Vector Quantization with Rule Extraction for Mixed Domain Data

Barbara Hammer[1], Andreas Rechtien[1], Marc Strickert[1], and Thomas Villmann[2]

[1] University of Osnabrück, Department of Mathematics/Computer Science, Albrechtstraße 28, D-49069 Osnabrück, Germany, {hammer,arechtie,marc}@informatik.uni-osnabrueck.de

[2] Clinic for Psychotherapy and Psychosomatic Medicine, University of Leipzig, Karl-Tauchnitz-Straße 25, D-04107 Leipzig, Germany.

Abstract. We use a variant of learning vector quantization (LVQ) for extracting a rule based characterization of given labeled mixed domain data. Thereby, standard LVQ is improved to not only a more stable prototype calculation mechanism, but also to automatic detection of the importance of the different input data components by implementing an adaptive metric. This component weighting is related to the importance of certain components for providing a good data classification, and the obtained component ranking is then used to generate a classification tree from the prototypes by calculating natural splits of the input space.
We present the generalized relevance vector quantizer (GRLVQ), its extension, supervised neural gas (SRNG), and we show how both methods can be used for extracting so called BB-classification trees and rules from data. Artificial data, data from the UCI repository, and linguistic data are used in our experiments. Since the data domains might be a mixture of real values, integers, and discrete nominal data, we also discuss appropriate preprocessing techniques.

1 Introduction

In machine learning we find the principle paradigms: rule based data representation and soft distributed representation.
Rule based descriptions are particularly well suited for labeled symbolic data, since frequency and information measures can be obtained from their discrete domains in a straightforward manner without prior partitioning. Usually, for symbolic data an explicit ordering scheme is given, like rule inference, which can be represented as data driven decision tree with labeled leaves [1, 2]. Classic programs like FOIL [3] and GOLEM [4] both provide an induction of Horn clauses from data, but the domain of ILP has been extended to dynamic hypothesis generation [5], learning recursive logic [6], and program synthesis [7, 8].
Soft representations are especially suitable for real value data, because the natural order of numbers allows to approximate data by in-between and nearby

states. The canonic coding is a flat vector format, which is used in several statistical and neural data modeling approaches. Many of these methods try to catch the data characteristics using regression techniques and provide some kind of factor analysis for revealing the relationships between the data vectors' components. Traditional techniques such as the principle component analysis (PCA) fail for a proper description of multi-modal or nonlinear or labeled data.
Self-organizing maps [9] have been proposed as a density based nonlinear neural alternative to PCA [10]. The training resembles cortical activity: similar high dimensional stimuli are mapped to neighboring neurons located in a low dimensional grid of neurons. Naturally, the unsupervised similarity-based ordering of the map crucially depends on the metric used for comparing the high dimensional input. Recent work proposes a recursive design of self-organizing maps [11], incorporates an adaptive metric into [12]; also some work about rule extraction can be found [13], and even labeled structured data is dealt with [14]. These advances make the self-organizing maps (SOM) an interesting research field.
Closely related to the SOM are the learning vector quantizers (LVQ). Like SOM, they also provide a similarity based Hebbian update mechanism, but they are a priori designed for handling labeled data, and they are prototype based and thus do not require a grid of neurons with uncertain topology. For these reasons, we take LVQ as a root technology that we extend by an adaptive metric for learning the data representation.
In an additional step, we use trained LVQ networks for inferring logical rules of controllable complexity. This topic, rule extraction from neural networks, has been at stagnation for the last few years, but very recent work of Duch et al. about special multilayer perceptrons (C-MLP2LN) has brought this issue back into discussion [15]. Related work to rule extraction from neural networks has been done by Tickle et al. [16] and Andrews et al. [17], and encoding in neural networks is realized by the hybrid SHRUTI [18], BRAINN [19], and the recursive BUR architecture [20]. Contrary to graphs or multilayer networks, our focus lies on first order rule extraction in combination with the LVQ classification algorithm. Thus, integrating soft and crisp representation, we set the foundations for realizing a hybrid data model.

2 Learning Vector Quantization and Beyond

The original LVQ algorithm was developed by Kohonen [9]. LVQ is an Hebbian style *supervised* learner with prototypes competing for the classification of input space regions according to the regions' most prominent class occurrence: prototypes responsible for a certain class try to maximize their classification accuracy by adapting their locations within the data points until a good position is reached. This dynamic is referred to as a self-organizing process, and, since prototypes compete for maximizing the similarity to data points of their own class, this winner-take-all dynamic is referred to as *neural self-organization* with lateral inhibition.

Several versions of the prototype update strategy exist, and they are known as LVQ1, LVQ2, LVQ3, or OLVQ, which differ in robustness and the rates of convergence. A crucial step in the design of an appropriate update rule is the choice of a similarity measure which is a *distance metric* to determine which data points are located in the neighborhood of a prototype. An intuitive *cost function* for quantifying the goodness of fit for the prototypes is the sum of distances of all data points to their nearest correct prototype: if for a given prototype set the overall sum is at minimum, we expect a good classification. In most cases, a new data point with unknown class should then be mapped to its correct class by its closest prototype. Therefore, we use an update rule for the prototype locations which is based on a cost function for which a *gradient descent* can be formulated as suggested in [21].
In order to also determine the relevance of the input vectors' components we supply variable weighting factors to the distance metric. Since this *adaptive metric* is nested in the cost function, we can obtain the factors by a gradient descent on the cost function as well. The technical details how both the prototype locations and the global metric are adapted will be given below.
Our training procedure, the iterative minimization of the cost function, can be outlined as follows: (a) presentation of a labeled data point from the training set, (b) determination of the closest correct prototype and the closest wrong prototype, (c) realizing a stochastic online gradient descent for the prototype locations by moving the correct prototype towards the given point and moving the closest wrong prototype away, and (d) adaptation of the metric weighting factors using the formulas for the gradient descent. After training, small weighting factors indicate input dimensions which are irrelevant for the classification, and the data set may be projected to only important components.
Furthermore, the remaining ranked components can be used to calculate splits of the input space: with decreasing importance, the midpoints between according prototype components hierarchically separate the data space into hypercubes for which the class membership is known. Finally, since paths in the split chain represent **and**-decision rules pointing to hypercubes, *rule simplification* can be obtained by merging those hypercubes belonging to the same class [25]. This combination leads to **or**-concatenations of the **and**-rule chains, where trivial merges with shared edges can be further reduced by just extending the respective cube boundaries.
The following subsections provide an overview of the formal aspects of our proposed LVQ variants and the rule extraction procedure: we briefly review the two related algorithms GRLVQ [22] and SRNG [23], and the procedure to obtain a BB-classification [24] tree and a possibly simplified set of rules.

2.1 Generalized Relevance Learning Vector Quantization (GRLVQ)

Let's consider the clustering task for a set of training data $X = \{(x^i, y^i) \in \mathbb{R}^n \times \{1, \ldots, C\} \mid i = 1, \ldots, m\}$ for n-dimensional items $x^k = (x_1^k, \ldots, x_n^k)$ to C classes. For these classes a set of prototypes $W = \{w^1, \ldots, w^K\}$ in $\mathbb{R}^n$ is chosen, $w^i = (w_1^i, \ldots, w_n^i, c^i)$, with class labels c^i attached to the prototypes'

locations. The classified region, the receptive field R^i, of prototype w^i is given by any point x nearer to w^i than to any other prototype: if $d(x, y)$ denotes an inter-point distance ignoring class labels, then $R^i = \{x \in \mathbb{R}^n \mid \forall w^j \, (j \neq i \rightarrow d(x, w^i) \leq d(x, w^j))\}$.
Here, we implement an adaptive Euclidean metric $d(x, y) = \sqrt{\sum_i \lambda_i (x_i - y_i)^2}$, with scaling factors $\lambda_i \geq 0$ for which $\sum_i \lambda_i = 1$. The cost function to be minimized is

$$C_{\text{GRLVQ}} := \sum_{i=1}^{m} \text{sgd}\left(\mu_\lambda(x^i)\right) \text{ where } \mu_\lambda(x^i) = \frac{d_\lambda^+(x^i) - d_\lambda^-(x^i)}{d_\lambda^+(x^i) + d_\lambda^-(x^i)} \, .$$

$\text{sgd}(x) = 1/(1 + \exp(-x))$ is the logistic function. $d_\lambda^+(x^i) = d_\lambda^2(x^i, w^{i+})$ is the squared weighted Euclidean distance of x^i to the nearest prototype w^{i+} with the same class as x^i, and $d_\lambda^-(x^i) = d_\lambda^2(x^i, w^{i-})$ is the distance of x^i to the nearest prototype w^{i-} of a different class. The update formulas for the closest correct and the closest wrong prototype and the metric weights, respectively, are obtained by taking the derivatives of the above cost function:

$$\triangle w^{i+} = \frac{4\epsilon^+ \text{sgd}'(\mu_\lambda(x^i)) d_\lambda^-(x^i)}{(d_\lambda^+(x^i) + d_\lambda^-(x^i))^2} \cdot \Lambda \cdot (x^i - w^{i+})$$

$$\triangle w^{i-} = \frac{-4\epsilon^- \text{sgd}'(\mu_\lambda(x^i)) d_\lambda^+(x^i)}{(d_\lambda^+(x^i) + d_\lambda^-(x^i))^2} \cdot \Lambda \cdot (x^i - w^{i-})$$

$$\triangle \lambda_j = \frac{-2\epsilon \, \text{sgd}'(\mu_\lambda(x^i))}{(d_\lambda^+(x^i) + d_\lambda^-(x^i))^2} \left(d_\lambda^-(x^i)(x_j^i - w_j^{i+})^2 - d_\lambda^+(x^i)(x_j^i - w_j^{i-})^2 \right)$$

Λ is the diagonal component weight matrix with entries $\lambda_1, \ldots, \lambda_n$; the non-negative adaptation rates are ϵ^+, ϵ^- and ϵ.
The update mechanism reflects Hebbian learning style by the difference terms $(x^i - w^{i-|+})$. Setting the weighting factors λ_i fixed to $1/n$ yields generalized LVQ (GLVQ). For variable λ_i as above we also obtain relevance adaptation for the input dimensions, which is a generalized relevance learning vector quantization algorithm, GRLVQ in short [22].

2.2 Supervised Relevance Neural Gas (SRNG)

No word has yet been said about prototype initialization: since GRLVQ 'pulls' correct prototypes and 'pushes away' wrong prototypes, attention must be payed to the case of prototype configurations, for which well fitted prototypes prevent badly fitted from reaching their data cluster; implicitly, the dislocated prototypes might be pushed away, barricaded by the well located to access better regions. In high dimensions with many degrees of freedoms, such suboptimal cases for which attraction and repulsion are at equilibrium are rare. Nevertheless, convergence is significantly faster for fairly initialized prototypes, and a way to achieve this is discussed in the following.

A common method for prototype based density representation of unlabeled data is given by the neural gas (NG) algorithm [26]. The essential idea is to adapt the NG prototypes according to a neighborhood ranking: the nearest prototype to a presented data point is adapted strongest, while the most distant prototype is affected weakest. This kind of neighborhood cooperation is now incorporated into GRLVQ.

For a class given by a training pattern, the class-related prototypes perform rank based updates according to GRLVQ towards this pattern, and the closest wrong prototype is again pushed away. As a result, prototypes spread over the data according to a combination of the density based neural gas and the GRLVQ active class separation.

Given mathematical shape, using $W(y^i)$ the set of prototypes with label y^i of cardinality K_i, the cost function becomes

$$C_{\text{SRNG}} = \sum_{i=1}^{m} \sum_{w^j \in W(y^i)} h_\sigma(k_j(x^i, W(y^i))) \cdot \text{sgd}(\mu_\lambda(x^i, w^j))/h(K_i)\,,$$

$$\mu_\lambda(x^i, w^j) = \frac{d^2_\lambda(x^i, w^j) - d^-_\lambda(x^i)}{d^2_\lambda(x^i, w^j) + d^-_\lambda(x^i)}\,.$$

$h_\sigma(x) = \exp(-x/\sigma)$ controls the neighborhood interaction. The degree of neighborhood cooperation $\sigma > 0$ is decreased to 0 during training to fade over from coarse global ordering to fine local adaptation. $k_j(x^i, W(y^i)) \in \{0, \ldots, K_i - 1\}$ denotes the rank of w^j in $W(y^i)$ for prototypes sorted by the distances $d_\lambda(w^k, x^i)$. For normalizing purposes, we apply the inverse of $h(K_i) = \sum_{j=0}^{K_i-1} h_\sigma(j)$ to the cost function. For small neighborhoods C_{GRLVQ} is really an instance of C_{SRNG}, because $\lim_{\sigma\to 0} C_{\text{SRNG}} = C_{\text{GRLVQ}}$.

Again, the learning rule is obtained by taking the derivatives of the cost function. Given a training pattern (x^i, y^i), all $w^j \in W(y^i)$, the closest wrong prototype w^{i-}, and the factors λ_k are adapted:

$$\triangle w^j = \frac{4\epsilon^+ \,\text{sgd}'(\mu_\lambda(x^i, w^j))\; h_\sigma(k_j(x^i, W(y^i)))\; d^-_\lambda(x^i)}{(d_\lambda(x^i, w^j) + d^-_\lambda(x^i))^2\; h(K_i)} \cdot \Lambda \cdot (x^i - w^j)$$

$$\triangle w^{i-} = -4\epsilon^- \sum_{w^j \in W(y^i)} \frac{\text{sgd}'(\mu_\lambda(x^i, w^j))\; h_\sigma(k_j(x^i, W(y^i)))\; d^2_\lambda(x^i, w^j)}{(d^2_\lambda(x^i, w^j) + d^-_\lambda(x^i))^2\; h(K_i)} \cdot \Lambda \cdot (x^i - w^{i-})$$

$$\triangle \lambda_k = -2\epsilon \sum_{w^j \in W(y^i)} \frac{\text{sgd}'(\mu_\lambda(x^i, w^j))\; h_\sigma(k_j(x^i, W(y^i)))}{(d^2_\lambda(x^i, w^j) + d^-_\lambda(x^i))^2\; h(K_i)} \cdot \left(d^-_\lambda(x^i)(x^i_k - w^j_k)^2 - d^2_\lambda(x^i, w^j)\,(x^i_k - w^{i-}_k)^2 \right)$$

As desired, for $\sigma \to 0$ GRLVQ is re-obtained (after some calculation).

The training of a given data set can be done in several modes: prototype adaptation only, relevance λ adaptation only, or both combined. Empirically, it turns out that keeping the λ_i fixed at the beginning and releasing them after a number of training cycles on a time scale larger than the prototype adaptation process, i.e. $\epsilon << \epsilon^{+|-}$, this will provide a good characterization of how relevant input dimensions are for the more or less settled prototypes.

2.3 BB-Trees: From Prototypes to Rules

We need the following ingredients for turning a trained GRLVQ or likewise a SRNG network into a decision tree: properly located prototypes and a ranking of the relevances of the input. Roughly speaking, we use the locations of the prototypes as centers for axes parallel bounding boxes enclosing the data points. Due to the ranking, though, we only need to take the most relevant dimensions into consideration. In a recursive manner, starting at the root node, we assemble a tree with these properties: each node N of the tree may possess an arbitrary number C^N of children; an interior node N is labeled by an index $I^N \leq n$ and real values $W_1^N < \ldots < W_{C^N-1}^N$; each leaf L is labeled with a class number $C_L \leq C$.

Now, classifying a data point requires taking a decision at each interior node which child-related path to follow. This is accomplished by projecting the point to its I^N th dimension and selecting the surrounding interval from the node's ordered list of real numbers. Since an interval refers to the next child node, the point successively percolates to a leaf node containing the class label, and the classification for this point is done.

For efficient tree construction it is a good idea to discriminate the most selective dimensions first; these are easily available by ranking the trained GRLVQ/SRNG dimensions' metric weights. The nodes' real number entries, the interval borders, are given by the midpoints of adjacent prototypes projected to the current dimension.

Putting all together we formulate the BB-tree generator with these arrangements: Λ is the list of indices i sorted according to the magnitude of the weighting factors λ_i; *first*(Λ) denotes its first entry, *rest*(Λ) the rest of the list; X denotes the training set and W is the set of prototypes. For reasons of efficiency, we assume that all dimensions i which, due to a small weighting factor λ_i, do not contribute to the classification are ignored, and all prototypes with empty receptive fields are removed before starting the following procedure:

BB-Tree (X,W,Λ)*:*
if STOP: output a leaf with class $argmax_c|\{x^i \mid y^i = c, (x^i, y^i) \in X\}|$
else: output an interior node N *with* $|W|$ *children,*
choose $I^N :=$ *first*(Λ)*,*
compile a sorted list $[a_1, \ldots, a_{|W|}]$ *from* $\{w^i_{I^N} \mid w^i \in W\}$
choose $W_i^N := (a_i + a_{i+1})/2$*,* $i = 1, \ldots, |W| - 1$ $(*)$
choose the i *th child of* N*,* $i = 1, \ldots, |W|$*, as the output of*
BB-Tree $(\{(x, y) \in X \mid x_{I^N} \in (W_{i-1}^N, W_i^N]\}, W,$ *rest*(Λ) $\bullet$ [*first*(Λ)])

As indicated by the list concatenation operator $\bullet$ and the recursive formulation, the height of the tree is technically not limited by the number $\tilde{n}$ of dimensions accounted for in the rank list. Instead, the stopping criterion *STOP* controls the tree's height and thus the length of the rule chains. In our experiments, we stop if the tree is higher than $n_S \leq \tilde{n}$, empirically choosing n_S according to satisfying classification results for both training and test set, without producing too

many rules. Other stopping criteria are useful, when smarter alternatives to the midpoint interval calculation are considered which prevent recursive repetitions of the same intervals; then, trees higher than $\tilde{n}$ of possibly arbitrary accuracy, though higher granularity and lower generality, might be obtained, which must be the subject of further investigations.

Rule Simplification As said before, single paths in the tree correspond to a rule of **and**-combined decisions which is equivalent to hypercube based representation. Different cubes containing the same class can be merged by **or**, which in adjacent cases can be achieved by interval extension. Also, disjoint cubes of the same class can be merged to their convex hull, if only a negligible amount of points belonging to other classes is located in this region.

3 Experiments

In order to shed light on how our approach performs on different kind of data, we use (a) an exclusively real-value artificial multi-modal data set, (b) discrete nominal data, the mushroom data from the UCI repository learning repository [27], and (c) discrete linguistic data concerning the diminutive prediction in Dutch, and (d) a mixture of real and boolean data, the hypothyroid data, also from the UCI repository. Prototypes and the metric weights have been determined by means of the SRNG algorithm.

3.1 Data Preparation

Some data preprocessing is useful before applying SRNG. Since real-life data, seldom live in the product space $D \times D \ldots \times D$ of the single domain $\mathbb{R}$, but appears as a mixture $D_1 \times D_2 \ldots \times D_n$ of different domains D_i, it is necessary to render data into the Euclidean space for establishing compatibility with the used adaptive Euclidean metric. For nominal data we suggest the unary bit encoding to represent the k different states of the concerned input dimension. Thus, all states span a simplex in $\mathbb{R}^k$, and the Hebbian term in the prototype update makes sure, that prototypes with valid initialization remain on the simplex' surface.
As will be demonstrated below, it might be advisable to visually inspect the data scatter plot matrix for continuous components. This will reveal relationships between pairs of data dimensions. Especially, class separability might be enhanced by a logarithmic transforms of exponential correlations, or double logarithmic transforms in the presence of power laws; both types of relationships are rather common for medical and real-life data.
Many training data sets exhibit strong asymmetries concerning the number of points for a certain class. This situation would yield a biased prototype update for GRLVQ and SRNG, because the frequent presentation of data points from the largest class would implicitly push away prototypes for smaller classes in the neighborhood. Therefore, the training set should be augmented to obtain a balanced class representation.

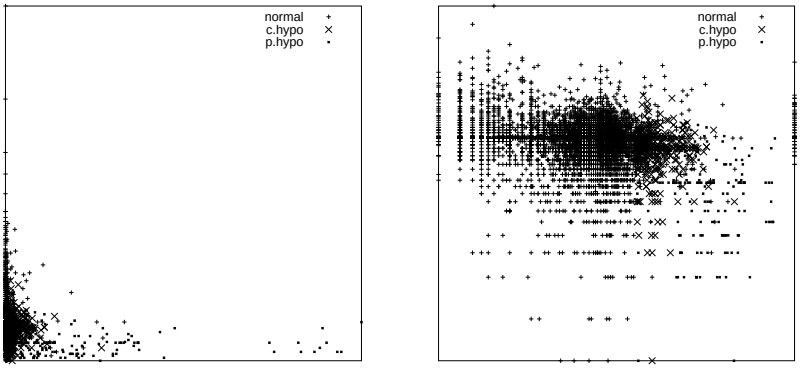

Fig. 1. Data preparation. Left: scatter plot of original data, hypothyroid features TSH versus T3. Right: same data after logarithmic and z-transform.

Normalization is suggested in the final preparation step. Rescaling the data components to the interval $[0, 1]$ is not tolerant to outliers, therefore, mean subtraction followed by rescaling the standard deviation to 1 is preferred, which is in statistics is known as z-transform. The mean and standard deviation for each dimension are saved for being applied to the test sets and to new data, and for reconverting the obtained rule interval boundaries back to the original domain. In figure 1 the positive effect of transforms becomes visible, exemplary for two features of the hypothyroid data used later. In the left plot, the data points are not well spread over the two dimensions, but they are clustered in a linear axes-parallel shape: roughly, the attribute 'normal function' is located near the left axis, 'compensated hypothyroid' states are near the lower left corner, and the 'primary hypothyroid' mode is rather at the bottom line. This point configuration is not suitable to be captured and separated into the three classes by the receptive fields, the Voronoï regions, of only few data prototypes.
The right plot shows the same data spreading faithfully in the plane after taking the logarithms, followed by a z-transform. Class clusters are turned into a more circular shape which is good for the prototype representation. The horizontal and vertical structures display the recording threshold, the resolution of the data set. At a glance, the data can still hardly be separated, but since there are more dimensions left in the data set, we expect that taking all of them into consideration will improve the prototype based description.
A benefit of relevance based clustering is that transformed data can just be added as new dimensions to the data, and if these dimensions turn out to be worthless for the separation task, their metric weight factors just vanish.

3.2 Artificial Data

As a first test, we use an artificial data set with overlap. We embed points in $\mathbb{R}^{10}$ as follows: starting with points $(x_1; x_2)$, we add 8 dimensions obtaining points

$(x_1; \ldots; x_{10})$. We choose $x_3 = x_1 + \eta_1$, $\ldots$, $x_6 = x_1 + \eta_4$, where η_i is distributed according to Gaussian noise, with variances 0.05, 0.1, 0.2, and 0.5; $x_7, \ldots, x_{10}$ contain randomly scaled white noise. This generator has been used to produce a multi-modal distribution for 3 classes. Then, data has been z-transformed and trained with SRNG.
A number of 3 prototypes per class has been used, and the training parameters have been set to $\epsilon^+ = 0.1$, $\epsilon^- = 0.05$ and $\epsilon = 10^{-5}$ with an initial neighborhood size of 9 that rapidly decayed to zero. Several training runs with 10,000 epochs produced an average accuracy of 85%, and the best of which with 93% has been used for rule extraction. Its weighting factors λ_i display a typical result for the runs: $\lambda = (0.27; 0.45; 0.28; 0; 0; 0; 0; 0; 0; 0)$. These values show that the important first two data dimensions are clearly separated, and also the small variance of 0.05 on the first component for the 3rd component has no negative effect on the classification.
A projection of the data to the two generating dimensions is plotted in figure 2. Moreover, the prototypes with training adaptation steps, as well as the extracted hypercubes are shown. From an extracted BB-tree of height 3, we have obtained 14 rules, which in a second step have been automatically reduced to a number of 6. Thereby, pairs of related hypercubes have been arranged in a hierarchy of unions in a binary tree and checked according to decreasing classification accuracy. Note, that neighboring boxes for the same class can merge without increasing the misclassification to more than the sum of the two, and also empty space might be bridged between disjoint rules. Manually, two more rules could be taken away by formulating an *else*-case for the (+)-shaped class, thus leading to a total number of 4 cubes, shown as projections in plot 2. The overall accuracy for this set of rules is 81.7%, which is lower than for the original trained SRNG, because axes parallel cuts usually do harm on overlapping classes, as in the present case. Here, convex receptive fields of the prototypes seem more suitable for the classification. Taking a second look on the boxes, though, reveals that the midpoint-between-prototype criterion could be refined in order to calculate cube boundaries that catch still more data.

3.3 Mushroom Data

The mushroom data set from the UCI repository consists of 8,124 vectors, each with 22 symbolic attributes, some of which are binary, others taking up to 12 different values. These attributes have been converted by unary encoding into vectors of dimension 117 and then z-transformed. The two class labels are 'e=edible', which represents 51.8% of the data, and 48.2% of 'p=poisonous' examples. 75% of this data set has randomly been taken for training, 25% for validation.
In a first training step over 1,000 epochs, 10 prototypes were used for each class with learning rates $\epsilon^+ = 0.1$, $\epsilon^- = 0.05$ with no weight adaptation $\epsilon = 0$, annealing the neighborhood size from 10 to almost 0. Calculating the number of data in the receptive fields showed, that 5 of the prototypes could be removed without significantly affecting the classification accuracy of roughly 92%, thus 3 prototypes were left for the poisonous case and 2 for the edible. Then, training

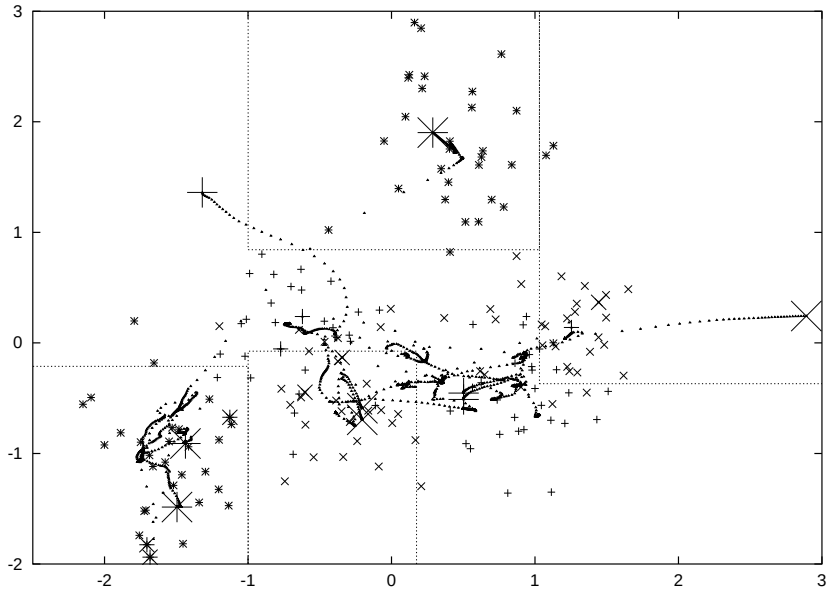

Fig. 2. Artificial data set with 3 classes, projected to its 2 original dimensions. Tiny symbols: data points. Medium symbols: initial prototype coordinates. Largest symbols: converged prototype locations. Dotted trajectories: paths of prototypes during training. Also shown: rule interval boundaries.

was refined with $\epsilon^+ = \epsilon^- = 0.075$, $\epsilon = 10^{-7}$, and an initial neighborhood size of practically zero. Tenfold cross-validation produced classification accuracies $\geq$ 97.5% on the test data. The best set of prototypes with its metric weights λ_i produced 98.7% accuracy, and it has been used for the BB-tree extraction procedure.

The relevances for the different components are shown in figure 3. Within the odor feature block, feature number 28, corresponding to odor:'none' out of 9 alternatives, displays highest discrimination ability. This feature is followed by 41.gill-color:'buff', 37.gill-size:'narrow', 36.gill-size:'broad', 101.spore-print-color:'chocolate', 22.bruises:'no', 21.bruises:'yes', 95.ring-type:'pendant', 93.ring-type:'large', and 27.odor:'foul', respectively. Obviously, for binary items like the bruises, both possible states complement each other.

The relevances are also reflected in the final set of rules which have been extracted for a BB-tree of height 6 and compiled in table 1. These rules explain 97.2% of the test set and 97.5% of the training set. A portion of 88.7% of the data can be classified using the single rule odor = 'none' $\rightarrow$ e, because the one conflicting rule accounts for only for 0.1% of the data. Still, redundancies are visible in the set of rules, because the gill-size attribute appears twice. At this point, the automatic rule simplification algorithm could not handle the don't care state (-), which will be improved in future.

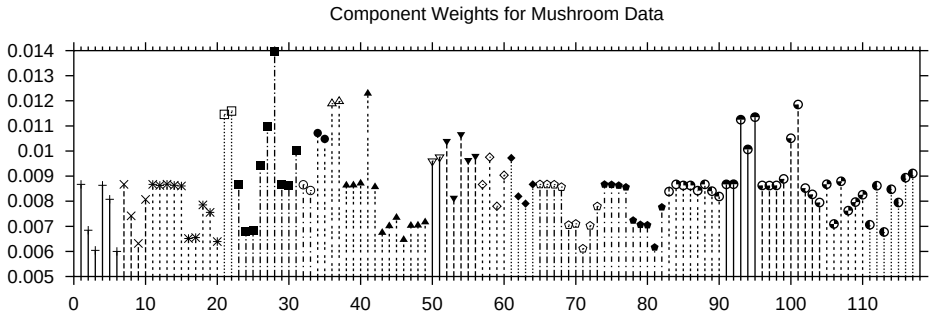

Fig. 3. Adapted weight factors $\lambda_{1 \le i \le 117}$ for the mushroom data. Different line styles and points indicate the membership to the 22 different attributes.

By means of the 4 rules for the edible case, using 6 components, we obtain a good data representation; the poisonous class is subsumed in the else case. Our findings are not quite as good as compared to the results of Duch et al. [15], who with 3 rules and 5 components obtain 100% accuracy. But interestingly, we have found the same important attributes: odor, gill-size, and spore-print-color.

3.4 Linguistic Data

In a preliminary test, we compared the SRNG-with-rule-extraction procedure to the nearest neighbor classifier TiMBL [28], which has been designed for the induction of linguistic knowledge. The TiMBL package supplies a test set with SAMPA coded syllable information of Dutch words, for which one of the 5 possible diminutive forms *-je*, *-etje*, *-pje*, *-kje*, and *tje* shall be predicted.
Since TiMBL is a nearest neighbor classifier, it stores training examples in memory, 2999 items for the diminutive data. Very roughly, new data is classified according to a majority vote of examples with best matching overlap metric. For the 950 test cases, an accuracy of 96.6% was achieved.

Table 1. Rules for the mushroom data set for a BB-tree of height 6.

bruises 22:'no'	odor 28:'none'	gill-size 36:'broad'	gill-size 37:'narrow'	gill-color 41:'buff'	spore-print-color 101:'chocolate'	Class	Freq.
-	0	-	-	1	-	p	21%
-	0	1	0	0	1	p	19%
0	0	-	-	0	0	p	3%
1	0	0	1	0	0	p	4%
1	1	0	1	0	-	p	0.1%
0	1	-	0	0	-	e	16%
1	0	1	0	0	0	e	8%
1	1	-	0	0	-	e	25%
0	1	0	1	0	-	e	3%

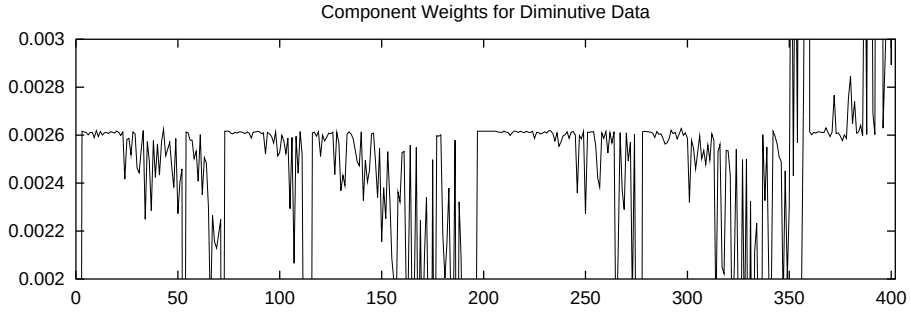

Fig. 4. Adapted weight factors $\lambda_{1 \leq i \leq 403}$ for the diminutive data.

SRNG training was performed with just 4 prototypes per class for the unary encoded, augmented, z-transformed data. Nevertheless, the final result for the prototype classification already gives 92.6% accuracy on the test set, and 92.3% on the training set. Figure 4 displays the metric weights. Since the data vector is organized in triples of ('stress','onset', 'nucleus', 'coda') aligned to the word endings, the weights reflect the intuitive fact that the word endings determine the choice of the proper diminutive suffix.

Another nice property can be obtained: calculating the rules for a BB-tree of height 25, we obtain an improved classification accuracy of 95.5% and 95.1% for the test and training set, respectively. Unfortunately, this height of 25 corresponds to a set of 117 rules, which cannot be simplified easily. After all, this result is not too bad, because it might give a hint for a dilemma in linguistic research: the question is, whether human linguistic skills are rule based rather than analogy based, or vice versa.

On the one hand, some rules have a high degree of generality and account for up to 10% of all cases. On the other hand, the large number of special rules indicate that many exceptions have to be learned extra. As a reference, the C4.5rules program [2] has generated 71 rules, and its accuracy is at about 97.1% for the test set and 97.5% for the training set.

3.5 Hypothyroid Data

A difficult separation task is given by the hypothyroid data from the UCI repository. There are three classes, 'normal', 'primary hypothyroid', and 'compensated hypothyroid', determined by 22 attributes, 15 of which are binary and 6 are continuous. In the training set, there are 3772 cases, and 3428 cases in the test set. Due to the large overlap between the classes and the strong prominence of the 'normal' class with a portion of about 92%, this data is known to be difficult for neural classification [29].

First SRNG runs with the original data reproduced the 92% accuracy, which can trivially be obtained, be classifying all data as 'normal'. In a second run with training set and test set augmented to about 10,000 cases each, this result

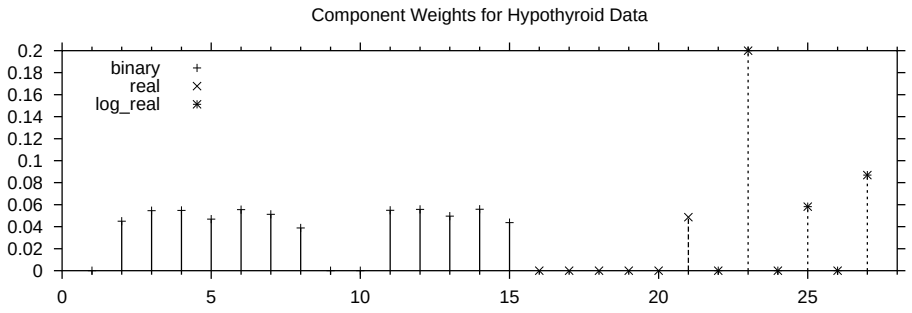

Fig. 5. Adapted weight factors $\lambda_{1\leq i\leq 27}$ for the hypothyroid data. Different line styles and points indicate the membership to the binary (left), real (center), and logarithms of the real attributes (right).

dropped to only 78%. Performing a z-transform did not significantly improve the situation.

A breakthrough was achieved by adding the logarithms of the real-value data. This step has been discussed in the data preparation section and has been supported by figure 1. For the weights shown in figure 5 and 5 prototypes per class, the classification accuracy increased to 97.1%. After augmentation, it turns out that now the correct prediction of the 'normal' class causes problems: the worst prototype for this class misclassifies 11.1% of the 'normal' cases, whereas the worst for 'compensated hypothyroid' only fails in 5.5% of the cases, and for the 'primary hypothyroid' class, no misclassification is done at all. In other words: if we predict one of the hypothyroid cases, we can be very certain about this result, whereas for 'normal' diagnosis we must not be too certain; this false alarm reduction induced by data augmentation might be a desirable feature in diagnostics.

Plot 5 indicates, that some of the binary features have been detected as irrelevant; moreover, only one of the original real components has survived, and the most prominent dimensions are 23:logarithmic TSH, and 27:logarithmic FTI, which confirms the findings of Duch et al. [15]. As related to the weight ranking shown in figure 5, the simplified extracted rules for a BB-tree of height 3 make are reduced to these two features. The three obtained rule are:

$$\begin{array}{llllll} \text{logZtsh}\in\,]-0.347;\,\infty] & \wedge & \text{logZfti}\in\,]-\infty;\,0.092] & \rightarrow\ 0 & (97\%)\,, \\ \text{logZtsh}\in\,]\,0.366;\,\infty] & \wedge & \text{logZfti}\in\,]\,0.068;\,\infty] & \rightarrow\ 1 & (94\%)\,, \\ \text{logZtsh}\in\,]-\infty;-0.347] & & & \rightarrow\ 2 & (100\%)\,. \end{array}$$

With these rules, 96.2% accuracy is obtained for the test set, and 97.0% for the training set. Increasing the hight of the extracted tree to 10, involves the binary data, too; then, many rules with little generality are generated: the accuracy of the training set increased to 99.1%, but the test set was only predicted correctly to an amount of 93.2%.

4 Conclusions

We have presented a brief introduction to the supervised GRLVQ and SRNG vector quantizers. These networks can be used to extract decision trees and hence first order rules from trained data, exploiting the ranking of the learned dimension relevances and the prototype locations in the data space. The method has been applied to four different data sets, all of which are difficult, exhibiting class overlap, mixed domains, or high dimensionality. The results are promising, and they might still be improved by refining the BB-classification tree extractor and the rule simplification strategy. Since the LVQ fundament of our method is inherently subsymbolic for which we now have access to rules, further work should focus on designing a real data-driven hybrid model with integrated prototype-rule representation. This will be an intriguing task for the near future.

Acknowledgment We gratefully thank the Ministry for Science and Culture (MWK) of Lower Saxony, Germany, for the financial support of this work.

References

1. J.R. Quinlan. Induction of Decision Trees. Machine Learning, Vol. 1, pp. 81–106, Kluwer Academic Publishers, 1986.
2. J.R. Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann Publishers, 1993.
3. J.R. Quinlan and R.M. Cameron-Jones. FOIL: A Midterm Report. In P. Brazdil (ed.), Proceedings of the 6th European Conference on Machine Learning, Vol. 667, pp. 3–20, Springer, 1993.
4. S. Muggleton and C. Feng. Efficient induction in logic programs. In S. Muggleton (ed.), Inductive Logic Programming, pp. 281–298, Academic Press, 1992.
5. I. Stahl. Predicate Invention in Inductive Logic Programming. In L. de Raedt (ed.), Advances in Inductive Logic Programming, IOS Press, pp. 34–47, 1996.
6. A. Jorge, P. Brazdil. Architecture for Iterative Learning of Recursive Definitions. In L. de Raedt (ed.), Advances in Inductive Logic Programming, IOS Press, pp. 206–218, 1996.
7. P. Flener and S. Yilmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. Journal of Logic Programming, 41(2-3), pp. 141–195, 1999.
8. U. Schmid. Inductive Synthesis of Functional Programs – Learning Domain-Specific Control Rules and Abstract Schemes. Habilitation thesis, Technical University of Berlin, 2001.
9. T. Kohonen: Self-Organizing Maps, 3rd ed., Springer Series in Information Sciences, Vol. 30, Springer, 2001.
10. T. Villmann. Topology preservation in Self-Organizing Maps. In E. Oja and S. Kaski (eds.), Kohonen Maps, Elsevier, pp. 279–292, 1999.
11. A.F.R. Arajo and G.A. Barreto. Context in temporal sequence processing: A self-organizing approach and its application to robotics. IEEE Transactions on Neural Networks, Vol. 13, No. 1, pp. 45-57, 2002.
12. S. Kaski. Learning metrics for exploratory data analysis. In David Miller, Tulay Adali, Jan Larsen, Marc Van Hulle, and Scott Douglas (eds.), Neural Networks for Signal Processing XI, Proceedings of the 2001 IEEE Signal Processing Society Workshop, pp. 53–62. IEEE, New York, 2001.

13. A. Ultsch. Knowledge extraction from selforganizing neural networks. In O. Opitz, B. Lausen, R. Klar (eds.), Information and Classification, pp. 301-306, Springer, 1993.
14. M. Hagenbuchner, A. C. Tsoi, and A. Sperduti. A supervised self-organising map for structured data. In L. Allinson, N. Allinson, J. Slack, H. Yin (eds.), Advances in Self-Organising Maps, Springer, 2001.
15. W. Duch, R. Adamczak, and K. Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. IEEE Transactions on Neural Networks, Vol. 12., pp. 277-306, 2001.
16. A.B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting knowledge embedded within trained artificial neural networks. IEEE Transactions on Neural Networks, Vol. 9, pp. 1057-1068, 1998.
17. R. Andrews, J. Diederich, A.B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. Knowledge Based Systems, Vol. 8, pp. 373-389, 1995.
18. L. Shastri. Types and Quantifiers in SHRUTI: A Connectionist Model of Rapid Reasoning and Relational Processing. In S. Wermter and R. Sun (eds.), Hybrid Neural Symbolic Integration Lecture Notes in Artificial Intelligence, Springer, pp. 28–45, 2000.
19. R. Bogacz and C. Giraud-Carrier. BRAINN: A Connectionist Approach to Symbolic Reasoning. In Proceedings of the First International ICSC/IFAC Symposium on Neural Computation (NC'98), ICSC Academic Press, 1998.
20. S. Hölldobler, Y. Kalinke, and J. Wunderlich. A Recursive Neural Network for Reflexive Reasoning. In S. Wermter and R. Sun (eds.), Hybrid Neural Symbolic Integration Lecture Notes in Artificial Intelligence, Springer, pp. 46–62, 2000.
21. A.S. Sato, K. Yamada. Generalized learning vector quantization. In G. Tesauro, D. Touretzky, and T. Leen (eds.), Advances in Neural Information Processing Systems, Vol. 7, pp. 423–429, MIT Press, 1995.
22. B. Hammer and T. Villmann. Estimating relevant input dimensions for selforganizing algorithms. In N. Allison, H. Yin, L. Allinson, J. Slack (eds.), Advances in SelfOrganizing Maps, pp. 173-180, Springer, 2001.
23. B. Hammer, M. Strickert, and T. Villmann. Learning Vector Quantization for Multimodal Data. Paper submitted to ICANN 2002.
24. B. Hammer, A. Rechtien, and M. Strickert. Rule Extraction from Self-Organizing Networks Paper submitted to ICANN 2002.
25. S. S. Kwek. Geometric Concept Learning and Related Topics. Dissertation thesis, University of Illinois, UIUCDCS-R-96-1980, 1996.
26. T. Martinetz, S. Berkovich, and K. Schulten. 'Neural-gas' network for vector quantization and its application to time-series prediction. IEEE TNN 4:(4), pp. 558–569, 1993.
27. C.L. Blake and C.J. Merz. UCI Repository of machine learning databases. http://www.ics.uci.edu/˜mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science, 1998.
28. W. Daelemans, J. Zavrel, van der Sloot, and van den Bosch. TiMBL: Tilburg MemoryBased Learner version 4.0 Reference Guide. ILK Technical Report – ILK 0104, http://ilk.kub.nl, 2001.
29. W. Schiffmann, M. Joost and R. Werner. Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons, 1992.