Improving Naive Bayes Classifiers Using Neuro-Fuzzy Learning[1]

A. Nürnberger, C. Borgelt, and A. Klose
Dept. of Knowledge Processing
and Language Engineering
Otto-von-Guericke-University of Magdeburg
Germany
andreas.nuernberger@cs.uni-magdeburg.de

Abstract

Naive Bayes classifiers are a well-known and powerful type of classifiers that can easily be induced from a dataset of sample cases. However, the strong conditional independence and distribution assumptions underlying them can sometimes lead to poor classification performance. Another prominent type of classifiers are neuro-fuzzy classification systems, which derive (fuzzy) classifiers from data using neural-network inspired learning methods. Since there are certain structural similarities between a neuro-fuzzy classifier and a naive Bayes classifier, the idea suggests itself to map the latter to the former in order to improve its capabilities.

1. Introduction

Naive Bayes classifiers are well-known and powerful classifiers that can easily be induced from a dataset of sample cases. However, the strong conditional independence and distribution assumptions underlying them can lead to poor classification performance, because the used type of probability distribution, e.g., normal distributions, may not be able to describe the data appropriately or (some of) the conditional independence assumptions do not hold.

Another prominent type of classifiers are neuro-fuzzy systems, which derive (fuzzy) classifiers from data using neural network inspired learning methods. Since there are some structural similarities between a neuro-fuzzy classifier and a naive Bayes classifier, the idea suggests itself to map the latter to the former in order to improve its capabilities.

For testing and analyzing of the presented approach we used an implementation of the NEFCLASS (**NE**uro **F**uzzy **CLASS**ification) model [11], since it was designed as an interactive classification tool. One of our goals is to develop algorithms that can learn automatically, but also allow a user to influence the learning and classification process, e.g. by initializing the system, and by modifying or extracting knowledge.

The paper is organized as follows: In section 2. we review naive Bayes classifiers and in section 3. we give a brief introduction to neuro-fuzzy classification systems. Section 4. describes how a naive Bayes classifier can be represented by a neuro-fuzzy classifier and how it can be improved by neuro-fuzzy learning. Section 5. discusses some implementation aspects. Experimental evaluation and conclusions are given in sections 6. and 7., respectively.

2. Naive Bayes Classifiers

Naive Bayes classifiers [4, 3, 7, 8] are an old and well-known type of classifiers, i.e., of programs that assign a class from a predefined set to an object or case under consideration based on the values of attributes used to describe this object or case. They do so using a probabilistic approach, i.e., they try to compute conditional class probabilities and then predict the most probable class. To be more precise, let C denote a class attribute with a finite domain of m classes, i.e., $\mathrm{dom}(C) = \{c_1, \ldots, c_m\}$, and let $A_1, \ldots, A_n$ be a set of (other) attributes used to describe a case or an object of the domain under consideration. These other attributes may be symbolic, i.e., $\mathrm{dom}(A_j) = \{a_1^{(j)}, \ldots, a_{m_j}^{(j)}\}$, or numeric, i.e., $\mathrm{dom}(A_j) = \mathbb{R}$. For simplicity, we always use the notation $a_{i_j}^{(j)}$ for a value of an attribute A_j, independent of whether it is a symbolic or a numeric one.[2] With this notation, a case or an object can be described by an instantiation $\omega = (a_{i_1}^{(1)}, \ldots, a_{i_n}^{(n)})$ of the attributes $A_1, \ldots, A_n$ and thus the universe of discourse is $\Omega = \mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_n)$.

For a given instantiation ω, a naive Bayes classifier tries to compute the conditional probability

$$P(C = c_i \mid \omega) \quad = \quad P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})$$

for all c_i and then predicts the class c_i for which this probability is highest. Of course, it is usually impossible to store all of these conditional probabilities explicitly, so that a simple lookup would be all that is needed to find the most probable class. If there are numeric attributes, this is obvious (we need some parameterized function then). But even if all attributes are symbolic, such an approach most often is infeasible: We have to store a class (or a class probability distribution) for each point of the Cartesian product of the attribute

[1] Acknowledgment: The research presented in this paper is partly funded by DFG contract KR 521/3-2.

[2] To be able to use this notation for numeric attributes, we simply have to choose an appropriate uncountably infinite index set $\mathcal{I}_j$, from which the index i_j is to be taken.

domains, whose size grows exponentially with the number of attributes. To circumvent this problem, naive Bayes classifiers exploit—as their name already indicates—Bayes rule and a set of conditional independence assumptions. With Bayes rule

$$P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)},$$

where X and Y are events, the conditional probabilities are inverted. That is, naive Bayes classifiers consider[3]

$$\begin{aligned} &P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \frac{f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_i) P(C = c_i)}{f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})} \end{aligned}$$

Of course, for this inversion to be always possible, the probability density function $f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})$ must be strictly positive.

There are two observations to be made about the inversion carried out above. In the first place, we can neglect the denominator of the fraction on the right, since for a given case or object to be classified, it is fixed and therefore does not have any influence on the class ranking (which is all we are interested in). In addition, its influence can always be restored by normalizing the distribution on the classes, i.e., we can exploit

$$\begin{aligned} &f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \sum_{j=1}^{m} f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_j) P(C = c_j). \end{aligned}$$

It follows that we only need to consider

$$\begin{aligned} &P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \frac{P(C = c_i)}{S} f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_i), \end{aligned}$$

where S is a normalization constant.

Secondly, we can see that just inverting the probabilities does not buy us anything, since the probability space is just as large as it was before the inversion. However, here the second ingredient of naive Bayes classifiers, which is responsible for the "naive" in their name, comes in, namely the conditional independence assumptions. To exploit them, we first apply the chain rule of probability:

$$\begin{aligned} &P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \frac{P(C = c_1)}{S} \\ &\quad \cdot \prod_{j=1}^{n} f\Big(A_j = a_{i_j}^{(j)} \,\Big|\, \textstyle\bigwedge_{k=1}^{j-1} A_k = a_{i_k}^{(k)}, C = c_i\Big) \end{aligned}$$

Now we make the crucial assumption that (given the value of the class attribute), any attribute A_j is independent of any other. That is, we assume that knowing the class is enough to determine the probability (density) for a value $a_{i_j}^{(j)}$, i.e., that we need not know the values of any other attributes. Of course, this is a pretty strong assumption, which is very likely to fail. It is truly "naive" to make it nevertheless. However, it considerably simplifies the formula stated above, since with it we can cancel all attributes A_j appearing in the conditions:

$$\begin{aligned} &P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \frac{P(C = c_i)}{S} \prod_{j=1}^{n} f(A_j = a_{i_j}^{(j)} \mid C = c_i) \end{aligned} \tag{1}$$

This is the fundamental formula underlying naive Bayes classifiers. For a symbolic attribute A_j the conditional probabilities $P(A_j = a_{i_j}^{(j)} \mid C = c_i)$ can be stored as a simple conditional probability table. This is feasible now, since there is only one condition and hence only $m \cdot m_j$ probabilities have to be stored.[4] For numeric attributes it is usually assumed that the probability density is a Gaussian function (a normal distribution) and hence only the expected values $\mu_j(c_i)$ and the variances $\sigma_j^2(c_i)$ need to be stored in this case. Alternatively, numeric attributes may be discretized [2] and then treated like symbolic attributes. In this paper, however, we make the normal distribution assumption.

Naive Bayes classifiers can easily be induced from a dataset of preclassified sample cases. All we have to do is to estimate the conditional probabilities/probability densities $f(A_j = a_{i_j}^{(j)} \mid C = c_i)$ using, for instance, maximum likelihood estimation. For symbolic attributes, this yields

$$\hat{P}(A_j = a_{i_j}^{(j)} \mid C = c_i) = \frac{\#(A_j = a_{i_j}^{(j)}, C = c_i)}{\#(C = c_i)},$$

where $\#(C = c_i)$ is the number of sample cases that belong to the class c_i and $\#(A_j = a_{i_j}^{(j)}, C = c_i)$ is the number of sample cases belonging to class c_i and having the value $a_{i_j}^{(j)}$ for the attribute A_j. To

[3] For simplicity, we always use a probability density function f, although this is strictly correct only, if there is at least one numeric attribute. If all attributes are symbolic, this should be a probability P. The only exception is the class attribute, since it necessarily has a finite domain.

[4] Actually only $m \cdot (m_j - 1)$ probabilities are really necessary. Since the probabilities have to add up to one, one value can be discarded from each conditional distribution. However, in implementations it is usually much easier to store all probabilities.

ensure that the probability is strictly positive (see above), it is assumed that there is at least one example for each class in the dataset. Otherwise the class is simply removed from the domain of the class attribute. If an attribute value does not occur given some class, its probability is either set to $\frac{1}{2N}$, where N is the number of sample cases, or a uniform prior distribution of $\frac{1}{N}$ is always added to the estimated distribution, which is then renormalized (Laplace correction). For a numeric attribute A_j the standard maximum likelihood estimation functions

$$\hat{\mu}_j(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} a_{i_j(k)}^{(j)}$$

for the expected value, where $a_{i_j(k)}^{(j)}$ is the value of the attribute A_j in the k-th sample case belonging to class c_i, and

$$\hat{\sigma}_j^2(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} \left(a_{i_j(k)}^{(j)} - \hat{\mu}_j(c_i)\right)^2$$

for the variance can be used.

3. Neuro-Fuzzy Classification

Fuzzy rules are well-suited to represent classification knowledge. It is mainly the abstraction from numbers to linguistic variables that makes them very easy to read and to interpret. In addition, fuzzy rules are applied in a very intuitive and comprehensible fashion to classify new data.

A fuzzy classification system [5] consists of a rule base, which contains a set of fuzzy classification rules (like the one shown below), and an inference engine, which evaluates the rule base for the datum to be classified. The basic idea of fuzzy classification systems is to describe the areas of the input space, to which different class labels are assigned, by vague cluster prototypes. These prototypes are defined by a number of fuzzy sets which characterize them in the different dimensions of the domain under consideration. That is, a specific cluster β labeled with class c is defined by a fuzzy classification rule r of the form:[5]

if A_1 is μ_1 **and** A_2 is μ_2 **and** ... **and** A_n is μ_n **then** pattern $(A_1, A_2, \ldots, A_n)$ belongs to class c,

where the μ_j are fuzzy sets describing the cluster β w.r.t. attribute A_j. In addition, some approaches introduce so-called rule weights w_r, which are intended to indicate the "importance" or "reliability" of a rule.

The degree of fulfillment of a rule is calculated from the membership degrees of the antecedents with a t-norm, usually $\top_{\min}$ or $\top_{\text{prod}}$. For example, if the t-norm $\top_{\text{prod}}$ is used, the degree of fulfillment or activation $o_r(\omega)$ of rule r at $\omega \in \Omega$ is defined as:

$$\begin{aligned} o_r(\omega) &= o_r\big(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}\big) \\ &= \prod_{j=1}^{n} \mu_j\big(A_j = a_{i_j}^{(j)}\big) \end{aligned} \tag{2}$$

To compute the output of a fuzzy classification system, first the degree of fulfillment of each rule in the rule base is calculated. Then, for each class, the sum, the maximum or the average of the (possibly weighted) rule activations is computed. (In the following we use the sum.) The output class is determined by a winner-takes-all principle, i.e., the class with the highest accumulated (weighted) activation is predicted.

One way to obtain fuzzy classifiers from data are neuro-fuzzy systems, that use learning algorithms derived from neural network theory to generate fuzzy rules [1, 10, 12]. To test and to analyze our approach we used an implementation of the NEFCLASS model [11].

Rule induction in systems like NEFCLASS, that start with a fixed number p_j of manually defined or equally spaced fuzzy sets as a partitioning of the domain of attribute A_j, is pretty straightforward: First the rule antecedents are constructed. To this end the sample cases are inspected in turn. For each case the fuzzy sets are evaluated and for each dimension that fuzzy set is selected, which yields the highest membership degree. Then a rule is constructed for each distinct selection of fuzzy sets, which forms the antecedent of the rule. The consequent of a rule is determined from the classes of all sample cases covered by the rule. In NEFCLASS the activations of the rule are summed per class over the (covered) sample cases and then the class with the highest activation sum is chosen as the consequent. In the learning phase the fuzzy partitioning of the input dimensions is adapted (i.e., the parameters of the fuzzy sets are changed) in order to optimize the location and extension of the clusters.

Commonly observed problems of neuro-fuzzy classifiers are either a huge number of rules (that are hard to read and to interpret) or a sparse covering of the input space (sometimes resulting in insufficient

[5] Note that in this section the letter μ denotes a fuzzy set and not, as in the previous section, an expected value. We regret this equivocation, but prefer it to introducing confusion by deviating from the standard notation of statistics or fuzzy set theory.

generalization). Both make pruning an important step in rule induction, which was a field of interest of our previous research [6].

4. Representing Naive Bayes Classifiers by Neuro-Fuzzy Systems

A naive Bayes classifier can be mapped to a neuro-fuzzy classification system, if the t-norm $\top_{\text{prod}}$ is used and some restrictions are placed on the fuzzy sets and the rule weights. In particular, we have to take care of the fact that probability distributions/density functions are normalized to 1, i.e., $\sum_x P(x) = 1$ or $\int_x f(x) = 1$.

To simplify the following explanation, let us first assume that there is—in analogy to a naive Bayes classifier—only one cluster β_i per class c_i, described by a (fuzzy) rule r_i. With this restriction, the membership functions $\mu_{r_i,j}$ can be used to define the probability density functions f for each attribute A_j given the class:

$$\mu_{r_i,j}(x_j = a_{i_j}^{(j)}) \;:=\; f(A_j = a_{i_j}^{(j)} \mid C = c_i) \qquad (3)$$

Furthermore, the rule weight w_{r_i} must represent the prior probability of the class c_i:

$$w_{r_i} \;:=\; P(C = c_i). \qquad (4)$$

With equations (1), (2) and $w_{r_i} \to w_{r_i} \cdot S^{-1}$, we obtain

$$\begin{aligned} &P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ &= \quad o_{r_i}(x_1 = a_{i_1}^{(1)}, \ldots, x_n = a_{i_n}^{(n)}) \cdot w_{r_i}. \end{aligned}$$

In this way the output of the (fuzzy) rule r_i represents the probability that a given case belongs to class c_i in direct analogy to a naive Bayes classifier.

However, in general the membership functions defined by equation (3) will not meet the usual requirements for fuzzy sets, i.e., $\exists x : \mu(x) = 1$ and $\forall x : \mu(x) \in [0, 1]$ (a probability density function can assume values greater than one). Therefore we scale each membership function $\mu_{r_i,j}$ of the rule r_i so that its maximum value becomes 1 and introduce a rule weight

$$w_{r_i}^{(\text{volume})} \;:=\; \prod_{j=1}^{n} w_{r_i,j}^{(\text{volume})}, \qquad (5)$$

where $w_{r_i,j}^{(\text{volume})}$ is defined as

$$\begin{aligned} w_{r_i,j}^{(\text{volume})} \quad &:= \quad \frac{1}{\sup_{a \in \text{dom}\, A_j} f(A_j = a | C = c_i)} \\ &= \quad \int_{a \in \text{dom}(A_j)} \mu_{r_i,j}(A_j = a) \end{aligned}$$

to take care of this scaling. Then we redefine the rule weight to incorporate this factor

$$w_{r_i} \to w_{r_i} \cdot \frac{1}{w_{r_i}^{(\text{volume})}}.$$

Here $w_{r_i,j}^{(\text{volume})}$ is the area of the appropriately scaled fuzzy set, and therefore $w_{r_i}^{(\text{volume})}$ is the (hyper)volume of the cluster described by the corresponding rule. In other words, with the factor $w_{r_i,j}^{(\text{volume})}$ we normalize the joint fuzzy set to integral 1 to make it interpretable as a probability density function.

Let us now assume that a class is described by more than one rule. With this we go beyond the limitations of naive Bayes classifiers and gain flexibility to describe the conditional distributions. Intuitively, we split each class into a set of subclasses or clusters, each of which is described by a separate (fuzzy) rule. Nevertheless, it must be admitted that in order to ensure interpretability, neuro-fuzzy systems also introduce restrictions that are not present in naive Bayes classifiers. Whereas in a naive Bayes classifier there are always as many (independent) distribution functions per dimension of the input space as there are classes, in a neuro-fuzzy system the number of fuzzy sets per dimension is fixed by the chosen fuzzy partition. If there is only one rule per class, this is no real restriction, since in most classification problems the number of classes is very small. With more than one rule per class, however, this limits the degrees of freedom. It should not be seen as a drawback, though, because too many degrees of freedom tend to lead to overfitting and thus poor generalization capabilities.

With more than one rule per class, the requirements the fuzzy sets have to meet are, of course, the same as above. Only the derivation of the rule weights is little more complex, since we have to consider the prior probability for each class c_i *and* the conditional probability that a sample case for this class belongs to the cluster β_{ik}, i.e., the k-th cluster describing class c_i which is represented by rule r_{ik}. For simplicity we use two rule weights: $w_{r_{ik}}^{(\text{class})}$, which states the prior probability of class c_i, and $w_{r_{ik}}^{(\text{cluster})}$, which states the conditional probability of cluster β_{ik} given that the case belongs to class c_i. The weights $w_{r_{ik}}^{(\text{class})}$ are defined as the w_{r_i} in (4) and the weights $w_{r_{ik}}^{(\text{cluster})}$ are defined as

$$w_{r_{ik}}^{(\text{cluster})} \;:=\; P(B = \beta_{ik} \mid C = c_i). \qquad (6)$$

If we want to use fuzzy sets to represent the probability density functions linguistically to ensure the interpretability of the learned (fuzzy) rules, we have

to use the scaling factors defined in equation (5). In our approach, however, this is primarily motivated by the fact that we want to use the NEFCLASS program to do the learning, and NEFCLASS is programmed to handle fuzzy sets. So we finally obtain the rule weight $w^*_{r_{ik}}$:

$$w^*_{r_{ik}} := \frac{w^{(\text{class})}_{r_{ik}} \cdot w^{(\text{cluster})}_{r_{ik}}}{w^{(\text{volume})}_{r_{ik}}} = \frac{P(C = c_i) \cdot P(B = \beta_{ik} \mid C = c_i)}{\text{volume}(r_{ik})}. \quad (7)$$

As shown, a naive Bayes classifier can be mapped to a neuro-fuzzy classifier. In addition, it is possible to use more than one cluster (i.e., more than one rule) per class to describe more complex distributions and so we may obtain a more powerful classifier. With this mapping neuro-fuzzy learning techniques can be used to learn and to optimize a naive Bayes classifier. Obviously, the learned probability distribution functions need not match the standard maximum likelihood estimation results, since the goal of the applied learning algorithms is to minimize the number of misclassifications and not to find the maximum likelihood estimate.

5. Implementation Aspects

To learn naive Bayes classifiers with neuro-fuzzy learning techniques any standard environment for neuro-fuzzy classification can be used, provided it supports the use of rule weights. As mentioned above, we used an implementation of NEFCLASS. However, it has to be ensured that the learning method does not modify the rule weights but computes them according to the formulae given below. The rule weights $w^*_{r_{ik}}$ defined in equation (7) have to be computed during the initialization process and after each learning step.

The rule weight $w^{(\text{class})}_{r_{ik}} := P(C = c)$ is computed just once during initialization by simply counting the number of cases d_i belonging to class c_i in the training set:

$$w^{(\text{class})}_{r_{ik}} := \frac{\#\text{cases belonging to } c_i}{\#\text{cases}}.$$

The rule weight $w_{r_{ik}} := P(R = r_{ik} \mid C = c_i)$ has to be computed during initialization and after each step of the learning algorithm (or at least in regular intervals during learning), since the number of cases covered by each rule changes during the learning process. As this is an iterative procedure, we write the weights as functions of timesteps t. Let $D = (d_1, \ldots, d_N)$ be the database of sample cases, $o_{r_{ik}}(d)$ the activation of rule r_{ik} for sample case $d \in D$, $w^*_{r_{ik}}(t)$ the rule weight in timestep t, and R_{c_i} the set of rules predicting class c_i. Then the weight $w^{(\text{cluster})}_{r_{ik}}(t+1)$ for the considered rule is calculated by the following two equations:

$$\rho_{d,r_{ik}}(t) := \frac{o_{r_{ik}}(d) w^*_{r_{ik}}(t)}{\sum_{r \in R_{c_i}} o_r(d) w^*_r(t)}$$

$\rho_{d,r_{ik}}$ is the fraction of case d that needs to be counted for rule r_{ik}. If d is covered by only one rule, $\rho_{d,r_{ik}}$ obviously equals 1. If d lies in the intersection of several rules, only a fraction according to the prior probabilities is counted in the second equation:

$$w^{(\text{cluster})}_{r_{ik}}(t+1) := \frac{\sum_{d \in D} \rho_{d,r_{ik}}(t)}{\sum_{r \in R_{c_i}} \sum_{d \in D} \rho_{d,r}(t)}$$

As the initial value for $w^{(\text{cluster})}_{r_{ik}}$ we use

$$w^{(\text{cluster})}_{r_{ik}}(0) = \frac{1}{|R_{c_i}|}.$$

The rule weights $w^{(\text{volume})}_{r_{ik}}$ that describe the volume of a cluster (see equation (5)) are calculated from the area under the membership functions. We obtain

$$w^{(\text{volume})}_{r_{ik}} := \prod_{j=1}^{n} \int_{A_j} \mu_{r_{ik},j}(x_j) \mathrm{d}x_j.$$

$w^{(\text{volume})}_{r_{ik}}$ needs to be recalculated every time the membership functions change.

6. Empirical Results

To demonstrate the effectiveness of our approach we present the results of the application to a dataset from the UCI machine learning repository [9], namely the Wisconsin breast cancer data. This dataset has 9 input features. There are 683 patterns (plus 16 patterns with missing values), that are assigned to 2 classes *benign* and *malign*. Tab. 1 shows the results of the classification. The naive Bayes classifier made 28 errors on the dataset. When only the two most significant features are used, the number of errors increases to 30.

We applied the modified NEFCLASS to the data with different partitions (i.e., fuzzy sets per input feature) and with different numbers of rules. When using only 2 inputs, NEFCLASS made 22 errors in each configuration. The configurations of NEFCLASS using one rule per class (i.e., c and g) are most similar to naive Bayes classification. However, NEFCLASS performs better with 17 (instead of 28) and 22 (instead of 30) errors. A look at the

Naive Bayes classifier				
	# inputs	# partitions	# rules	# errors
a	9	n/a	n/a	28
b	2	n/a	n/a	30
modified NEFCLASS				
	# inputs	# partitions	# rules	# errors
c	9	2	2	17
d	9	2	4	16
e	9	3	4	16
f	9	3	8	12
g	2	2	2	22
h	2	3	4	22
i	2	3	8	22

Table 1: Results: a) and b) naive Bayes classifier; c) ... i) modified NEFCLASS.

projections of the data shows, that they do not fit a Gaussian distribution very well. Therefore, the use of more rules gives NEFCLASS the possibility to represent more complex distributions and reduce the number of errors (configurations d, e and f). An interesting development could be observed in configuration f, which was initialized with 12 rules. The prior probability and hence the influence of 4 of those 12 rules decreased to zero during learning. After learning there were 2 rules for class malign and 6 for class benign.

7. Conclusions

The results of this paper are twofold. On the one hand the representation of naive Bayes classifiers in a neuro-fuzzy system improves classification results by direct optimization, and—with more than one rule per class—allows us to represent more complex, non-Gaussian distributions. After learning, the classification knowledge is represented in readable fuzzy rules. On the other hand, NEFCLASS as a neuro-fuzzy classification system gets a probabilistic basis, which offers a new mathematically founded interpretation, in particular of the rule weights, that were hardly justifiable before.

The original NEFCLASS software for UNIX (written in C++, with a user interface in TCL/TK) that was used to obtain the results described above can be freely obtained from the World Wide Web (`http://fuzzy.cs.uni-magdeburg.de`).

References

[1] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Trans. Neural Networks*, 3:724–740. IEEE Press, Piscataway, NJ, USA 1992

[2] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. *Proc. 12th Int. Conf. on Machine Learning (ICML'95, Lake Tahoe, CA, USA)*, 194–202. Morgan Kaufman, San Mateo, CA, USA 1995

[3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. J. Wiley & Sons, New York, NY, USA 1973

[4] I.J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA, USA 1965

[5] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis*. Kluwer, Amsterdam, Netherlands 1998

[6] A. Klose, A. Nürnberger, and D. Nauck. Improved NEFCLASS pruning techniques applied to a real world domain. *Proc. Neuronale Netze in der Anwendung (NN'99)*. University of Magdeburg, Magdeburg, Germany 1999

[7] P. Langley, W. Iba, and K. Thompson. An Analysis of Bayesian Classifiers. *Proc. 10th Nat. Conf. on Artificial Intelligence (AAAI'92, San Jose, CA, USA)*, 223–228. AAAI Press and MIT Press, Menlo Park and Cambridge, CA, USA 1992

[8] P. Langley and S. Sage. Induction of Selective Bayesian Classifiers. *Proc. 10th Conf. on Uncertainty in Artificial Intelligence (UAI'94, Seattle, WA, USA)*, 399–406. Morgan Kaufman, San Mateo, CA, USA 1994

[9] C.J. Merz and P.M. Murphy. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html

[10] D. Nauck, F. Klawonn, and R. Kruse. *Foundations of Neuro-Fuzzy Systems*. J. Wiley & Sons, Chichester, England 1997

[11] D. Nauck and R. Kruse. NEFCLASS — A Neuro-fuzzy Approach for the Classification of Data. *Proc. ACM Symposium on Applied Computing*, 461–465. ACM Press, New York, NY, USA 1995

[12] N. Tschichold-Gürman. Generation and Improvement of Fuzzy Classifiers with Incremental Learning using Fuzzy Rulenet. *Proc. ACM Symposium on Applied Computing*, 466–470. ACM Press, New York, NY, USA 1995