



Fuzzy Sets: [✓ individually for each variable]; Number: 3; Form: triangular

Aggregation Function: [✓ maximum] weighted sum

Interpretation of Classification Result: winner takes all (wta)

The numbers and types of fuzzy sets can be set the same for each variable or can be specified individually. If you select "individually for each variable" press the specify button to invoke the editor.

Prior knowledge in form of rules can be specified with the rule editor. If there is an inserted or learnd rule base it can be edited.

- **Change the Number and Type of Fuzzy Sets**

Number and Type of Fuzzy Sets

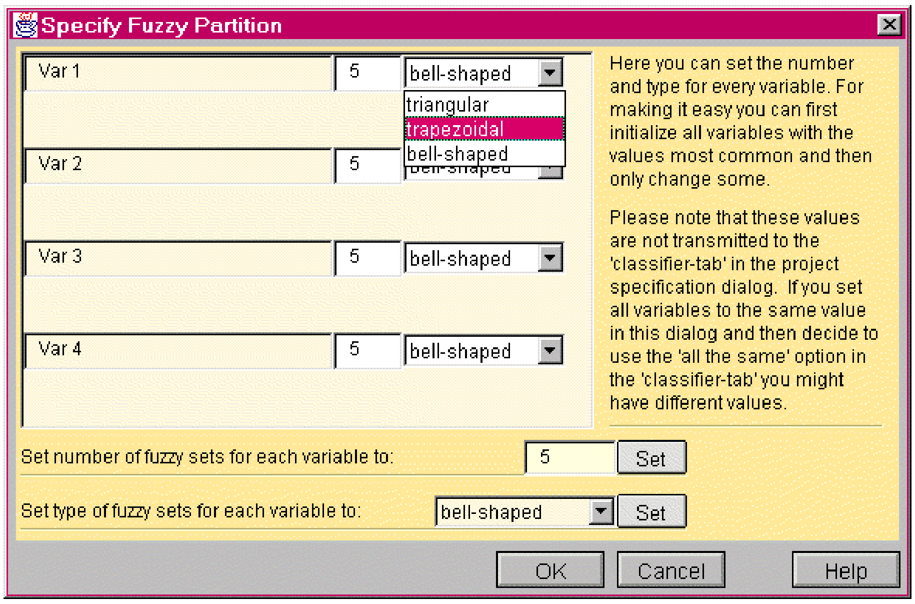
For each variable a number of fuzzy sets (a fuzzy partitioning) has to be defined. By default three fuzzy sets are defined which automatically get names The membership functions are equally distributed on the domain of the respective input feature. There are three types that can be selected (triangular, trapezoidal and bell-shaped). As default the triangular membership function is used where the leftmost and rightmost functions are shouldered. You have the following options:

- same for all variables
- individualy for each variable

○ select individually
○ click the specify button

If no data file is loaded this option cannot be selected. So if you have problems one day, remember to check the *Data* tab for a data file

A dialog opens where the variables and their fuzzy set types and numbers are listed. Here the number and type of fuzzy set for each variable can be set.



Here you can set the number and type for every variable. For making it easy you can first initialize all variables with the values most common and then only change some.

Please note that these values are not transmitted to the 'classifier-tab' in the project specification dialog. If you set all variables to the same value in this dialog and then decide to use the 'all the same' option in the 'classifier-tab' you might have different values.

Set number of fuzzy sets for each variable to: 5

Set type of fuzzy sets for each variable to: bell-shaped

For making it easy all variables can be initialized with the values most common for that there is only need to make some changes. You can do this in the two lines on the bottom of the yellow area. Write the number and/or select the type. If you click the ‘Set’ button of the specified feature the list above will be updated.

The values set there are not transmitted to the *Classifier* tab of the project specification dialog. When the ‘set all’ option in this dialog was used and then you decide not to define them individually and therefore select the ‘all the same’ option in the *Classifier* tab you might have different values.

- set all variables to ‘5 fuzzy sets’ and ‘bell-shaped’
- set the first variable to triangular
- leave the dialog with ‘OK’
- select ‘same for all variables’ in the *Classifier* tab

As you can see there the old values are set. The individual definition does not influence the ‘same for all option’. But imagine that you perhaps want to define hundred variables individually. 92 have to be set to ‘5 fuzzy sets’ and only seven of them to ‘8 fuzzy sets’. You would be very angry to set them one by the other. So it is necessary to have a possibility to initalize all and then only change some.

- set the number of fuzzy sets to ‘5 for all variables’
- the triangular type is ok.

- **Change the Aggregation Function**

The Aggregation Function

Let us use the network view of the classifer for this explanation. Many rule units (hidden layer) can be connected to one class unit (output layer), i.e. many rules can have the same class in their consequent. The output of a rule unit is a degree of fulfilment. The rule units send their outputs to the class units. The aggregation function of a class unit (net_u) calculates a value from the output values of the rule units that will be the output of the class unit .

- 'Weighted Sum'
  A weighted sum of the rule unit output values is calculated. If you decide to use rule weights this will be a weighted mean.
- 'Maximum'
  The maximum of the rule unit output values will be used (eventually multiplied by an optional rule weight).

  - the selection of 'maximum' is ok.

- **Select the Interpretation of Classification Result**

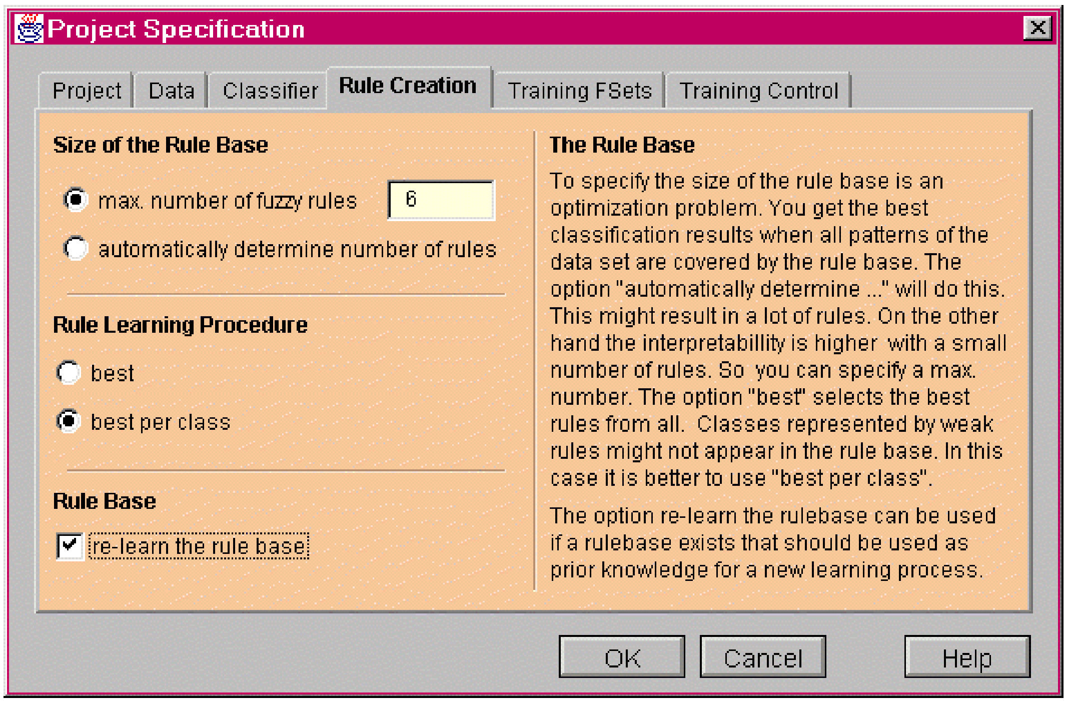
Interpretation of Classification Result

The output values of the class units calculated by the aggregation function need not to give a clear classification decision if the propagated pattern belongs to a class. A crisp output would be a vector where all components are '0' exept of one with '1'. Normally you will get a vector where all components have real values from [0,1]. The question is now how this can be interpreted. Up to now only one interpretation strategy is implemented

- the winner takes all
  The highest value is selected to define the class the propagated pattern is assigned to.

In a next version of NEFCLASS-J additional interpretation strategies will be implemented.

○ select the *Rule Creation* tab



Size of the Rule Base

[✓ max. number of fuzzy rules] 6

automatically determine number of rules

Rule Learning Procedure

best

[✓ best per class]

Rule Base

[✓ re-learn the rule base]

The Rule Base

To specify the size of the rule base is an optimization problem. You get the best classification results when all patterns of the data set are covered by the rule base. The option "automatically determine ..." will do this. This might result in a lot of rules. On the other hand the interpretabillity is higher with a small number of rules. So you can specify a max. number. The option "best" selects the best rules from all. Classes represented by weak rules might not appear in the rule base. In this case it is better to use "best per class".

The option re-learn the rulebase can be used if a rulebase exists that should be used as prior knowledge for a new learning process.

- **Determine the Size of the Rule Base**

The Size of the Rule Base

To specify the size of the rule base is an optimization problem. The minimum number is one rule for every class. So you have to try out the options:

- automatically determine number of rules
  This option encreases the accuracy. The best classification results are received when all patterns of the data set are covered by the rule base but this might produce a lot of rules.
- max. number of rules
  This option increases the interpretability. The interpretability is higher with a small number of rules but not all patterns might be covered by the rule base.

○ select ‘max. number of rules’
○ set the value to ‘6’ rules

- **Change the Rule Learning Procedure**

The Rule Learning Procedure

Here you can decide which rules are selected during the rule learning process.

- best
  selects the best rules of all. Classes represented by weak rules might not be represented in the rule base.
- best per class
  selects under the constraint of the size of the rule base the best rules per class. So every class is represented by the same number of rules. From this follows that the maximum number of rules you have specified has to be rounded up if it cannot be devided through the number of classes.

  ○ use the ‘best per class’ option.

- **Relearn the Rule Base**

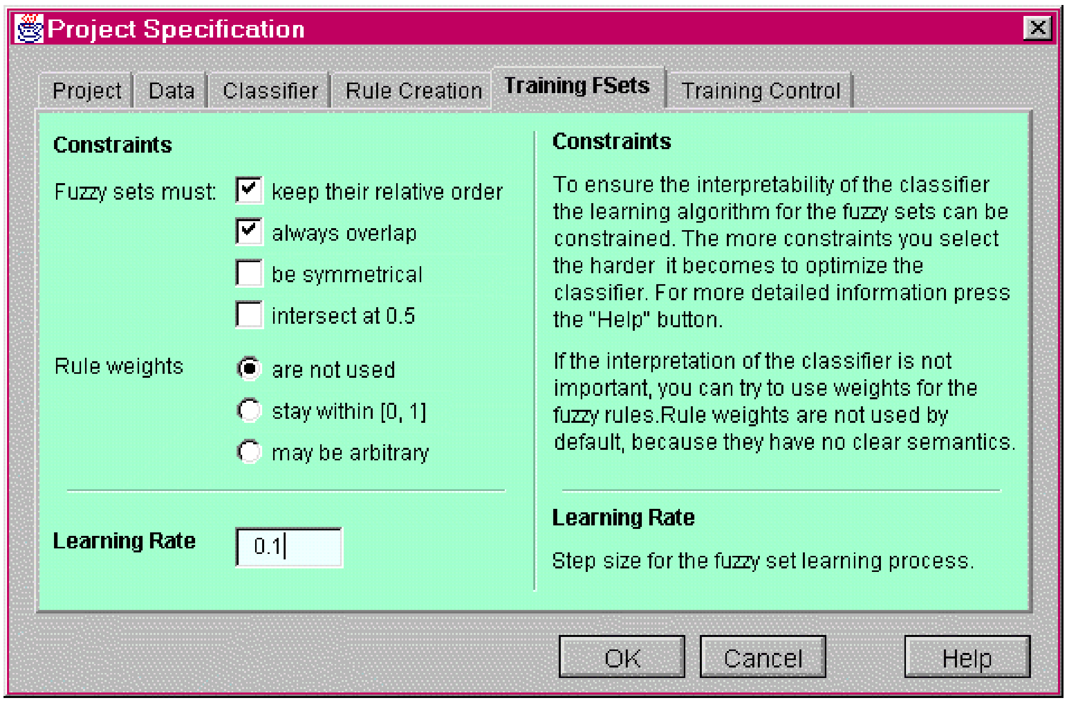
Relearn the Rule Base

This option is needed if a rule base exists, but nevertheless rule learning should be done. The existing rule base will be used as prior knowledge and will be compared against additionally created rules during rule learning. If there is a rule base and this option is not selected, rule learning will be skipped.

This option is needed when parameters are set, although a rulebase is just learned. In this section of the tutorial we have opened an existing project and decided that the results are not good enough to keep the classifier. On the other hand we just edited the labels. This we would have to do again if we created a new project. So we can change the parameter we want to and then relearn the rule base. If we come to the decision that we do not want to overwrite the old classifier we can save the project with a new name.

  ○ select this option

- select the *Training Fuzzy Sets tab*



Constraints

Fuzzy sets must: [✓] keep their relative order

[✓] always overlap

[ ] be symmetrical

[ ] intersect at 0.5

Rule weights: [● are not used] ○ stay within [0, 1] ○ may be arbitrary

Learning Rate 0.1

Constraints

To ensure the interpretability of the classifier the learning algorithm for the fuzzy sets can be constrained. The more constraints you select the harder it becomes to optimize the classifier. For more detailed information press the "Help" button.

If the interpretation of the classifier is not important, you can try to use weights for the fuzzy rules.Rule weights are not used by default, because they have no clear semantics.

Learning Rate

Step size for the fuzzy set learning process.

- **Determine the Constraints for the Fuzzy Sets**

Constraints for the fuzzy sets

These parameters are used to find an suitable balance between high interpretability and a high accuracy. There are four constraints that can be selected.

- keep relative order - is selected by default
  (in NEFCLASS-PC this option was named ‘do not pass neighbours’)
- always overlap - is selected by default
  (was not available inNEFCLASS-PC)
- be symmetrical
  (in NEFCLASS-PC this option was named ‘learn asymetrically’ but the meaning is the opposite)
- intersect at 0.5
  (in NEFCLASS-PC this option was named the same)

More than one option can be used at a time so every combination is possible. It is on you to decide whether it makes sense. The options are described in the following boxes.

Keep Relative Order

It means that during the learning process a fuzzy set must not pass its left or right neighbors. For triangular membership functions for example this is enforced by checking that **all three** parameters of the membership function (left spread, center, right spread) stay smaller (larger) than the parameters of its right (left) neighbor.

To unselect this constraint may lead to rules that cannot be interpreted, but sometimes there are better classification results.

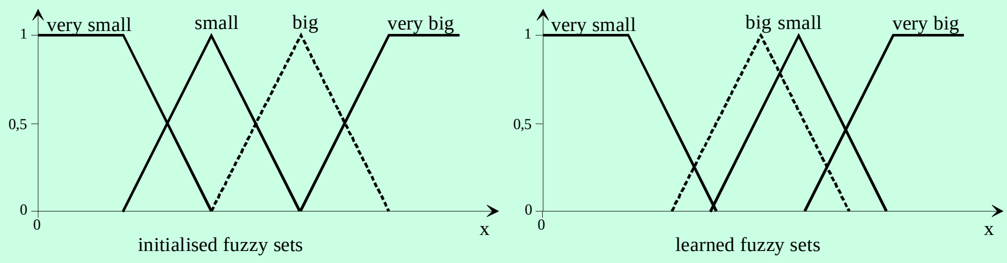


Figure 5.1: Fuzzy sets before (left) and after (right) learning process with unselected option.

In fact the fuzzy set “big” was passed by all three parameters of the triangular membership function “small”. But even passing one point violates this constraint.

Always Overlap

It means that there are no gaps in the fuzzy partition so that the domain is covered by fuzzy sets everywhere.

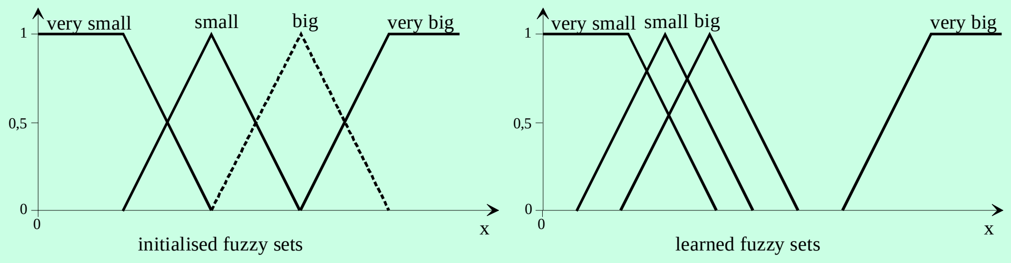


Figure 5.2: Fuzzy sets before (left) and after (right) learning process with unselected option.

There is a gap between the fuzzy sets big and very big.

Be Symmetrical

It means that that both spreads are equally changed during the learning process.

If this option is selected during the learning process only this side of the triangular function is changed, where the input is located, i.e. only either the left spread, or the right spread is changed. It is possible that this constraint cannot be guarnateed, if other constraints must be met.

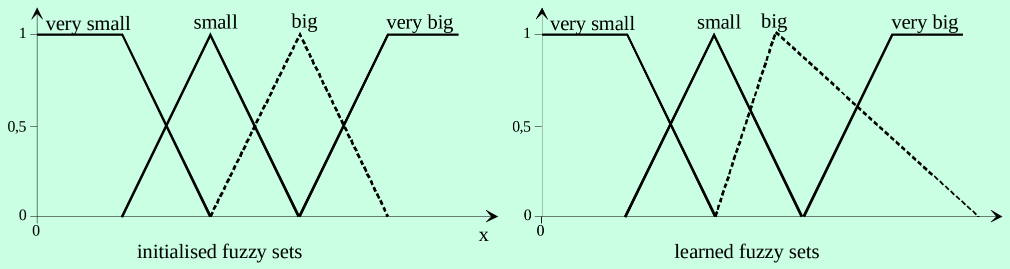


Figure 5.3: Fuzzy sets before (left) and after (right) learning process with unselected option.

The right spread of fuzzy set big is longer than the left.

Intersect at 0.5

It means that two adjacent fuzzy sets must always intersect at a membership value of 0.5, i.e. for each value of the domain the sum of membership values over all fuzzy sets is always equal to one.

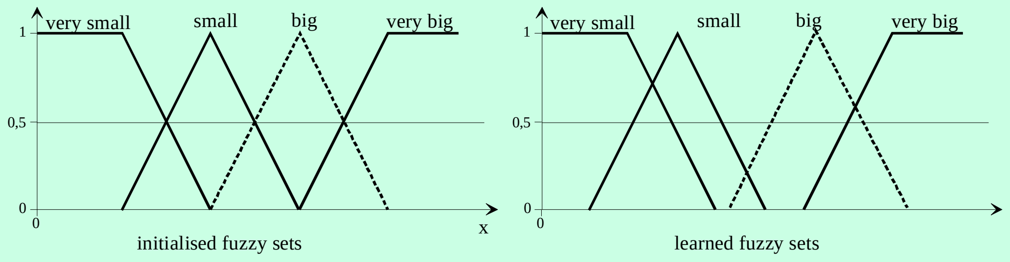


Figure 5.4: Fuzzy sets before (left) and after (right) learning process with unselected option.

Here the fuzzy sets are learned symmetrically without intersecting at 0.5.

- the default selections are ok.

- **Determine the Constraints for the Rule Weights**

Rule Weights

Here can be selected whether rule weights are used for the learning process or not. The selection is:

- not used (default)
  the weights are fixed to 1 for each rule (unweighted rules
- stay within [0, 1]
  rule weights are used but restricted to the interval of [0, 1]
- may be arbitray
  any value for a rule weight is allowed.

It should be stressed here that only a learning process without using rule weights can result in a classifier that can be interpreted.

- **Determine a Learning Rate**

Learning Rate

The learning process is done for minimizing the number of errors of the classifier. This can be visualized by a multidimensional error surface. Imagine you get lost in a mountain area and the task is to find the lowest point in this region. So you will always go downhill and one time reach a valley. But you cannot be sure that you found the valley nearest to the sea level. Because this depends on your starting point, and if you are on a plateau the direction you go first. The same problems have to be solved by the learning algorithm.

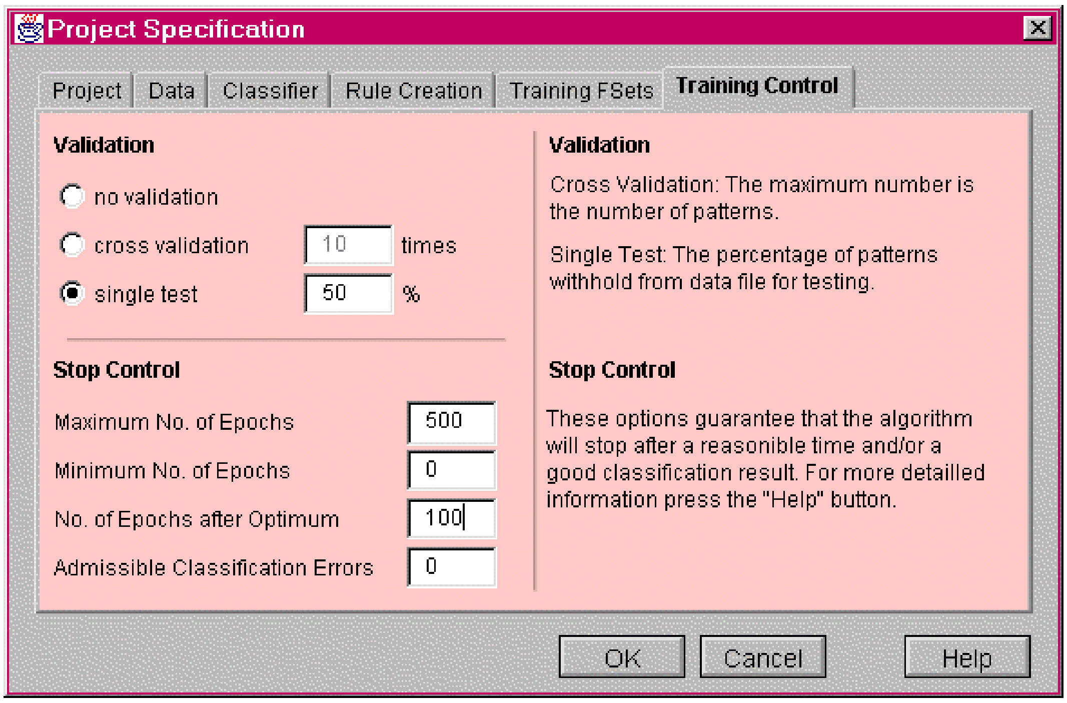
Now imagine that you are a giant going with big steps and one step is as big as the other. You perhaps stamp over a small but deep valley without remarking it. On the other hand if your steps have always the same size and you enter a small deep valley getting smaller and smaller you only can step from one wall to the other but never go deeper because for this you have to do smaller steps.

This width of the steps is the learning rate for the algorithm. Depending on the error surface (that you do not know) you can be lucky with a high learning rate and find a minimum very fast. But if the surface is very rugged you better go with small steps even if it lasts longer.

So if the error increases during the learning process you jumped over a minimum (what also can be an advantage because it *perhaps* was not a good local minimum), or if you find an oszillation of error values try to reduce the learning rate.

- ○ set the learning rate to 0.1 (later you should also try larger values to get a feeling how the learning rate can speed up the process without leading to oszillation. As now batch learnig is used values around 1 are usually acceptable).

○ select the *Training Control* Tab



Validation

[ ] no validation
[ ] cross validation 10 times
[✓ single test] 50 %

Stop Control

Maximum No. of Epochs	500
Minimum No. of Epochs	0
No. of Epochs after Optimum	100
Admissible Classification Errors	0

Validation

Cross Validation: The maximum number is the number of patterns.

Single Test: The percentage of patterns withhold from data file for testing.

Stop Control

These options guarantee that the algorithm will stop after a reasonible time and/or a good classification result. For more detailled information press the "Help" button.

- **Determine the Validation Procedure**

The Validation Procedure

This is the most important new feature of NEFCLASS-J in comparison to NEFCLASS-PC. In the former version only a single test could be done, and for this the user had to devide the data set into two parts by himself. With this you get just an idea about the quality of the classifier but no suitable statements for professional use. Furthermore there was the problem that the learning process should be independant from the data for that the classifier generalizes well. To divide a data file into two parts which both represent the distribution of patterns to classes is not easy. Now you have the possibility to select the ‘single test’ or ‘cross validation’ procedure for the learning process where the division of the data set happens randomly.

There are three options
- no validation
- cross validation
- single test

How to use them depends on the size of the data set and the time you want to put into the learning process. In the following boxes the options are described in detail.

The Training Process

During the learning process the patterns are propagated through the classifier, a total error of these propagations is calculated and with this the weights are changed hoping that the next propagation of the patterns will cause a smaller error. This process will be stopped by parameters you can set and which are described in the next subsection.

No Validation

The whole data set is used for training

- result of the learning process,
  the smallest error reached during all propagations through the classifier.
- use if
  - you do not have much patterns and do not want to divide them to keep some for a test process
  - you do not know much about the data and want to get a feeling for how to set the parameters of the NEFCLASS system to create a good classifier.

Cross Validation

The data set is randomly divided into a number of parts you can set. Let us say we have 150 patterns and chose ‘10’ parts. So you get 10 parts with 15 patterns. The first part is taken to test the classifier and the whole rest of the data set consisting of 135 patterns will be used to train the classifier. You get an error value from the test process.

This is also done with the remaining 9 parts. So in every run of training and testing the whole data set is used but the mixture of the training and test set always changes. In the end you obtain 10 error values which are used to compute a mean error.

- result of the learning process,
  - a mean error
  - a confidence interval calculated on the 99% level

  For our example let us say there is a mean error of 5% and a confidence interval of [-1%, +1%]. Then you can assume with a possibility of 99% that the classification of unseen data will cause an error rate between 4% to 6%.
- use if
  you have found good parameters for the classifier. The number of parts you choose is the number of training processes.

An additonal training process will be carried out. The real classifier is trained with the whole data set. The more data is used the better the classifier should be able to generalize. The patitioning of the data set is only for validation. This validation process may need a lot of time, so the normal way of creating a classifier should be:

- Try out the parameters that can be set with the ‘no validation’ option first. When you think that you are on the right way
- try out the ‘single test’ option and perhaps find parameter setting more precisely.
- If you think that you found a promising parameter setting for the classifier then validate your solution with the ‘cross validaton’ option.

A last point mentioned here is for those who used NEFCLASS-PC. If you want to use your old data sets that you divided yourself you can do the following:

- Load your training data file as depicted
- use ‘no validation’
- train the classifier

You will get the error of the training set. Then

- load the test data file selecting ‘Project|Load Application Data’ from the main menu
- then classify this file selecting ‘Classifier|Classifiy Applicatin Data’.

This computes the classification results and if data with class information is used it also calculates the error.

Single Test

The data set is randomly devided into two parts according to the given percentage value. The smaller part is used for training as depicted above. The bigger part is used to test the classifier.

- result of the learning process,
  - the smallest error caused during all propagations through the classifier.
  - the error on the training data of the same cycle.

- use if
  - the data set is big enough to be devided
  - you do not know much about the data and want to get a feeling for how to set the parameters of the NEFCLASS system to create a good classifier.

- the selection ‘single test’, 50% is ok.

- **Determine the Parameters for the Stop-Learning Control**

The Parameters for the Stop-Learning Control

As depicted in the section about the learning rate you can never be sure to find the absolute minimum. So you need another criterion for stopping the learning process always considering not to stop to early but also to reduce the period of time the learning process needs. Remember that you perhaps want to do a cross validation. Then you need this period several times. The stop criterions are:

- Maximum No. of Epochs (default 100)
  here you can set how often at most the pattern set should be propagated to train the classifier
- Minimum No. of Epochs (default 0)
  here you can set how often at least the pattern set must be propagated to train the classifier
- No. of Epochs after Optimum (default 10)
  with this value you can set how often the pattern set will be propagated after the error value increases again. Refering to the example depicted in the learning rate section: this value is used to ‘climb up the hill’ with the hope to find another valley which goes deeper.
- Admissible Classification Errors (default 0)
  Define the maximum number of admissable misclassifications. If you have - let us say - 100 patterns, and you want to have a 95% correct classification, then you could enter 5 for this option. Unfortunately, the learning procedure is heuristic, and it cannot be guaranteed that the error (or the number of misclassifications) becomes less than any given value.

○ set ‘Maximum No. of Epochs’: 500
○ set ‘No. of Epochs after Optimum’: 100
○ close the dialog with ‘OK’
○ relearn the classifier
For this select ‘Classifier|Create Classifier’ from the main menu.

Please remember that we just have a rule base and that the fuzzy sets are trained. We used the ‘relearn rule base’ option and so the existing rules are used as prior knowledge and the learning process is based on the trained fuzzy sets.
You also have the possibility to reset the fuzzy sets. For this select ‘Classifier|Reset Fuzzy Sets’ from the main menu. To clear the rule base you have to use the rule editor. This will be explained in the next section.

- save the project with a new name

select ‘Project|Save As’ from the main menu and save under a new name. Perhaps ‘Project1b’ is a good idea because ‘Project1’ is used as basis for the learning process

- view the rule base there are only six rules now
- view the fuzzy sets there are five sets per partition now.

5.2.4 Use Prior Knowledge

Prior knowledge in form of fuzzy rules can be inserted to the system before and after the learning process. Rules can also be edited after a learning process. With the option ‘relearn the rule base’ these rules are integrated into to process of selection during the creation of the rulebase. It can happen that these rules will not appear in the resulting rulebase because they were not good enough.

- **The Rule Editor**
  - select ‘Rules|Edit’ from the main menu

A yellow window is invoked which is devided into three rows which are derived from the three parts a rule consists of:

- fuzzy term - e.g. Var 1 is big
- antecedent - many fuzzy terms plus consequent, e.g. class is 2
- many rules

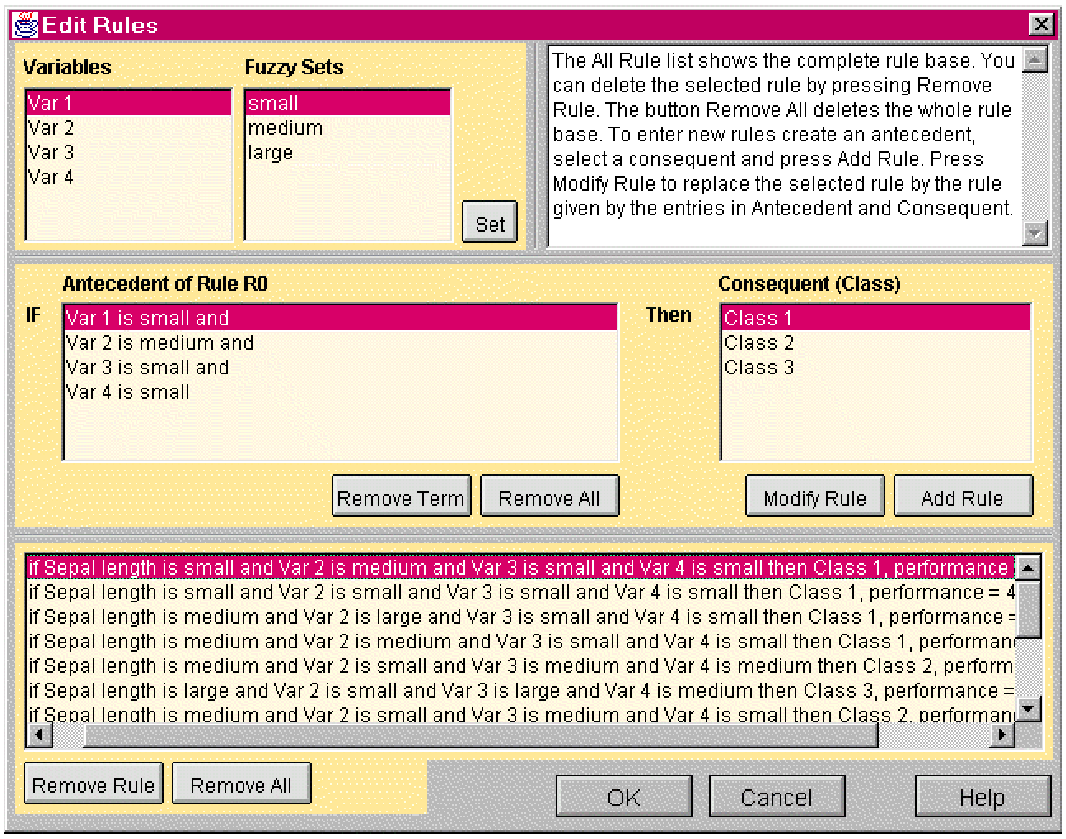
In the right top corner of the window an area with help text is displayed. The informations depend on the field you have clicked in.

Insert a rule

First you have to create an antecedent. So click the ‘Remove All’ button in the *second* row. This will clear the antecedent area. Then build the fuzzy terms by selecting a variable name and a fuzzy set from the lists in the *first* row. With using the ‘Set’ button this term is inserted into the antecedent in the second row. Do so till the antecedent is complete. Then select a class in the list on the right side of the second row. Click the ‘Add Rule’ button to insert the rule into the list of rules in the *third* row.

Modify a rule

Select the rule to be modified in the list of the *third* row. The antecedent of this rule will be displayed in the antecedent field of the *second* row, and the class will be marked. Then select the fuzzy term you want to change. The variable name and the fuzzy set will be marked in the *first* row. Change it and use the ‘Set’ button and so on. If the rule is ready use the ‘Modify Rule’ button in the second row, and the changed rule will replace the old one.



Var 1
Var 2
Var 3
Var 4

small
medium
large

The All Rule list shows the complete rule base. You can delete the selected rule by pressing Remove Rule. The button Remove All deletes the whole rule base. To enter new rules create an antecedent, select a consequent and press Add Rule. Press Modify Rule to replace the selected rule by the rule given by the entries in Antecedent and Consequent.

Var 1 is small and
Var 2 is medium and
Var 3 is small and
Var 4 is small

Class 1
Class 2
Class 3

```
if Sepal length is small and Var 2 is medium and Var 3 is small and Var 4 is small then Class 1, performance
if Sepal length is small and Var 2 is small and Var 3 is small and Var 4 is small then Class 1, performance = 4
if Sepal length is medium and Var 2 is large and Var 3 is small and Var 4 is small then Class 1, performance =
if Sepal length is medium and Var 2 is medium and Var 3 is small and Var 4 is small then Class 1, performan
if Sepal length is medium and Var 2 is small and Var 3 is medium and Var 4 is medium then Class 2, perform
if Sepal length is large and Var 2 is small and Var 3 is large and Var 4 is medium then Class 3, performance =
if Sepal length is medium and Var 2 is small and Var 3 is medium and Var 4 is small then Class 2, performan
```

- ○ if you had not closed it Project1b is still open if not please open it now
- ○ relearn the classifier.
- • select ‘Classifier|Reset Fuzzy Sets’
- • select ‘Rules|Edit’
- • select the ‘Remove All’ button in the *third* row to clear the rule base
- • close the rule editor with ‘OK’
- • select ‘Classifier|Create Classifier’ from the main menu.
- ○ repeat this again and compare the results

The results may vary a lot. This is because you have chosen the ‘single test’ option. For every training process the two parts of the data set are selected again. Here you can see that the development of a classifier depends on the used data and that it is necessary to reduce this influence to a minimum.

- ○ save and close the project.

5.2.5 Pruning the Rule Base

Pruning the rule base is used to improve the interpretability of the classifier. Pruning tries to get the same or a better classification results with a smaller rule base. There are three options to prune the rule base:

- **Manual Pruning**

  For this open the rule editor. Every line for a rule has a number and the performance of the rule in the beginning. Use the values for the performance to find out which are the good rules. Then you have two possibilities:

  - ○ use the 'good rules as prior knowledge'
    - erase the rules with the smallest performance values
    - set the 'number of rules' parameter the same as rules are specified now,
    - set the 'relearn the rulebase' option and
    - create the classifier again.
      Think about resetting the fuzzy sets before (this need not to be done but you can try it).
  - ○ create a completely new rule base
    - count the 'good rules',
    - remove all rules,
    - set the 'number of rules' parameter to the value you counted,
    - reset the fuzzy sets,
    - create the classifier.

  If this process was not successful you can get the old rulebase by using 'Rules|Restore Rule Base'. But remember that only the last rule base can be restored. If you do this manual pruning process several times you have to save the former classifier under a new project name. Please remember further that you might have changed the fuzzy sets. The item 'Classifier|Restore Fuzzy Sets' of the main menu will restore the state before the last step.

  With this you only can erase rules. Methods which also reduces the number of variables in a rule are used in the pruning option described next

- **Automatic Pruning**

  Please note that the old rule base can be restored after the pruning process. But just in case it may be better to save the classifier under a new project name beforestarting to prune.

  There are four methods that are implemented for automatical pruning. These methods are automatically used one after the other. After a method is used the fuzzy sets are trained. For this the selected validation mode is used. If the resulting classifier produces a smaller number of misclassifications or a smaller error value than the former one, this new one is taken to be improved with the next pruning method. During this process all parameters you specified, except these concerning the rulebase, are used.

The orange and green windows that you just know from creating a rule base and training the fuzzy sets are invoked. In the green window for the fuzzy set training process the error and misclassifications are plotted as known. In the orange window the text area is used to describe the steps of the pruning process.

- **Semi-automatic Pruning**

  This is not implemented yet. It is planed to offer the four pruning methods to the user. This way the sequence of using the method is free to select.

5.3 Strategies to Create a Classifier

From the last sections the following can be extracted as strategies to create a classifier.

- Use the ‘no validation’ option in the beginning and try out the parameter settings. For example use very small and very large values. So you get a feeling for the range of values which will work.
- For the beginning it can be a good idea only to learn the rule base till you found out how many rules might be ok and then learn the fuzzy sets. Do not forgett to reset the fuzzy sets and clear the rule base if you start to create a completely new classifier.
- To find out a good number of rules you also can try the option ‘automatically determine the number of rules’. This will find a possibly big rule base but it will cover all patterns. With using the automatic pruning you can find out how small the rule base can be. Then use this number as a basis for a new creation process and so on.
- If you think you found a good parameter setting use the ‘cross validation’ option to get a statement about the quality of the classifier.

5.4 The Main Menu

This is a very short overview of the main menu entries. Most of them were just used in the tutorial.

- **Menu Project**
  - New Project
    Invokes the ‘Project Specification Dialog’ with default values.
  - Open Project
    Invokes the ‘Project Specification Dialog’. All files saved with the project are opened automatically.
  - Edit Project
    Invokes the ‘Project Specification Dialog’.
  - Save
    Saves the project under the given name.

- Save As
  Saves the project by invoking a save dialog.
- Close Project
  Closes a project and all files concerning it. Furthermore all windows concerning the project are closed.
- Load Training Data
  Loads a data file with patterns to use for training. Training data files can also be loaded by the *Data* tab of ‘Project Specification’ dialog.
- Load Application Data
  Loads a data file with patterns to use as an application. Please do not mix it up with the training data. Application data can only be loaded via main menu.
- Exit NEFCLASS

- **Menu Classifier**

  - Create Classifier
    Starts the rule learning and the fuzzy set training process. It invokes an orange window which displays the rule learning progress and it invokes a green window which displays the fuzzy set training progress.
  - Create Rule Base Only
    Starts the rule learning process. It invokes an orange window which displays the rule learning progress. If you use it with a cross validation option and the rule bases differ a lot you can take this as an indication that the classes of the data set cannot be separated very easily. So that a good rule base found here might not generalize good enough. This you will find out with a cross validation for the fuzzy set training.
  - Train Fuzzy Sets Only
    Starts the fuzzy set training process. It invokes a green window which displays the fuzzy set training progress.
  - Prune Classifier
    Starts the automatic pruning process using the specified parameters. Invokes the fuzzy set learning window to show the behaviour of the error, and the rule learning window to display informations of the pruning process. Although rules are not learned here this window is used because this is a process concerning the rule base. If the ‘cross validation’ option is selected the a validation of the parameter learning is performed.
  - Create Classifier and Prune It
    This item is a service for the user. It concatenates the ‘Create Classifier’ and the ‘Prune Classifier’ feature. So if you use the ‘cross validation’ option the classifier is created with this option and the resulting classifier is pruned with this option too. Figure 5.5 shows this process.
  - Create Pruned Classifier
    If this item is used with the ‘no validation’ or the ‘single test’ option it is the same as ‘Create Classifier and Prune it’ using ‘no validation’or the ‘single test’. With selecting ‘cross validation’ everything changes. Creation and pruning are considered to be a unified whole which is validated. Figure 5.6 shows this process.

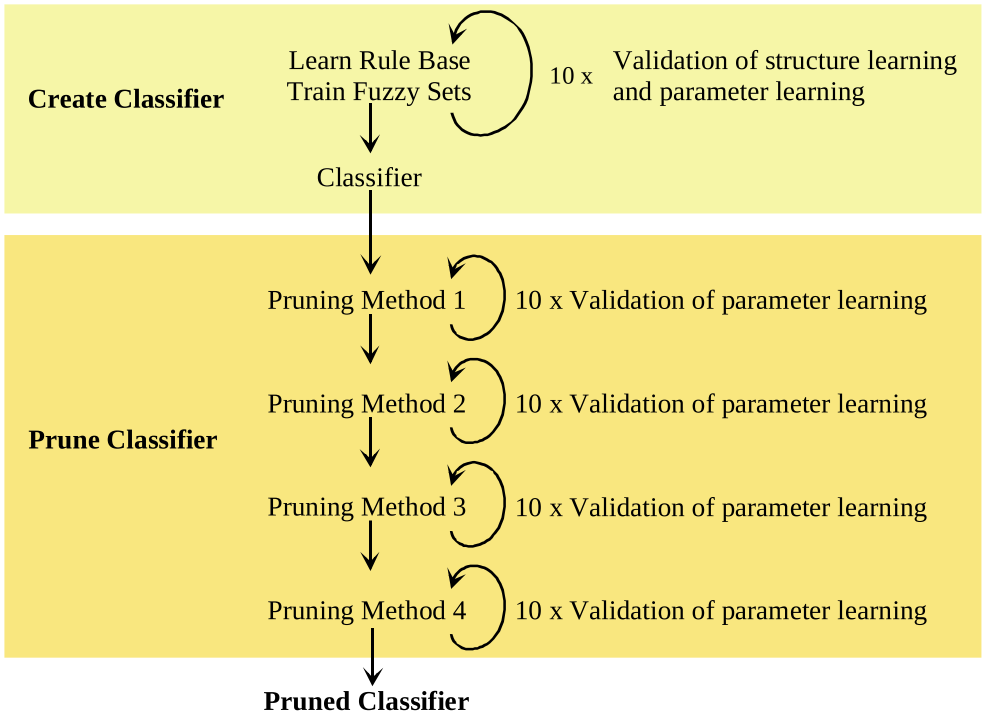

Create Classifier
Learn Rule Base
Train Fuzzy Sets
10 x
Validation of structure learning and parameter learning
Classifier
Prune Classifier
Pruning Method 1
10 x Validation of parameter learning
Pruning Method 2
10 x Validation of parameter learning
Pruning Method 3
10 x Validation of parameter learning
Pruning Method 4
10 x Validation of parameter learning
Pruned Classifier

Figure 5.5.: Process of the feature 'Create a Classifier and Prune it' using a 10-fold cross validation.

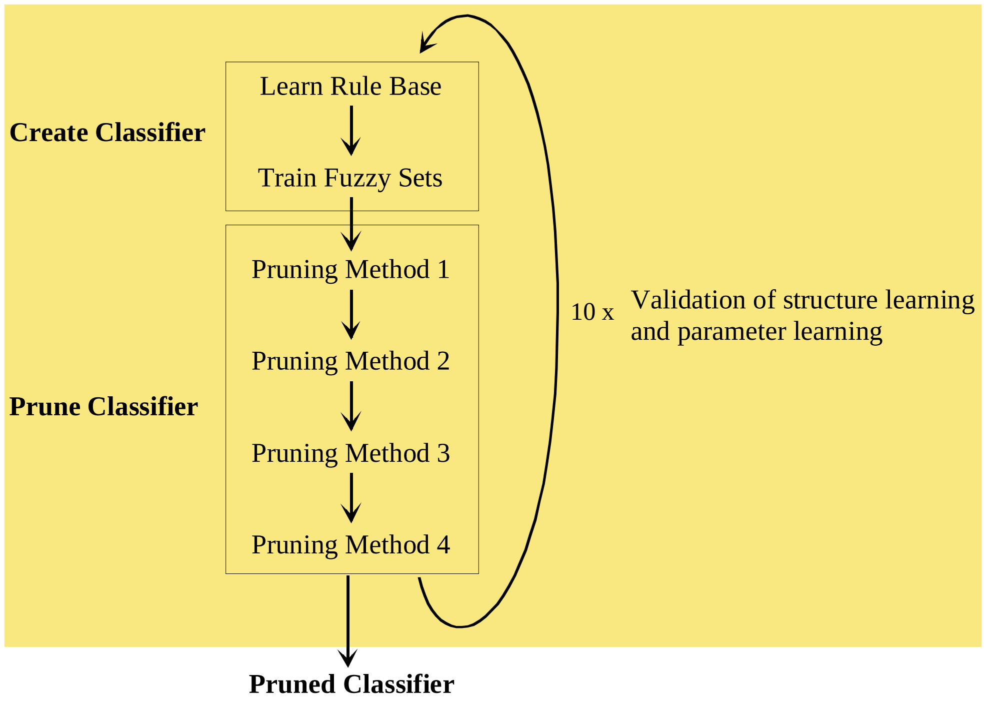
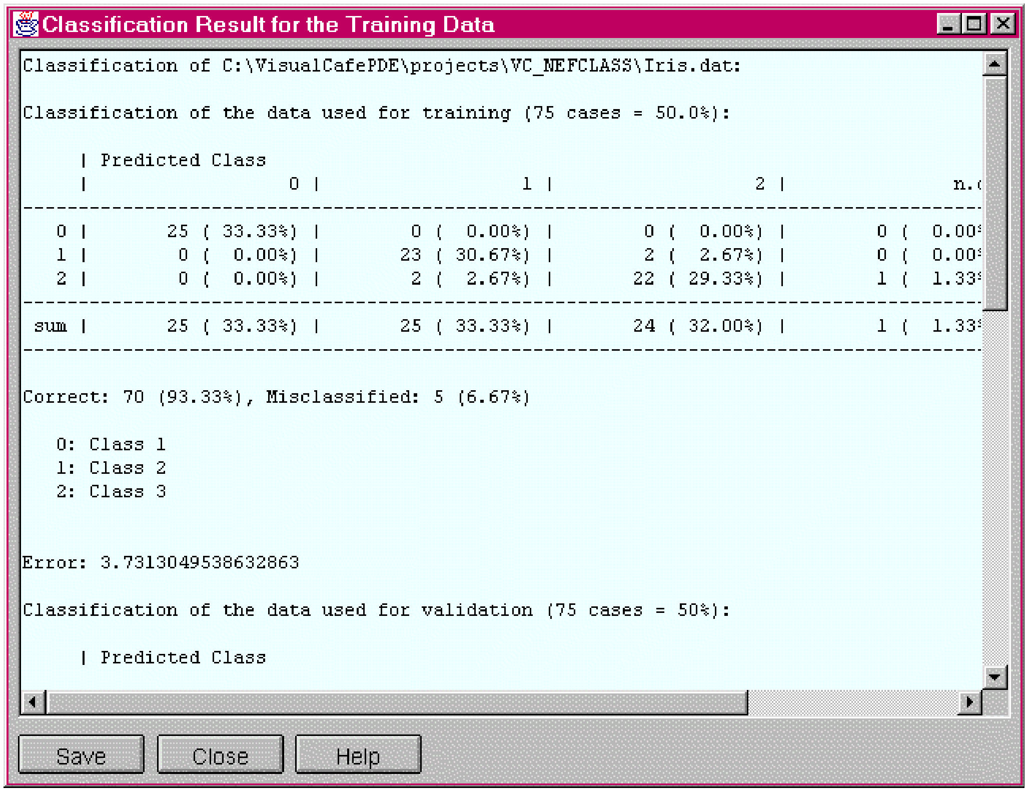


Figure 5.6.: Process of the feature 'Create a Pruned Classifier' using a 10 fold cross validation.

- Restore the Fuzzy Sets - Reset the Fuzzy Sets
  Reset means to create new initial fuzzy partitions according to the parameters given in the ‘Project Specification’ dialog. Restore means that the last step that changed the fuzzy sets can be undone.
- Classify Training Data - Classify Application Data
  The training data or the application data can be classified using ’Classifier|Classify Training Data’ and ’Classifier|Classify Training Data’ respectively.

A blue window will be invoked and display a matrix where the columns are the predicted class and the rows are the actual class. The classification results per class are displayed as absolute values and as percentage values. An additional column is for not classified patterns. The sums over all columns and rows are given as well as the entire classification result.



```
Classification of C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat:

Classification of the data used for training (75 cases = 50.0%):

     | Predicted Class
     |                 0 |                 1 |                 2 |                n.
----------------------------------------------------------------------------------
   0 |        25 ( 33.33%) |         0 (  0.00%) |         0 (  0.00%) |         0 (  0.00
   1 |         0 (  0.00%) |        23 ( 30.67%) |         2 (  2.67%) |         0 (  0.00
   2 |         0 (  0.00%) |         2 (  2.67%) |        22 ( 29.33%) |         1 (  1.33
----------------------------------------------------------------------------------
 sum |        25 ( 33.33%) |        25 ( 33.33%) |        24 ( 32.00%) |         1 (  1.33
----------------------------------------------------------------------------------

Correct: 70 (93.33%), Misclassified: 5 (6.67%)

   0: Class 1
   1: Class 2
   2: Class 3


Error: 3.7313049538632863

Classification of the data used for validation (75 cases = 50%):

     | Predicted Class
```

- Stop Training Process
  Stops any learning process as soon as possible.

- **Menu View**
  - Error
    Invokes the window which shows the progress of the fuzzy set training process. This window can be viewed to either time after the training process has ended. In this window a graph which displays the errors and the misclassifications during the training process is shown. The error graphs can be printed (one graph per page).
  - Fuzzy Sets
    Invokes a window where the fuzzy partitions of all variables are displayed. It is possible to select only some variables to be shown. The fuzzy partitions can also be printed (three partitions per page).
  - Statistics of the Training Data Set - Statistics of the Application Data Set
    In the first part of the tutorial we just mentioned that statistics of the training data can be viewed using the ‘Statistics’ button of the *Data* tab. You can also open the statistics window by the ‘View|Statistics Training Data’ or ‘View|Statistics Application Data’ entries.
  - Hide All - Show All
    All windows (except the main window) are closed or opened.

- **Menu Rules**
  - Edit
    Invokes a window which displays the rules. The rules can be edited or deleted, the rule base can be cleared or new rules can be inserted.
  - Restore the Rule Base
    This means that the last step that changed the rule base can be undone. This last step can be a learning process, a pruning process or the use of the rule editor.
  - Export Rules
    The rule base is be written to an ASCII text file.

- **Menu Help**
  - Help Topics
    Invokes an HTML browser to show a HTML file. There the help topics are displayed in alphabetical order. If no browser can be found, a message with further instructions appears.
  - About
    Displays information about the NEFCLASS-Tool

5.5 The Files of a NEFCLASS Project

There are some ASCII files NEFCLASS-J uses or creates. It might be necessary to edit or view them so that they are described in the following.

5.5.1 The Data File

The data file is the only file you must edit. NEFCLASS-J needs some informations to initialize the classifier. These initial values with the corresponding keywords must be written on the top of the pattern file. A NEFCLASS data file consists of two to five blocks. Comments are allowed between the blocks as well as line feeds or any kind of whitespace. Every line of comment has to begin with %, # or //.

The blocks have the following structure:

- The first line of a block is the *keyword*, i.e. this word is the only word in this line. Keyword are written in capital letters.
- The following lines contain the corresponding *data*.
- Some blocks need a *keyword* for the end of data

There is a mandatory first block and a mandatory last block. The other blocks are optional and because of the use of kewords there is no need to stick to a sequence. The blocks are described now in detail.

Mandatory blocks:

- ♦ DIMENSIONS
  DIMENSIONS has to be *the first block* in the file. The next line consists of two integer values. The first gives the number of independent variables that means input values, the second gives the number of dependant variables that means the number of classes. If no class information is given this value must be set to ‘0’.

- ♦ PATTERNS
  PATTERNS has to be *the last block* in the file.The line following the keyword consist of one integer value. This is the number of patterns. In the following lines the patterns are listed one pattern per line.
  - the values can be separated by any whitespace.
  - the values for the input variables may be real or integer
  - the classes must be coded as a real vector from $[0, 1]^m$ (m is the number of classes). The class of the pattern is determined by the largest value. Usually each pattern belongs only to one class. Therefore all components except one are set to 0 and the one indicating the class is set to 1. For example, if there are 4 classes then class 2 will be encoded as 0 1 0 0.
  - this block can be finished with the keyword END. This is meant for data files used by NEFCLASS-PC. So files containing this keyword need not to be changed.

Optional blocks:

- ♦ Block for a data file description
  - NAME
    With using NAME only one line of description can follow in the next line. This command is used in data files used by NEFCLASS-PC. These files need not to be changed.

  or
  - STARTNAME / ENDNAME
    The first line of this block is the keyword STARTNAME. The description follows in the next lines. The block has to be finished with the keyword ENDNAME.
- ♦ VARNAMES
  In the lines following this keyword the variable names can be set. It might be easier to use NEFCLASS-J for this. When a data file is loaded default names are given to the input variables and the classes. If you open the ‘Project Specification’ dialog and select the *Data* tab you will find a button ‘Edit Labels’. Using this an editor is invoked where you can change the names. It has the advantage that you can change only some names. For the others the default names will be saved. If you write the names directly to the file, you have to name all variables and classes.
- ♦ INRANGES
  Here the ranges for each input variable can be specified. The exact ranges are computed during the patterns are loaded but here you can specify larger ranges for the patterns.

In the blue box the top of the Iris data file is given as an example where only some keywords are used.

```
% This is the IRIS data set reformatted for use in NEFCLASS-PC 2.0
% Class 1 0 0 is Iris-setosa, class 0 1 0 is Iris-versicolor, and
% class 0 0 1 is Iris-virginica.
% This file contains the whole data set with 150 cases, 50 for each class
DIMENSIONS
4 3
% Name of the pattern set
NAME
IRIS.DAT: Iris data, sorted, whole set
% Ranges of the 4 input variables
INRANGES
4 8
2 5
1 7
0 3
% This are the 150 patterns,
% first given are the number of patterns, inputs and outputs
PATTERNS
150
5.1  3.5  1.4  0.2  1 0 0
```

5.5.2 The Parameter File

This file has the extension '.prj' and is an ASCII file that can be viewed but should not be edited. It consists of the parameters set in the 'Project Specification' dialog, and also the status of the buttons. The lines only contain the values without any description so you will not be able to find out which line contains which value. You can change these values by the dialog so that there is no need to edit this file

```
PARAMETERLIST
C:\VisualCafePDE\projects\VC_NEFCLASS\Project 1.prj
Project 1
Project using default values.
*
C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat
```

The first lines of this file show a keyword followed by the directory and name to which the project is saved , the project description and the data file used.

5.5.3 The Classifier File

This file has the extension '.cls' and is an ASCII file that can be viewed but should not be edited. This file contains all the values created from NEFCLASS to specify the classifier. On the top of the file the directory, the filename and the date of creation is shown in a comment. The various values and parameters that define the classifier are written in blocks that are marked with keywords on the top. In the yellow box you can see a few lines from the top of the file.

```
% NEFCLASS file created by NEFCLASS-J 1.0
% (c) U. Nauck, Braunschweig, 1999
% Filename: C:\VisualCafePDE\projects\VC_NEFCLASS\Project 1.cls
% This file was created on 11/27/1998 at 13:51

% This are the structure parameters
PARAMETERS
```

5.5.4 The Training Protocol File

This file is named 'nefclass.log' and it is an ASCII file. The file is a documentation of the work with a certain project. On the top of the file the used parameters are listed. It is followed by the documentation of the feature you selected from the main menu. A very complex feature for example is 'Create a Pruned Calssifier' with cross validation. The rule creation process with the performances of the rules, the fuzzy set learning process with the error values and the process of the four pruning methods with the particular fuzzy set learning is documented. If you are doing a 10 times cross validation you will find this documentation 10 times. If you close a project this file is closed and will be overwritten when another project is opened or a new one is created. If you want to keep it it must be copied to a file with a different name. The blue box shows the top of a log file listing the feature mentioned above.

```
% NEFCLASS log file file created by NEFCLASS-J 1.0 (c) Ulrike Nauck,
Braunschweig, 1999
% This file was created at January 1, 1999 4:55:30 PM GMT+00:00

Training data: C:\VisualCafePDE\projects\VC_NEFCLASS\Iris.dat
Training for at most 100 cycles and for at least 0 cycles.
Continue for 30 cycles after a local optimum was found.
Learning will stop at 0 misclassifications.
Validation mode is 10-fold cross validation.
Parameter: LearningRate = 2.0, Fuzzy set constraints:
keep order, must overlap,
rule weights are not used.
The rule base will consist of 3 rules using the best rule for each class.
Each variable uses 3 TRIANGULAR fuzzy sets.
Fuzzy sets will be trained.
Each created classifier will be pruned.
```

5.5.5 The Result File

For every pattern the classification result is given.

```
Classification of data file C:\VisualCafePDE\projects\VC_NEFCLASS\wbc.dat

pattern 1 is class malign, predicted class is malign,    error = 0.004 => correct
    target:   1.000 0.000
    output:   0.939 0.000

pattern 2 is class malign, the pattern was not classified,  error =  1.000 => false
    target:   1.000 0.000
    output:   0.000 0.000
```

6

The Structure of NEFCLASS-J

The structure of NEFCLASS-J is determined by the graphical user interface (GUI) and three Java packages `nauck.fuzzy`, `nauck.data` and `nauck.util`. The packages contain classes that implement the NEFLCASS model and learning algorithms, the data structure for training data and some utility classes that are used to draw graphs or format strings. In the following we use the `Courier` font to denote class, package and file names.

6.1 The Graphical User Interface

The main object of NEFCLASS-J is found in `NEFCLASS.java`. The object is an extension of the Java Frame object. It contains the menu and action handlers for all menu entries. NEFCLASS-J uses several modal dialogs (they must be closed before the program resumes its work) and non-modal frames (they can remain open on the desktop) to obtain user input and to provide information about the current status.The following list gives a short description of all classes that are used to construct the GUI.

- `AboutDialog`
  Information about the program version.

- `DialogClose`
  A dialog to ask the user if he wants to save the current project before closing it.

- `DialogEditLabels`
  Dialog to edit the names of the variables used in the current project. This dialog is invoked via the project specification dialog `DialogProjectSpecification`.

- `DialogYesNo`
  A dialog to ask the user for a yes or no decision.

- `DialogMessage`
  A dialog to display error messages and information about the current state.

- `DialogProjectSpecification`
  The project specification dialog is the most important dialog of NEFCLASS-J. Here all parameters for the current project and training process are specified.

- `DialogRuleEdit`
  This dialog implements the rule editor that allows to enter or modify fuzzy rules.

- `DialogSpecifyFP`
  This dialog is invoked by the project specification dialog. It is used to specify individual fuzzy partitions for each variable.

- `DialogProgress`
  A simple progress dialog to illustrate the process of reading a data file.

- `FrameShowFS`
  This frame displays the fuzzy sets of the variables.

- `FrameStatistics`
  This frame is used to display statistics of training and application data as well as classifications results.

- `FuzzySetLearningProgressDialog`
  This is actually a frame that displays the progress of the fuzzy set learning algorithm. It contains a progress bar, a text area for messages and draws two graphs: one to illustrate the error, and one to illustrate the number of misclassifications. The frame uses classes from the `nauck.util` package to draw the graphs.

- `NEFCLASS`
  The main class of the program. It contains the main frame of the application and displays the menu and status bar.

- `ParameterList`
  This is a non-graphical class that contains all informations about the current project. The class is able to write itself to an ASCII file and to read itself from such a project file. Project files use the extension `prj`.

- `QuitDialog`
  This dialog is used to ask the user if he really wants to exit the program.

- `Resources`
  This is a non-graphical class that contains strings used for error and information messages. If the application shall be translated into another language it is sufficient to translate this class.

- `RuleLearningProgressDialog`
  This is actually a frame that displays the progress of the rule learning algorithm. It contains three progress bars for the different stages of rule learning and a text area for messages. This dialog is also used to display general training and pruning information.

Figure 6.1 displays the class hierarchy of the GUI. The classes described above can be found in the upper right corner of the figure. The other leave nodes of the depicted tree are classes that are needed to process mouse or window events.

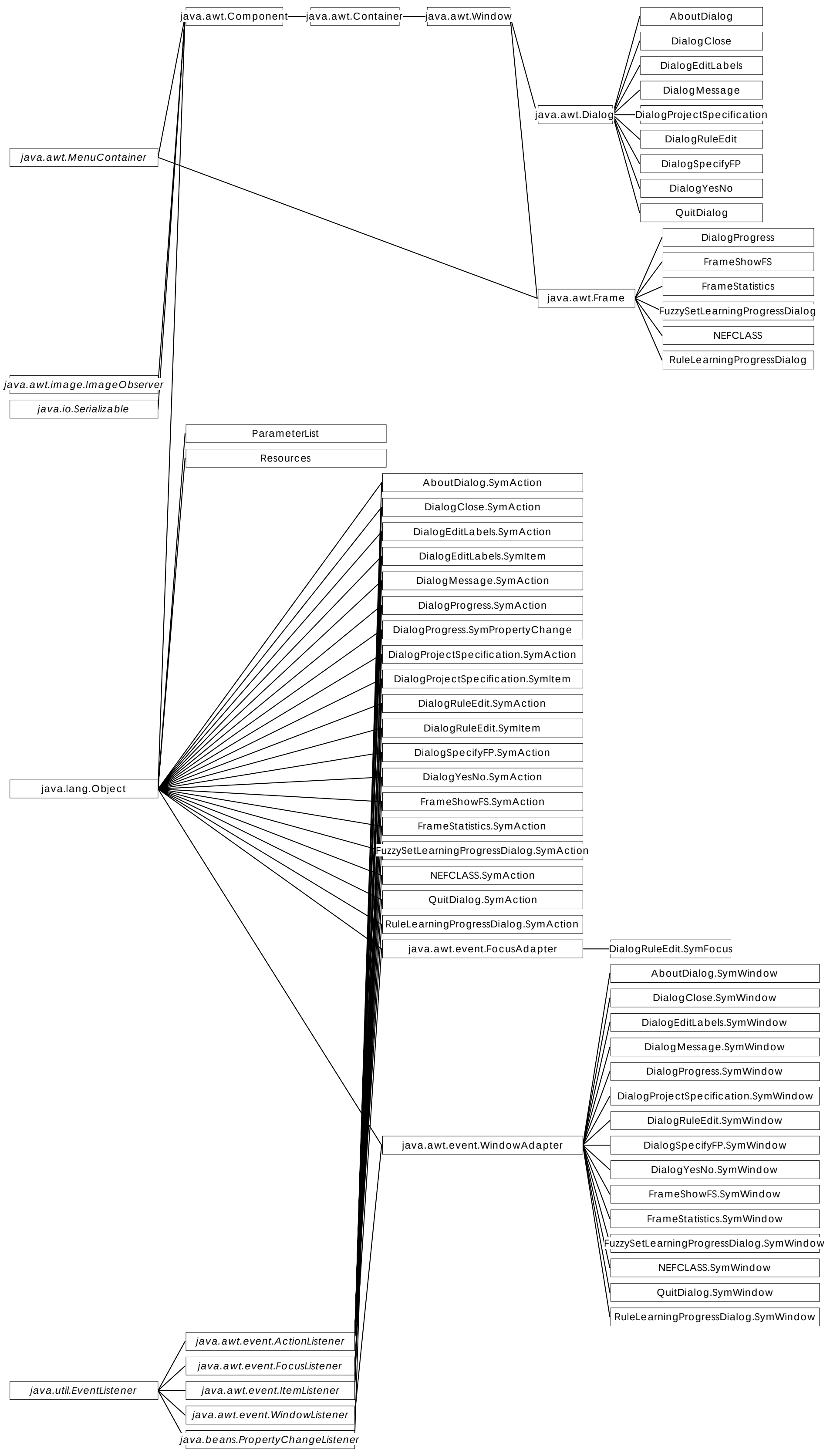


Figure 6.1: The class hierarchy of the GUI of NEFCLASS-J

6.2 The Data Structures of the NEFCLASS Model

The NEFCLASS model is implemented by the classes of the `nauck.fuzzy` package. The class hierarchy of this package is given in Figure 6.2.

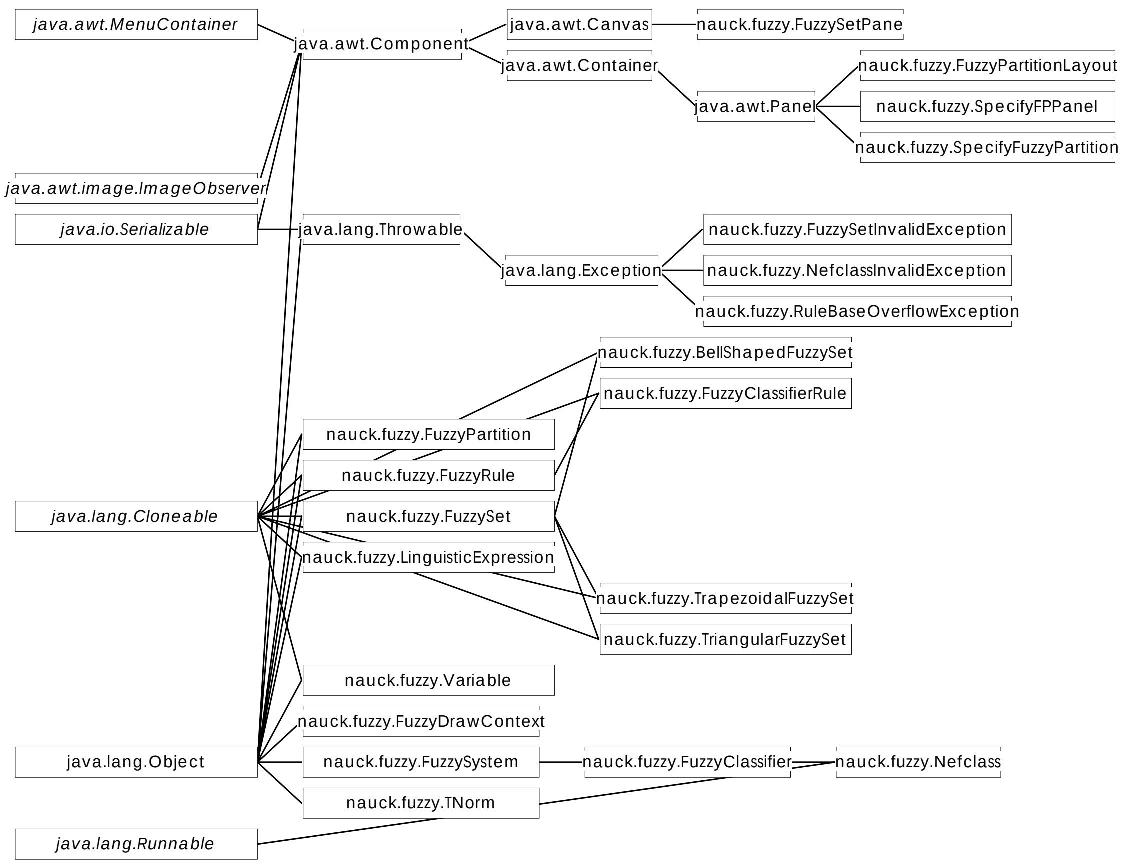

java.awt.MenuContainer
java.awt.Component
java.awt.Canvas
nauck.fuzzy.FuzzySetPane
java.awt.Container
nauck.fuzzy.FuzzyPartitionLayout
java.awt.Panel
nauck.fuzzy.SpecifyFPPanel
nauck.fuzzy.SpecifyFuzzyPartition
java.awt.image.ImageObserver
java.io.Serializable
java.lang.Throwable
nauck.fuzzy.FuzzySetInvalidException
java.lang.Exception
nauck.fuzzy.NefclassInvalidException
nauck.fuzzy.RuleBaseOverflowException
nauck.fuzzy.BellShapedFuzzySet
nauck.fuzzy.FuzzyClassifierRule
nauck.fuzzy.FuzzyPartition
nauck.fuzzy.FuzzyRule
java.lang.Cloneable
nauck.fuzzy.FuzzySet
nauck.fuzzy.LinguisticExpression
nauck.fuzzy.TrapezoidalFuzzySet
nauck.fuzzy.TriangularFuzzySet
nauck.fuzzy.Variable
nauck.fuzzy.FuzzyDrawContext
java.lang.Object
nauck.fuzzy.FuzzySystem
nauck.fuzzy.FuzzyClassifier
nauck.fuzzy.Nefclass
nauck.fuzzy.TNorm
java.lang.Runnable

Figure 6.2: The class hierarchy of the nauck.fuzzy package

The learning algorithms of the NEFCLASS model are implemented in the class `Nefclass`. It extends the class `FuzzyClassifier` which is derived from the abstract class `FuzzySystem`. This class defines the main features of a generic fuzzy system and must be subclassed to implement any kind of fuzzy system, e.g. a fuzzy classifier. `FuzzySystem` provides a rule base which is a dynamic array (`java.util.Vector`) of objects of type `FuzzyRule` and methods to propagate a pattern vector through the rule base. Subclasses of `FuzzySystem` must add functionality such as to compute outputs.

The class `FuzzyClassifier` adds this kind of functionality by providing a method to compute a classificaton result from a rule base of fuzzy classification rules. The class `Nefclass` further extends this class by providing learning methods for creating fuzzy rules, train membership functions, pruning and several methods to compute and to present classification results. `Nefclass` implements the interface `java.lang.Runnable` to be able to run the learning and pruning algorithms in a separate thread. Thus the user is able to use the GUI while the classifier is trained. The learning process can be stopped at any time via a menu entry.

`Nefclass` uses a rule base consisting of objects of the class `FuzzyClassificationRule`. This class extends the abstract class `FuzzyRule` that provides a generic framework for fuzzy rules. A generic fuzzy rule contains an array of `LinguisticExpression` objects which implement the linguistic terms of the antecedent of the rule. A `LinguisticExpression` contains a `Variable` and a `FuzzySet`. A linguistic term is therefore given by a pair (variable, fuzzy set). A fuzzy rule uses a `TNorm` object to evaluate its antecedent, when it receives an input vector to compute a degree of fulfilment. A `FuzzyRule` object cannot compute an output value. This functionality must be provided by subclasses. The subclass `FuzzyClassificationRule` implements a special kind of fuzzy rule that uses a class label as consequent. Its output value is a degree of membership for an input vector to the class specified by the consequent. This value is simply the degree of fulfilment of the antecedent.

A `Variable` describes a variable from a data file that is organized as a single relation, i.e. as a table with one column for each variable and one row for each pattern or case. A `Variable` contains the index (column number) of the variable, its name and some statistical information like lower and upper bound.

The abstract class `FuzzySet` implements a generic fuzzy set that defines several abstract methods for computing degrees of membership and computing parameter updates which must be implemented in subclasses. The package provides three types of fuzzy sets by the classes `BellShapedFuzzySet`, `TrapezoidalFuzzySet` and `TriangularFuzzySet`. The fuzzy sets are given by parameters and their membership functions are defined as follows (the parameter names match the class implementations):

$$\text{bell-shaped:}\quad \mu(x) = e^{-\left(\frac{overlap\,\cdot\,(x-center)}{width}\right)^2}$$

$$\text{trapezoidal:}\quad \mu(x) = \begin{cases} 0 & \text{if } x < left \vee x > right \\ 1 & \text{if } x > centerLeft \wedge x < centerRight \\ \dfrac{centerLeft-x}{centerLeft-left} & \text{if } x \geq left \wedge x \leq centerLeft \\ \dfrac{x-centerRight}{right-centerRight} & \text{if } x \geq centerRight \wedge x \leq right \end{cases}$$

triangular: $$\mu(x) = \begin{cases} 0 & \text{if } x < left \lor x > right \\ \dfrac{center - x}{center - left} & \text{if } x \geq left \land x \leq center \\ \dfrac{x - right}{right - center} & \text{if } x > center \land x \leq right \end{cases}$$

A “bell-shaped” membership function is alway symmetrical. By default *overlap* = 1.7 is used. Thus the parameter *width* specifies the half width of the α-cut at $\alpha = e^{-2.89} \approx 0.05$. All three types of fuzzy sets can be “shouldered”. This means, the degree of membership is one for all values small than the *center* (*centerRight*) parameter for the lefmost fuzzy set of a partiton (“left-shouldered”) or for values greater than *center* (*centerLeft*) for the rightmost fuzzy set of a partition (“rightshouldered”).

In addition to methods for computing the degree of membership each of the three classes provides methods for updating its parameters under certain constraints. Parameter updates are only carried out, if they result in a legal form of the membership function. Additional constraints can be used to make sure, that a fuzzy set does not pass another fuzzy set, that a fuzzy set overlaps with another fuzzy set, that the membership function stays symmetrical or that the degrees of membership add up to one for neighboring fuzzy sets. Fuzzy sets can also draw graphs of their membership functions.

The class `FuzzyPartition` is used by `Nefclass.` As described in Chapter 2 the NEFCLASS model makes sure that each linguistic value is only represented once. Thus it is necessary that all fuzzy rules use the same fuzzy sets in the linguistic terms of their antecedents. `Nefclass` ensures this by providing a fuzzy partition for each variable. The fuzzy rules use references to the fuzzy sets of those fuzzy partitions. A fuzzy partition provides methods to inovke the methods of its fuzzy sets that compute the parameter updates and it can draw the graphs of the complete partition. There are also methods for setting up intial partitions of equally distributed membership functions. A fuzzy partition can have an arbitrary number of fuzzy sets, but they must be of the same type. If there is only one fuzzy set, the respective variable becomes a “don't care” variable, as for each value the membership degree is one. `Nefclass` can use for each variable a fuzzy partition of different numbers and types of fuzzy sets.

The remaining classes of the package provide some special functions and exceptions that are used to denote errors while using the classes at run time. The classes `FuzzySet-Pane`, `FuzzyDrawContext` and `FuzzyPartitonLayout` implement the graphical interface for fuzzy sets and fuzzy partitions. A `FuzzySetPane` extends `java.awt.Canvas` and provides a canvas for a fuzzy partition to draw its fuzzy sets on. A `FuzzyPartitonLayout` is a container that can hold several `FuzzySetPane` objects to be displayed on screen. Fuzzy sets are displayed by adding a `Fuzzy-PartitonLayout` to a `java.awt.ScrollPanel` of the `DialogShowFS` class of the GUI. The classes `SpecifyFPPanel` and `SpecifyFuzzyPartition` are used in `DialogSpecifyFP` of the GUI to specify individual fuzzy partitions for all variables.

6.3 Training Data

The package `nauck.data` (see Figure 6.3) provides the `DataTable` class that stores data in from of a table (single relation) with one row for each pattern and one column for each variable. The class can read itself from an ASCII file that must use a special format (see Chapter 5.6). If the format of the file is invalid, the class throws a `ParseData-FileException.` Reading is done in a separate thread so the execution of the main program is not blocked

The class is also able to arrange the data in random sequences and to draw stratified samples either by providing a percentage value or by specifying how many subsets have to be formed. Thus subsets for training, testing or validation can easily be created. The class can also compute some statistics of the data like mean, variance, ranges and correlations.

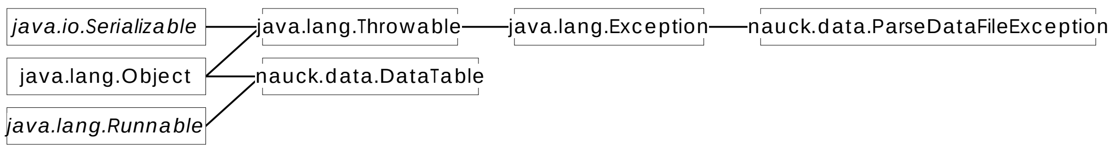


Figure 6.3: The class hierarchy of the package `nauck.data`

6.4 Utility Classes

The package nauck.util contains some utility classes that are either used throughout all other packages and classes (like FormatString) or that did not fit into the fuzzy or data packages. The latter is true for the classes `FunctionPane`, `FunctionDrawContext` and `FunctionLayoutPanel` which provide the functionality of drawing graphs in a coordinate system and display them on screen. They are very similar to the correponding classes of the `nauck.fuzzy` package for drawing fuzzy partitions.

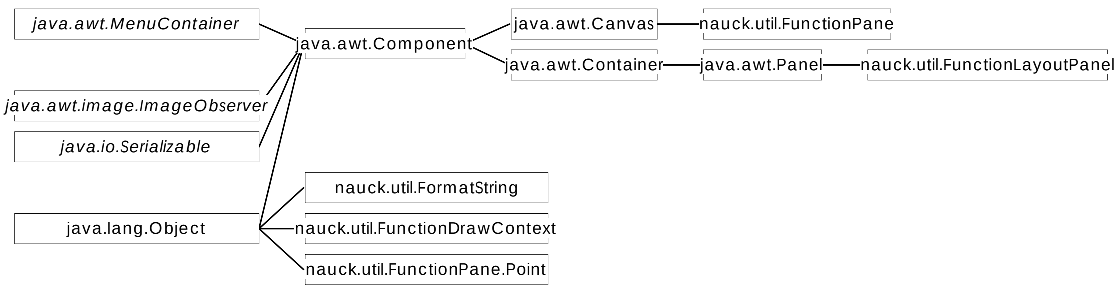


Figure 6.4: The class hierarchy of the `nauck.util` package

The class `FormatString` provides means to format strings and numbers. This class is by far not as powerful as the formatting classes from the `java.text` package, but it is easier to use for creating text output.

7

Conclusions

In this thesis an new implementation of NEFCLASS - a neuro fuzzy approach for classification of data - was described. Previous versions of NEFCLASS that were either available for MS-DOS-based personal computers [Nauck et al. 96, UNauck 97] or Unix workstations [Hoferichter 96, Bode 97, Nauck/Kruse 98b] had several limitations. The new software tool was written in Java and can therefore be used on any platform for which an implementaton of the Java Virtual Machine is available.

NEFCLASS-J offers the following new features for creating neuro-fuzzy classifiers based on the NEFCLASS model:

- batch learning to remove the dependeny of the learning algorithm from the sequence of data,
- automatic cross validation to determine the validity of a classifier,
- automatic determination of the rule base size,
- handling of missing values,
- automatic pruning of a classifer to reduce its size and to increase its interpretability,
- a complete new GUI with a look and feel of standard applications.

NEFCLASS-J can be used in data analysis or data mining, both important areas of solving todays industrial problems. In data analysis it is often required to obtain results in a fast and inexpensive way. Often the precision of the solution is less important than its applicability and therefore its understandability. As its predecessors, NEFCLASS-J tries to create interpretable classifiers by applying constraints to the learning algorithm.

The philosophy that NEFCLASS-J follows is *not* to create a fuzzy classifier *automatically* from data, *but to support* the user in finding an appropriate system by being responsible for the tedious tasks of data preparation, optimizing the classifier and pruning it. The tool gives the user a maximum of control over the learning process without burdening him with petty tasks. The user can concentrate on the creation of a classifier that meets the requirements of the area of application.

The new implementation has taken almost all requirements into account that have been mentioned in [UNauck 97] where possible extensions to future NEFCLASS tools were discussed. Only one item - interconnected NEFCLASS modules - could not be implemented. However, the current research on the model has not yet provided the foundations for connecting several NEFCLASS modules and how to create such a sequence from data. So this topic remains for future work.

Other items that should be considered for new versions of NEFCLASS-J are

- give the user some more control over the pruning process,
- include other forms of rule learning, for example methods from machine learning like induction of decision trees,
- add simplification algorithms to remove superfluous fuzzy sets from partitions,
- provide an algorithm to automatically determine the number of fuzzy sets for each variable,
- provide other forms of fuzzy sets that are suitable for non-numeric data,
- provide more flexible data handling (selection of variables, filtering, import of standard formats, etc.),
- provide some more statistics on the training data,
- provide an integrated help function instead of calling an external HTML browser for displaying help information.

As the current implementation is written in Java, it is expected that it will not be necessary anymore to create a new implementation from scratch. The Java language seems to have a lot of potential for future applications, so there should be no need to change the programming language again. The object-oriented approach makes it very easy to extend the tool so it should be possible to implement future versions with little effort.

NEFCLASS-J continues the philosophy of its predecessors as it will also be available via the Internet. Although limited in functionality, the previous MS-DOS version was very popular and enjoyed over 5000 downloads. It is expected that NEFCLASS-J can continue this success story.

References

[BERENJI / KHEDKAR 93] H.R. Berenji and P. Khedkar (1993). *Clustering in Product Space for Fuzzy Inference.* In *Proc. IEEE Int. Conf. on Neural Networks 1993* , pp. 1402-1407. San Francisco.

[BODE 97] J. Bode (1997). *Verfahren zur Regel- und Variablenreduktion unter dem Neuro-Fuzzy-Klassifikationsansatz NEFCLASS*. Diplomarbeit, TU Braunschweig.

[HOFERICHTER 96] Th. Hoferichter (1996). *Entwurf und Implementierung eines Unix-basierten Softwaretools zur Realisierung des Neuro-Fuzzy-Klassifikationsansatzes NEFCLASS*. Diplomarbeit, TU Braunschweig.

[KRUSE ET AL. 91] R. Kruse, E. Schwecke and J. Heinsohn (1991). *Uncertainty and Vagueness in Knowledge based Systems. Numerical Methods*. Series Artificial Intelligence, Springer-Verlag, Berlin.

[KRUSE ET AL. 94] R. Kruse, J. Gebhardt and F. Klawonn (1994). *Foundations of Fuzzy Systems.* Wiley, Chichester.

[KRUSE ET AL. 95] R. Kruse, J. Gebhardt and F. Klawonn (1995). *Fuzzy-Systeme, 2. erweiterte Auflage.* Teubner, Stuttgart.

[NAUCK ET AL 96] D. Nauck, U. Nauck and R. Kruse (1996). Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS. In *Proc. Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS'96).* Berkeley, CA.

[NAUCK ET AL 97] D. Nauck, F. Klawonn and R. Kruse (1997). *Foundations of Neuro-Fuzzy Systems.* Wiley, Chichester.

[NAUCK / KRUSE 93] D. Nauck and R. Kruse (1993). *A Fuzzy Neural Approach Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation.* In *Proc. IEEE Int. Conf. on Neural Networks 1993*, pp. 1022-1027. San Francisco.

[NAUCK / KRUSE 95] D. Nauck and R. Kruse (1995). *NEFCLASS - A Neuro-Fuzzy Approach for the Classification of Data.*In K. George, J.H.Carrol, E. Deaton, D. Oppenheim and J. Heightower, eds: *Applied Computing 1995. Proc. 1995 ACM Symposium on Applied Computing, Nashvillle, Feb. 26-28*, pp. 461-465. ACM Press, New York.

[NAUCK/KRUSE 97] D. Nauck and R. Kruse (1998). *New Learning Strategies for NEFCLASS.* In *Proc. of the Seventh International Fuzzy Systems Association World Congress, IFSA'97, Vol. IV.* Prague.

[NAUCK/KRUSE 98] D. Nauck and R. Kruse (1998). *NEFCLASS-X: A Soft Computing Tool to build Readable Fuzzy Classifiers.* In *BT Technology Journal, Vol. 16, No. 3, July 1998.* Martelsham Heath Ipswich.

[NAUCK / KRUSE 98b] D. Nauck and R. Kruse (1998). *Obtaining Interpretable Fuzzy Classification Rules From Medical Data*. In *Artificial Intelligence in Medicine*

[NAUCK / KRUSE 98c] D. Nauck and R. Kruse (1998). *How the Learning of Rule Weights Affects the Interpretability of Fuzzy Systems*. In *Proc. IEEE Int. Conf. Fuzzy Systems (FUZZIEEE'98).* Anchorage, Alaska.

[ROSENBLATT58] F. Rosenblatt (1958). *The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain.* In *Psychological Review*, 65:386-408.

[RUMELHARDT/MCCLELLAND86] D.E. Rumelhart and J.L. MCCLELLAND, eds. (1986). *Parallel Distributed Processing:Explorations in the Microstructures of Cognition. Foundations*, Band 1. MIT Press, Cambridge.

[SULZBERGER ET AL.93] S.M Sulzberger, N.N. Tchichold-Gürman and S.J. Vestli (1993). *FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks*. In *Proc. IEEE Int. Conf. on Neural Networks 1993,* pp. 1022-1027. San Francisco.

[SYMANTEC 97] Symantec Corporation (1997). *Symantec Visual Cafe for Java (Windows NT, Windows 95), Getting Started.* Cupertino.

[TSCHICHOLD GÜRMAN 95] N. Tschichold-Gürman (1995). *Generation and Improvement of Fuzzy Classifiers with Incremental Learning Using FuzzyRuleNet*. In K. George, J.H.Carrol, E. Deaton, D. Oppenheim and J. Heightower, eds: *Applied Computing 1995. Proc. 1995 ACM Symposium on Applied Computing, Nashvillle, Feb. 26-28,* pp. 461-465. ACM Press, New York.

[UNauck 97] U. Nauck (1997). *NEFCLASS-PC: Ein Neuro-Fuzzy Klassifikationstool unter MS-DOS*. Studienarbeit. TU-Braunschweig.

[Zadeh65] L.A. Zadeh (1965). *Fuzzy-Sets.* Information and Control, 8:338-353.