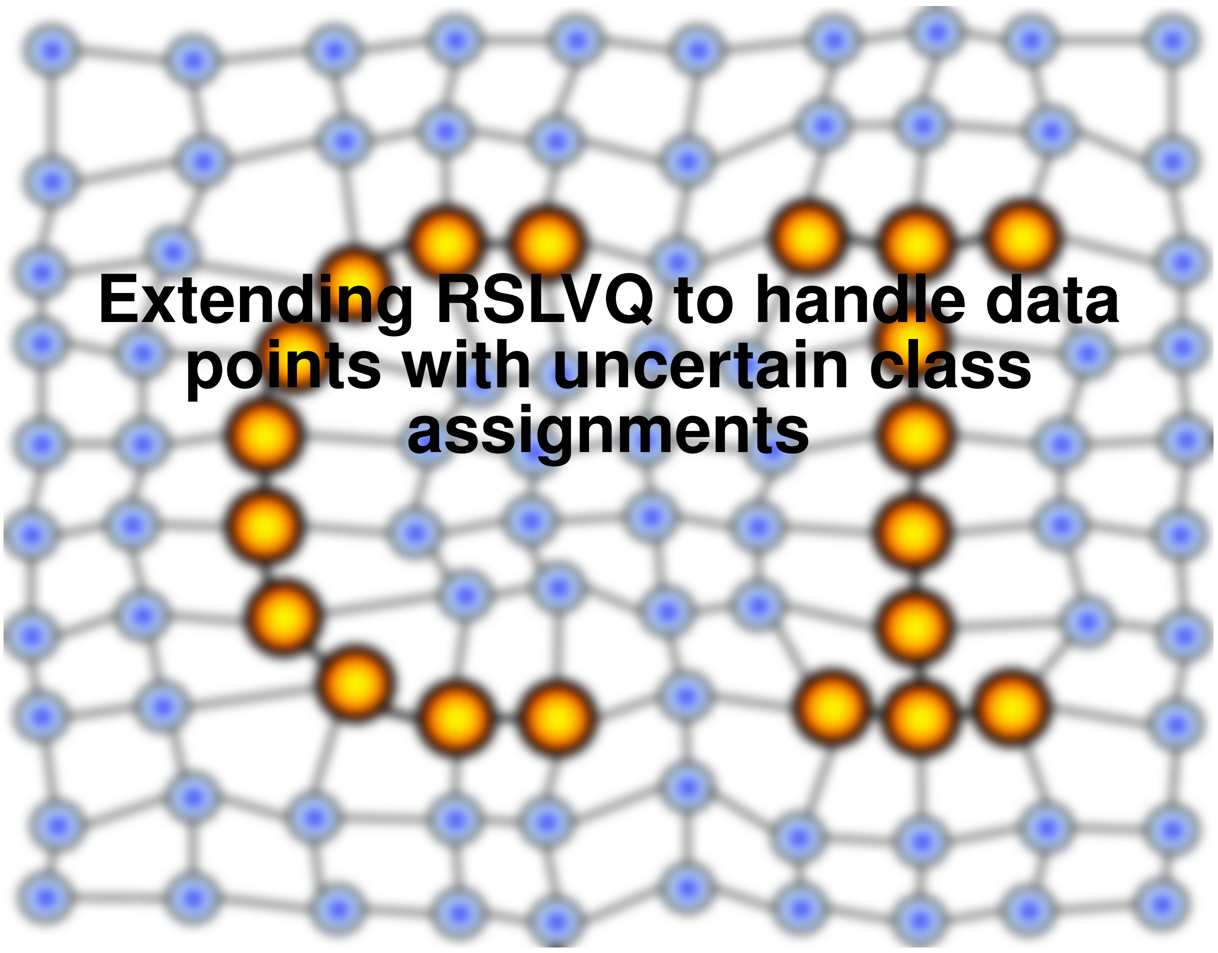
MACHINE LEARNING REPORTS



Extending RSLVQ to handle data points with uncertain class assignments

Report 02/2009
Submitted: 19.11.2009
Published: 22.01.2010

Tina Geweniger[1], Petra Schneider[2],
Frank-Michael Schleif[1],Michael Biehl[2], Thomas Villmann[3]
[1]University of Leipzig - Group of Computational Intelligence Semmelweisstrasse 10, 04103 Leipzig, Germany; [2]Institute of Mathematics and Computing Science, University of Groningen P.O.Box 407, 9700 AK Groningen, The Netherlands; [3]University of Applied Sciences Mittweida Department of MPI, Computational Intelligence Group Technikumplatz 17, 09648 Mittweida, Germany

Abstract

Learning Vector Quantization has been introduced by Kohonen for classification. An advanced method derived thereof is the Robust Soft Learning Vector Quantization developed by Seo&Obermayer bearing on parametrized probability estimates for class and data distributions. We extended this method to handle data with fuzzy class assignments. Further, a parameter optimization scheme is considered for improvement of the classifcation accuracy. The performance of new algorithm is demonstrated on two examples: a two-dimensional toy problem and a real life data set. The outcomes are compared with the known results using Fuzzy Cohen Kappa.

1 Introduction

The general Learning Vector Quantization (LVQ) belongs to the category of supervised prototype based vector quantizers for classification [3]. This method is intuitive and known for its stability solving a wide range of classification problems. In this framework, each class is represented by one ore more prototypes. Usually, the prototype training is based on Hebbian Learning, which provides a paradigm to obtain fast and easy-to-use algorithms. There exist a variety of sub-types of the basic LVQ algorithm. Standard LVQ is based on heuristics aiming on the reduction of classification errors [3]. Advanced LVQ schemes like Generalized LVQ (GLVQ) [5] or Robust Soft Learning Vector Quantization (RSLVQ, [7]) replace the simple classification error by sophisticated cost functions which allow a gradient ascent/descent learning or EM-optimization.

Generally, for all the approaches, the placement of the prototypes within a class depends on their distance to the respective data points. Given labeled training data $\boldsymbol{\xi} \in \mathbb{R}^N$ and an appropriate distance measure $d(\boldsymbol{\xi}, \boldsymbol{w})$ in $\mathbb{R}^N$ the class prototypes $\boldsymbol{w} \in \mathbb{R}^N$ are placed according to the class distributions, which itself are determined by the underlying metric. A common metric for the calculation of the similarity between prototypes and data points is the Euclidean distance $d(\boldsymbol{\xi}, \boldsymbol{w}) = (\boldsymbol{\xi} - \boldsymbol{w})^T(\boldsymbol{\xi} - \boldsymbol{w})$. But other distance measures might as well be suitable depending on the specific classification problem.

All of the different LVQ schemes have in common, that they work on hard labeled data only. Each data point for the training is known to belong to exactly one class. In practice, this might not be a realistic assumption. E.g. in medicine a patients disease might not be uniquely classifiable to only one diagnosis. The medical doctor implicitly makes probability assumptions about the true kind of illness. Therefore, learning algorithms for prototype classifiers handling uncertain class assignments during training are required. RSLVQ is a gradient based method maximizing the likelihood ratio [7]. It is a robust scheme but so far only applicable to crisp labeled training data. In this paper we extend this approach for handling fuzzy labeled training data. For this purpose a vectorial adaption scheme is proposed.

Example applications for toy and real world data show the abilities of the method. In parallel some essential properties of the training behavior differing from crisp learning are demonstrated.

2 Robust Soft Learning Vector Quantization

The RSLVQ algorithm, an advanced LVQ, was introduced by Seo and Obermayer [7] and is also a Nearest Prototype Classifier (NPC). Yet, on the contrary to the original LVQ, RSLVQ is based on a statistical model. Here it is assumed that the probability density $p(\boldsymbol{\xi})$ of the data points $\boldsymbol{\xi} \in \mathbb{R}^N$ with N being the dimensionality of the data points can be described by a mixture model. Every component j of the mixture is assumed to generate data which belongs to only one of the C classes. The classification itself is based on a winner takes all scheme.

The probability density of all the data points is given by

$$p(\boldsymbol{\xi}|W) = \sum_{i=1}^{C} \sum_{j:c(\boldsymbol{w}_j)=i}^{m} p(\boldsymbol{\xi}|j)P(j) \tag{1}$$

where $W = \{(\boldsymbol{w}_j, c(\boldsymbol{w}_j))\}_{j=1}^{m}$ is the set of m labeled prototype vectors $\boldsymbol{w}_j \in \mathbb{R}^N$ and their assigned class labels $c(\boldsymbol{w}_j)$. $P(j)$ stands for the probability that data points are generated by component j of the mixture and is commonly set to an identical value for all the prototypes. The conditional density $p(\boldsymbol{\xi}|j)$, which describes the probability that component j is generating a particular data point $\boldsymbol{\xi}$, is a function of the prototype $\boldsymbol{w}_j$ itself. $p(\boldsymbol{\xi}|j)$ can be chosen to have the normalized exponential form $p(\boldsymbol{\xi}|j) = K(j) \cdot \exp f(\boldsymbol{\xi}, \boldsymbol{w}_j, \sigma_j^2)$ where $K(j)$ is the normalization constant and the hyper parameter σ_j^2 the width of component j.

The aim of RSLVQ is to place the prototypes such that a given data set is classified as accurately as possible. Therefore the likelihood ratio

$$L = \prod_{i=1}^{l} L(\boldsymbol{\xi}_i, y_i), \qquad with \;\; L(\boldsymbol{\xi}_i, y_i) = \frac{p(\boldsymbol{\xi}_i, y_i|W)}{p(\boldsymbol{\xi}_i|W)} \tag{2}$$

where l is the number of data points has to be maximized. The ratio is built up of the particular probability density $p(\boldsymbol{\xi}_i, y_i|W)$ that data point $\boldsymbol{\xi}_i$ is generated by a mixture component of the correct class y_i

$$p(\boldsymbol{\xi}_i, y_i|W) = \sum_{j:c(\boldsymbol{w}_j)=y_i} p(\boldsymbol{\xi}_i|j)P(j) \tag{3}$$

and the total probability density $p(\boldsymbol{\xi}_i, y_i|W)$

$$p(\boldsymbol{\xi}_i|W) = \sum_{j} p(\boldsymbol{\xi}_i|j)P(j). \tag{4}$$

The derivation of the learning rules is obtained by a stochastic gradient ascent of the cost function and can be found in detail in [6].

In general the cost function to maximize can be stated as

$$E_{RSLVQ} = \sum_{i=1}^{l} \log(\frac{p(\boldsymbol{\xi}_i, y_i|W)}{p(\boldsymbol{\xi}_i|W)}). \tag{5}$$

3 Fuzzy RSLVQ

We now introduce fuzzy class labels for the data points, which implies a level of uncertainty in the data set itself. There are various applications where there is only a diffuse classification possible or the training data can only be obtained by insecure methods.

3.1 Introducing fuzzy class labels

The assumption of fuzzy labeled data points requires an adaption of the original RSLVQ algorithm. The originally crisp class label y_i for the training data point $\boldsymbol{\xi}_i$ becomes a C-dimensional vector $\boldsymbol{y}_i^k$ of assignment probabilities with $\sum_{k=1}^{C} y_i^k = 1$ and $y_i^k \in [0, 1]$. As before, each prototype $\boldsymbol{w}$ describes exactly one class with $c(\boldsymbol{w}) \in [1, C]$ and the classification of untrained data is based on the winner takes all scheme. Taking the fuzzy class assignments of the data points into account equation (3) changes to

$$p(\boldsymbol{\xi}_i, \boldsymbol{y}_i|W) = \sum_{k=1}^{C} \boldsymbol{y}_i^k \sum_{j:c(\boldsymbol{w}_j)=k}^{m} p(\boldsymbol{\xi}_i|j)P(j). \tag{6}$$

The total probability density $p(\boldsymbol{\xi}_i, \boldsymbol{y}_i|W)$ (4) does not change.
Therefore, the cost function of Fuzzy RSLVQ reads as

$$E_{FRSLVQ} = \sum_{i=1}^{l} \log\left(\frac{p(\boldsymbol{\xi}_i, \boldsymbol{y}_i|W)}{p(\boldsymbol{\xi}_i|W)}\right). \tag{7}$$

3.2 Derivation of learning rules

In order to optimize the classification the cost function has to be maximized, which can be done by a stochastic gradient ascent with respect to the parameter to update. In the appendix we give a detailed description of the derivation process leading to the following general update rule

$$\frac{\partial \log \frac{p(\boldsymbol{\xi}, \boldsymbol{y}|W)}{p(\boldsymbol{\xi}|W)}}{\partial \Theta_j} = \big(P_{\boldsymbol{y}}(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi})\big)\left(\frac{1}{K(j)}\frac{\partial K(j)}{\partial \Theta_j} + \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_j, \sigma_j^2))}{\partial \Theta_j}\right) \tag{8}$$

Here we assumed a general parameter $\Theta \neq \boldsymbol{\xi}$ to be updated. By replacing this parameter with the parameters of interest the appropriate learning rules can be obtained easily.
The terms $P_{\boldsymbol{y}}(j|\boldsymbol{\xi})$ and $P(j|\boldsymbol{\xi})$ are assignment probabilities. $P_{\boldsymbol{y}}(j|\boldsymbol{\xi})$ is the assignment probability of $\boldsymbol{\xi}$ to component j within class $c(\boldsymbol{w}_j)$. $P(j|\boldsymbol{\xi})$ is the assignment probability of $\boldsymbol{\xi}$ to component j independent of the class membership.

$$P_{\boldsymbol{y}}(i|\boldsymbol{\xi}) = \frac{y^{c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}, \boldsymbol{y}|W)}$$

$$P(i|\boldsymbol{\xi}) = \frac{P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}|W)}$$

Further we concentrate on the update of the prototypes $\boldsymbol{w}$ and the hyper parameter σ^2 assuming the special case of a Gaussian mixture model with $P(j) = 1/m$ for all j. The normalization constant $K(j)$ can now be set to $K(j) = (2\pi\sigma_j^2)^{(-N/2)}$.

3.2.1 Updating the prototypes w

To derive the update rule for the prototypes we replaced the general parameter Θ_j in equation (8) with the prototype $\boldsymbol{w}_j$. $K(j)$ is independent of $\boldsymbol{w}_j$, therefore the partial derivate $\partial K(j)/\partial \boldsymbol{w}_j$ evaluates to zero. Since we assume a Gaussian mixture model we use the similarity function $f(\boldsymbol{\xi}, \boldsymbol{w}, \sigma^2) = \frac{-d(\boldsymbol{\xi}, \boldsymbol{w})}{2\sigma^2}$ with $d(\boldsymbol{\xi}, \boldsymbol{w})$ being the distance between data point $\boldsymbol{\xi}$ and prototype $\boldsymbol{w}$. This leads to

$$\frac{\partial \log \frac{p(\boldsymbol{\xi}, \boldsymbol{y}|W)}{p(\boldsymbol{\xi}|W)}}{\partial \boldsymbol{w}_j} = \big(P_{\boldsymbol{y}}(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi})\big) \left(-\frac{1}{2\sigma^2} \frac{\partial d(\boldsymbol{\xi}, \boldsymbol{w}_j)}{\partial \boldsymbol{w}_j} \right) \tag{9}$$

The original RSLVQ algorithm was based on the squared Euclidean distance $d(\boldsymbol{\xi}, \boldsymbol{w}) = (\boldsymbol{\xi} - \boldsymbol{w})^T (\boldsymbol{\xi} - \boldsymbol{w})$ which yields the update rule

$$\Delta \boldsymbol{w}_j = \frac{\alpha_1}{\sigma^2} \big(P_{\boldsymbol{y}}(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi})\big) (\boldsymbol{\xi} - \boldsymbol{w}_j) \tag{10}$$

for each prototype with $\alpha_1 > 0$ as learning rate.

3.2.2 Updating the hyper parameter σ^2

The partial derivatives of $K(j)$ and $f(\boldsymbol{\xi}, \boldsymbol{w}_j, \sigma_j^2)$ with respect to the hyper parameter σ^2 are

$$\begin{aligned} \frac{\partial K(j)}{\partial \sigma_j^2} &= -\frac{N}{2} \frac{1}{(2\pi\sigma_j^2)^{N/2} \, \sigma_j^2} \\ \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_j, \sigma_j^2)}{\partial \sigma_j^2} &= \frac{d(\boldsymbol{\xi}, \boldsymbol{w}_j)}{2\, \sigma_j^4} \end{aligned}$$

in combination with equation (8) we obtain the learning rule

$$\Delta \sigma_j^2 = \alpha_2 \left(P_{\boldsymbol{y}}(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi})\right) \left(\frac{d(\boldsymbol{\xi}, \boldsymbol{w}_j)}{\sigma_j^4} - \frac{N}{\sigma_j^2} \right), \tag{11}$$

where α_2 is the learning rate for the hyper parameter and $d(\boldsymbol{\xi}, \boldsymbol{w}_j)$ most commonly the Euclidean Distance between the data point $\boldsymbol{\xi}$ and prototype $\boldsymbol{w}_j$.

For the more general case of the global parameter $\sigma_j^2 = \sigma^2$ being identical for all components j, the hyper parameter can be updated by the summation of the probability assignments

$$\Delta\sigma^2 = \alpha_2 \sum_{j=1}^{m} \left(P_{\boldsymbol{y}}(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi})\right) \cdot \frac{d(\boldsymbol{\xi}, \boldsymbol{w}_j)}{\sigma^4}. \tag{12}$$

3.3 Artificial Data

In a first set of experiments, we apply Fuzzy RSLVQ to artificial toy data sets. The data sets consist of two spherical Gaussian clusters of equal variance in a two-dimensional space. We set the distribution's mean values to $\mu_1 = [-1, 0]$ and $\mu_2 = [1, 0]$ and choose four different settings for the variance $\varphi^2 \in \{0.3, 0.5, 0.7, 1.0\}$. Each cluster consists of 1000 samples. We define the class memberships $y^{1,2}$ of sample $\boldsymbol{\xi}$ depending on the first component $\xi(1)$ according to the linear relationship specified in Tab. 1, see also Fig. 1. The experiments are split into three parts: At first, we learn one prototype per class with constant σ^2. Next, only the hyper parameter is optimized with the prototypes being fixed in the cluster centers. Finally, hyper parameter and prototypes are learned simultaneously. The findings are compared to identical experiments with RSLVQ.

For our analysis, we use the learning parameter settings $\alpha_1 = 1 \cdot 10^{-3}$, $\alpha_2 = 5 \cdot 10^{-5} \cdot \sigma^2(0)$. As fixed and initial values of the hyper parameter we set $\sigma^2(0) \in \{0.05, 0.15, 0.3\}$. The prototypes are initialized close to the cluster means, and training is continued for 500 epochs. We perform each experiment on ten independent data sets.

During FRSLVQ training with constant σ^2, the prototypes move away from each other; they move along the first axis away from the cluster centers. The final distance $\|\boldsymbol{w}_1 - \boldsymbol{w}_2\|$ depends on the value of the hyper parameter and the cluster's variance. Namely, the distance increases with increasing softness and increasing variance φ^2; though, the influence of φ^2 is comparably weak. These observations are depicted in Fig. 2. On the contrary, we observe the opposite effect during RSLVQ training, i.e., the prototypes move in the direction of the decision boundary; the distance between $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ decreases during training. The prototypes saturate closer to the decision boundary, the larger φ^2 and the smaller σ^2 (see Fig. 3).

The results of hyper parameter learning with fixed prototypes are visualized in Fig. 4. FRSLVQ converges to values very close to zero after only a small number of training epochs. On the other hand, RSLVQ approaches the cluster's variance. These observations hold for both algorithms independent of the initialization $\sigma^2(0)$.

Concerning FRSLVQ, the simultaneous training of prototypes and hyper parameter initially shows the same behavior as described above: the prototypes move away

Table 1: Relation between $\xi(1)$ and the assignment probabilities of sample $\boldsymbol{\xi}$ to the two classes.

	$\xi(1) \leq -1$	$-1 < \xi(1) < 1$	$1 \leq \xi(1)$
y^1	1	$-0.5 \cdot \xi(1) + 0.5$	0
y^2	0	$0.5 \cdot \xi(1) - 0.5$	1

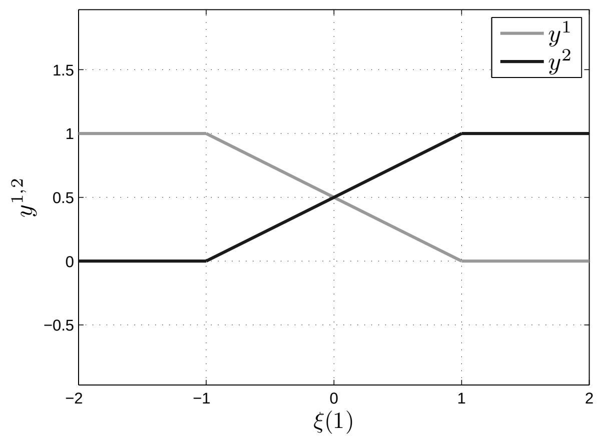


Figure 1: Artificial data. Visualization of the relation between the first component of sample $\boldsymbol{\xi}$ and the sample's assignment probabilities to the two classes. The value $\xi(2)$ is irrelevant for the labeling.

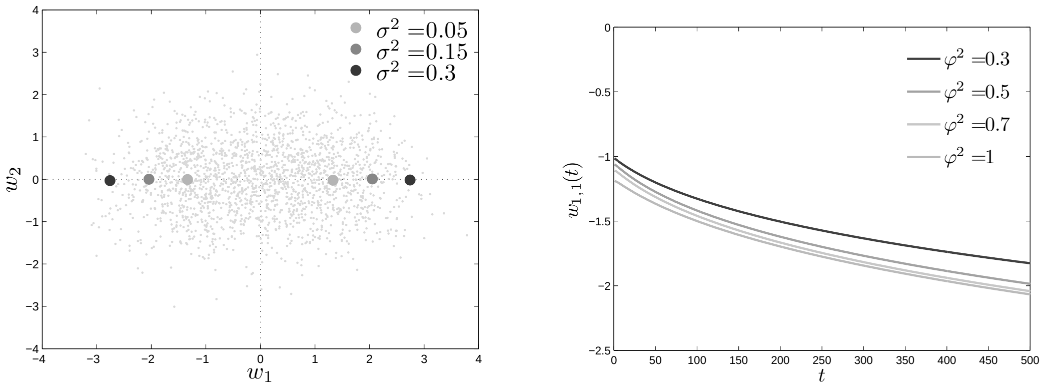


Figure 2: Artificial data. Left: Mean final prototype locations after FRSLVQ training with varying softness on data sets with $\varphi^2 = 0.3$. Right: Trajectories of first component of class one prototype during FRSLVQ training on datasets with different cluster variance and equal hyper parameter $\sigma^2 = 0.15$.

from each other and the hyper parameter quickly converges to zero. However, the prototypes' movement is stopped, when σ^2 reaches very small values. Hence, the distance between the prototypes does not increase that extensively as observed in the experiments with constant σ^2.

The hyper parameter training in RSLVQ weakens the movement of prototypes towards to decision boundary; σ^2 reaches smaller values compared to the experiments with fixed prototypes.

3.4 Real Life Data

In our second set of experiments, the algorithms are applied to the real life data set used in [1],[8] . The data is based on serial transverse sections of barley grains at different developmental stages. The classification task consists in the identification of

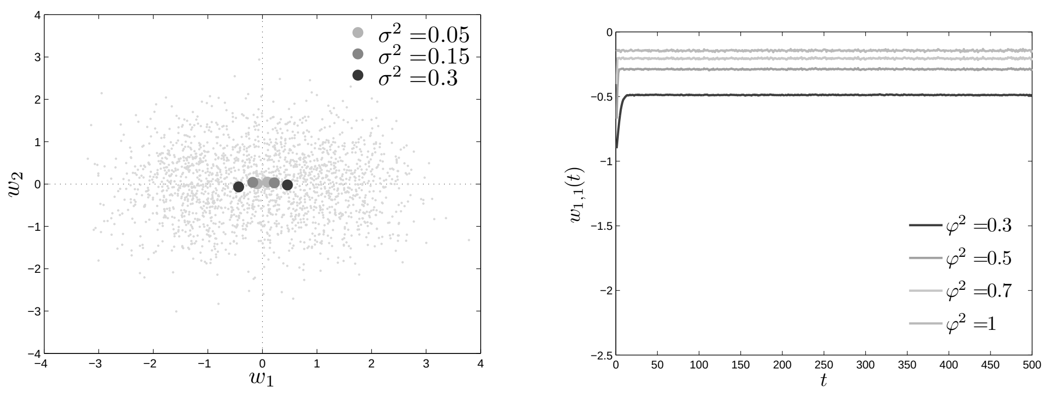


Figure 3: Artificial data. Left: Mean final prototype locations after RSLVQ training with varying on data sets with $\varphi^2 = 0.3$. Right: Trajectories of first component of class one prototype during RSLVQ training on datasets with different cluster variances and equal hyper parameter $\sigma^2 = 0.15$.

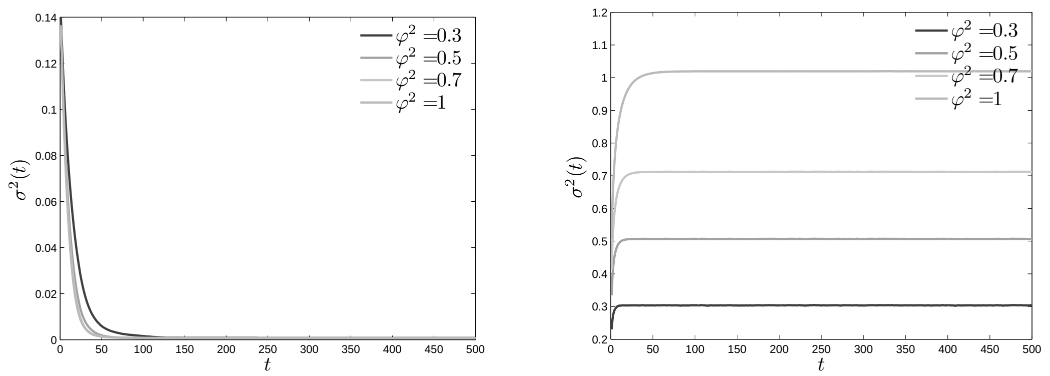


Figure 4: Artificial data. Mean evolution of the hyper parameter during training on datasets with different cluster variances φ^2 and constant prototypes fixed in the cluster centers. The hyper parameter was always initialized with $\sigma^2(0) = 0.15$. The plots are representative for all tested $\sigma^2(0)$. Left: FRSLVQ-Training. Right: RSLVQ-Training.

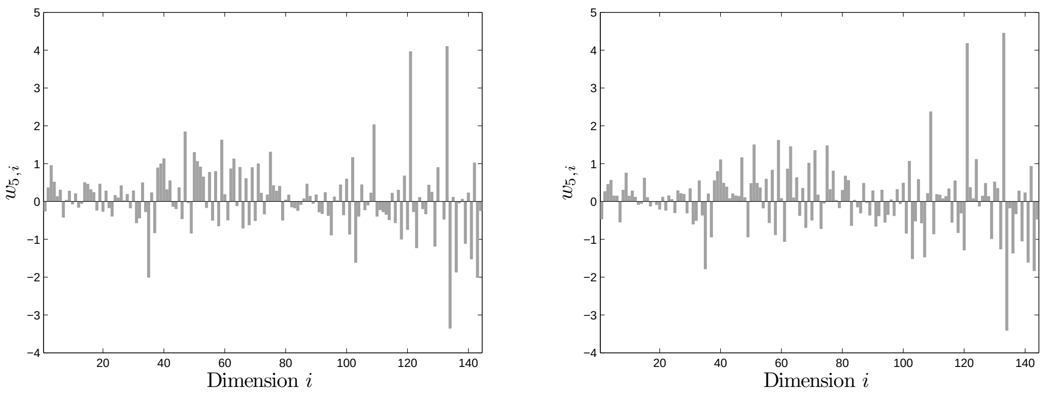


Figure 5: Grain data set. Visualization of the class 5 prototypes obtained by FRSLVQ-Training (left) and RSLVQ-Training (right) in one training run.

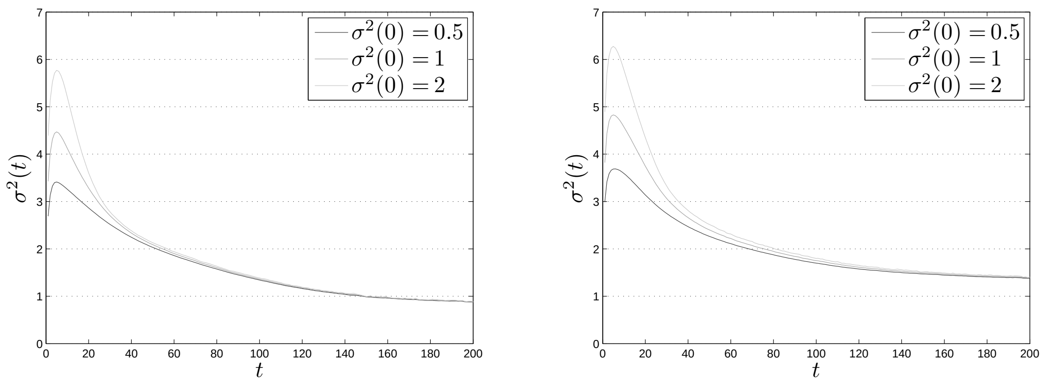


Figure 6: Grain data set. Mean evolution of the hyper parameter during training with different initial settings $\sigma^2(0)$. Left: FRSLVQ-Training. Right: RSLVQ-Training.

11 different tissue types. Many sections cannot be classified distinctively, this holds especially for borders between tissues. For this reason, fuzzy class assignments are provided beside the crisp labeling. The samples are described by means of 144 features and 4418 data points are available. The data set is randomly split into 3800 samples for training, while the remaining data is used for testing purposes.

In order to evaluate the classification accuracy, we compute Fuzzy Cohen's Kappa κ as introduced in [2]. This coefficient always lies in the interval [-1 1] and measures the agreement of two classifiers. The degree of classification agreement reaches from *slight* agreement with $0 < \kappa <= 0.2$ over *fair*, *moderate*, and *substantial* up to *perfect* agreement with $0.8 < \kappa <= 1.0$. Values beneath zero indicate a *poor* or *accidential* agreement only (see [4] for details). We train FRSLVQ and compare the results to RSLVQ with the crisp labeling. In all experiments, one prototype per class and a global hyper parameter are adapted to the data. We apply $\alpha_1 = 0.01$, $\alpha_2 = 5 \cdot 10^{-4} \cdot \sigma^2(0)$, $\sigma^2(0) \in \{0.5, 1.0, 2.0\}$ and train the system for 200 epochs. To verify the results, the experiments are repeated on five independent constellations of training- and test set.

The two algorithms identify nearly the same prototypes (see Fig. 5 for examples). The same holds for the learning process of the hyper parameter, we do not observe such drastic differences as in the previous experiments. The evolution of σ^2 in the

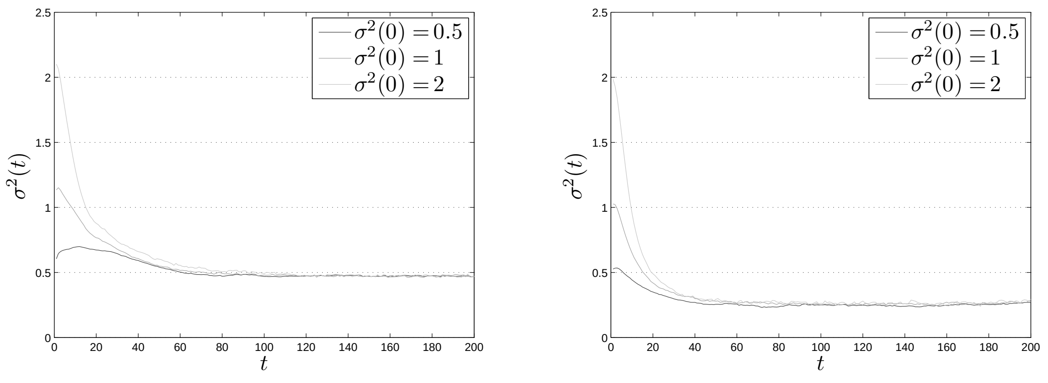


Figure 7: Grain data set. Mean evolution of the hyper parameter during FRSLVQ training with different initial settings $\sigma^2(0)$. The fuzziness of the original data set was increased by adding random noise to the class labels. Left: Uniform noise with variance 0.1 added. Right: Uniform noise with variance 0.5 added.

course of FRSLVQ- and RSLVQ-Training for the different initializations $\sigma^2(0)$ is depicted in Fig. 6. The curves yielded by the alternative algorithms resemble to large extent: σ^2 increases slightly at the beginning of training, but starts degrading after $\approx$ 10 sweeps through the training set; finally, the hyper parameter always converges to the same value, independent of $\sigma^2(0)$. Note however, that the final value $\sigma^2(t)$ is slightly smaller after FRSLVQ training; we observe $\sigma^2_{FRSLVQ}(t) \approx 0.9$ and $\sigma^2_{RSLVQ}(t) \approx 1.3$. Obviously, the uncertainty in the class memberships induces smaller optimal values σ^2. To verify this assumption, we artificially increase the fuzziness of the class labels: uniform noise of different variance is added to the label vectors, followed by a normalization step to guarantee $\sum_i y^i = 1$. We add noise of variance 0.1, 0.3 and 0.5 and repeat the FRSLVQ training process with identical learning parameters. As depicted in Fig. 7, $\sigma^2(t)$ approaches smaller values with increasing noise level.

The evolution of the coefficient κ during FRSLVQ-Training on the original data set is depicted in Fig. 8. The coefficient reaches $\kappa_{train} \approx 0.83$ which corresponds to perfect agreement; the final value on the test data $\kappa_{test} = 0.78$ implies substantial agreement (Fig. 8, left). The additional noise in the class labeling clearly reduces the algorithm's performance. With the lowest noise level we applied in our testings, both values κ_{train} and κ_{test} degrade to only moderate agreement (Fig. 8, right).

4 Conclusion

We extended the known RSLVQ algorithm to work with uncertain class labels and called this new variant Fuzzy RSLVQ (FRSLVQ). Therefore we substituted the crisp class assignment of each prototype by a possibilistic vector reflecting the relative class assignments. We derived the update rules for the prototypes and the hyper parameters respectively. In extensive experiments we analyzed the behavior of the learning process and obtained interesting results which we compared with the characteristics of RSLVQ.

- Using RSLVQ the prototypes converge towards the decision boundary. For FRSLVQ we observed that the prototypes tend to move away from each other into a region

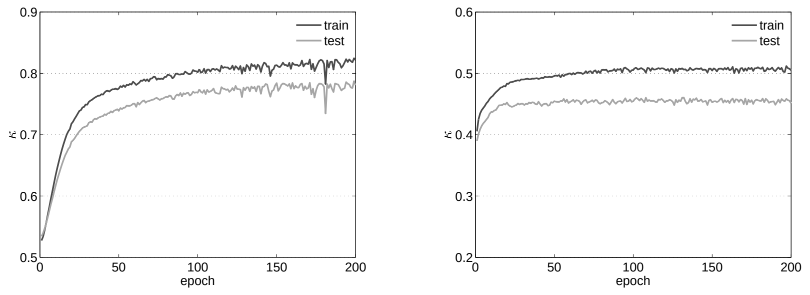


Figure 8: Grain data set. Mean evolution of coefficient κ on training and test data during FRSLVQ-Training with adaptive softness and $\sigma^2(0) = 1$. The plot is representative for all initializations $\sigma^2(0)$. Left: Training on original data set. Right: Training on data set with increased fuzziness.

of higher classification accuracy. This is due to the fact that the contribution of the data points to the prototype update for a specific class depends on their strength for describing this specific class. The higher the classification accuracy of the data points the higher their attraction for the prototypes.

- The hyper parameter is very crucial for the classification accuracy and the efficiency of the runs. Therefore we considered to adapt this parameter in the course of the training. We observed that the initial choice of σ^2 has no influence on the final value. Yet contrary to RSLVQ, where the hyper parameter approaches the cluster's variance, with FRSLVQ this parameter converges to rather small or even close to zero values during the learning process depending on the degree of uncertainty within the data set. The prototype update stops when the hyper parameter reaches very small values.

Using real life data FRSLVQ shows the same behavior as with the artificial dataset with the minor difference, that the results are not as strong pronounced. The hyper parameter converges to small values but does not reach zero. The protoypes found by the two algorithms are nearly identical, which is due to the fact that the real life dataset is a mixture of fuzzy and crisp labeled data points.

We also compared the two algorithms in terms of Fuzzy Cohen Kappa and found a perfect agreement of the classification accuracy for the training data and a substantial agreement for the test data.

5 Appendix

The learning rules for RSLVQ using fuzzy class labels are deduced in the style of the derivation approach given in [6]. There the derivative of the likelihood ratio with respect to a general parameter $\Theta_i \neq \boldsymbol{\xi}$ was deduced. The parameter Θ_i can later on be substituted by the prototype $\boldsymbol{w}_i$, the hyper parameter σ_i^2 or the metric parameter λ_i.

For our further considerations we chose the conditional density to have the normalized exponential form $p(\boldsymbol{\xi}|i) = K(i)\mathtt{exp}f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)$ where $K(i)$ is the normalization factor. This factor depends on the shape of component i. Assuming a N-dimensional Gaussian distribution $K(i)$ can be set to $K(i) = 2\pi\sigma_i^{2(-N/2)}$.

$$
\begin{aligned}
\frac{\partial \log \frac{p(\boldsymbol{\xi},\boldsymbol{y}|W)}{p(\boldsymbol{\xi}|W)}}{\partial \Theta_i} &= \frac{\partial \log p(\boldsymbol{\xi},\boldsymbol{y}|W)}{\partial \Theta_i} - \frac{\partial \log p(\boldsymbol{\xi}|W)}{\partial \Theta_i} \\
&= \frac{1}{p(\boldsymbol{\xi},\boldsymbol{y}|W)} \underbrace{\frac{\partial p(\boldsymbol{\xi},\boldsymbol{y}|W)}{\partial \Theta_i}}_{(a)} - \frac{1}{p(\boldsymbol{\xi}|W)} \underbrace{\frac{\partial p(\boldsymbol{\xi}|W)}{\partial \Theta_i}}_{(b)} \\
&= \frac{\boldsymbol{y}^{c(\boldsymbol{w}_i)} P(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi},\boldsymbol{y}|W)} \left(\frac{\partial K(i)}{\partial \Theta_i} + K(i) \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i} \right) \\
&\quad - \frac{P(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi}|W)} \left(\frac{\partial K(i)}{\partial \Theta_i} + K(i) \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i} \right) \\
&= y^{c(\boldsymbol{w}_i)} \left(\frac{P(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi},\boldsymbol{y}|W)} \frac{\partial K(i)}{\partial \Theta_i} + \frac{P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi},\boldsymbol{y}|W)} \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i} \right) \\
&\quad - \left(\frac{P(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi}|W)} \frac{\partial K(i)}{\partial \Theta_i} + \frac{P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi}|W)} \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i} \right)
\end{aligned}
$$

(continued on next page)

$$
\begin{aligned}
&= \frac{1}{K(i)} \underbrace{\frac{y^{c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}, \boldsymbol{y}|W)}}_{P_{\boldsymbol{y}}(i|\boldsymbol{\xi})} \frac{\partial K(i)}{\partial \Theta_i} \\
&\quad + \underbrace{\frac{y^{c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}, \boldsymbol{y}|W)}}_{P_{\boldsymbol{y}}(i|\boldsymbol{\xi})} \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}{\partial \Theta_i} \\
&\quad - \frac{1}{K(i)} \underbrace{\frac{P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}|W)}}_{P(i|\boldsymbol{\xi})} \frac{\partial K(i)}{\partial \Theta_i} \\
&\quad - \underbrace{\frac{P(i) K(i) e^{f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}}{p(\boldsymbol{\xi}|W)}}_{P(i|\boldsymbol{\xi})} \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}{\partial \Theta_i} \\
&= P_{\boldsymbol{y}}(i|\boldsymbol{\xi}) \left(\frac{1}{K(i)} \frac{\partial K(i)}{\partial \Theta_i} + \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}{\partial \Theta_i} \right) \\
&\quad - P(i|\boldsymbol{\xi}) \left(\frac{1}{K(i)} \frac{\partial K(i)}{\partial \Theta_i} + \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}{\partial \Theta_i} \right) \\
&= \big(P_{\boldsymbol{y}}(i|\boldsymbol{\xi}) - P(i|\boldsymbol{\xi}) \big) \left(\frac{1}{K(i)} \frac{\partial K(i)}{\partial \Theta_i} + \frac{\partial f(\boldsymbol{\xi}, \boldsymbol{w}_i, \sigma_i^2, \lambda_i)}{\partial \Theta_i} \right)
\end{aligned}
$$

The terms $P_{\boldsymbol{y}}(i|\boldsymbol{\xi})$ and $P(i|\boldsymbol{\xi})$ are assignment probabilities

$$P_{\boldsymbol{y}}(i|\boldsymbol{\xi}) = \frac{\sum_{k=1}^{C} y^k \delta_{k,c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{\sum_{k=1}^{C} y^k \sum_{j=1}^{m} \delta_{k,c(\boldsymbol{w}_j)} P(j) K(j) e^{f(\boldsymbol{\xi},\boldsymbol{w}_j,\sigma_j^2,\lambda_j)}}$$

$$= \frac{y^{c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{\sum_{k=1}^{C} y^k \sum_{j=1}^{m} \delta_{k,c(\boldsymbol{w}_j)} P(j) K(j) e^{f(\boldsymbol{\xi},\boldsymbol{w}_j,\sigma_j^2,\lambda_j)}}$$

$$= \frac{y^{c(\boldsymbol{w}_i)} P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi},\boldsymbol{y}|W)}$$

$$P(i|\boldsymbol{\xi}) = \frac{P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{\sum_{j=1}^{m} P(j) K(j) e^{f(\boldsymbol{\xi},\boldsymbol{w}_j,\sigma_j^2,\lambda_j)}}$$

$$= \frac{P(i) K(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}}{p(\boldsymbol{\xi}|W)}$$

$P_{\boldsymbol{y}}(i|\boldsymbol{\xi})$ is the assignment probability to component i within class $c(\boldsymbol{w}_i)$. $P(i|\boldsymbol{\xi})$ is the assignment probability to component i independent of the class membership.

(a)

$$\begin{aligned}
\frac{\partial p(\boldsymbol{\xi},\boldsymbol{y}|W)}{\partial \Theta_i} &= \frac{\partial}{\partial \Theta_i}\left(\sum_{k=1}^{C} \boldsymbol{y}^k \sum_{j=1}^{m} \delta_{k,c(\boldsymbol{w}_j)} p(\boldsymbol{\xi}|j) P(j)\right) \\
&= \sum_{k=1}^{C} \boldsymbol{y}^k \sum_{j=1}^{m} \delta_{k,c(\boldsymbol{w}_j)} P(j) \frac{\partial p(\boldsymbol{\xi}|j)}{\partial \Theta_i} \\
&= \sum_{k=1}^{C} \boldsymbol{y}^k \sum_{j=1}^{m} \delta_{k,c(\boldsymbol{w}_j)} P(j) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)} \left(\frac{\partial K(j)}{\partial \Theta_i} + K(j) \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i}\right) \\
&= \boldsymbol{y}^{c(\boldsymbol{w}_i)} P(i) e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)} \left(\frac{\partial K(i)}{\partial \Theta_i} + K(i) \frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i}\right)
\end{aligned}$$

(b)

$$\begin{aligned}
\frac{\partial p(\boldsymbol{\xi}|W)}{\partial \Theta_i} &= \frac{\partial \sum_{j=1}^{m} p(\boldsymbol{\xi}|j)P(j)}{\partial \Theta_i} \\
&= P(i)\frac{\partial p(\boldsymbol{\xi}|i)}{\partial \Theta_i} \\
&= P(i)e^{f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}\left(\frac{\partial K(i)}{\partial \Theta_i} + K(i)\frac{\partial f(\boldsymbol{\xi},\boldsymbol{w}_i,\sigma_i^2,\lambda_i)}{\partial \Theta_i}\right)
\end{aligned}$$

References

[1] C. Brüß, F. Bollenbeck, F.-M. Schleif, W. Weschke, T. Villmann, and U. Seiffert. Fuzzy image segmentation with fuzzy labeled neural gas. In M. Verleysen, editor, *Proc. Of European Symposium on Artificial Neural Networks (ESANN'2006)*, pages 563–568, Brussels, Belgium, 2006. d-side publications.

[2] W. Dou, Y. Ren, Q. Wu, S. Ruan, Y. Chen, and D. B. A.-M. Constans. Fuzzy kappa for the agreement measure of fuzzy classifications. *Neurocomputing*, 70:726–734, 2007.

[3] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).

[4] L. Sachs. *Angewandte Statistik*. Springer Verlag, 7-th edition, 1992.

[5] A. Sato and K. Yamada. Generalized learning vector quantization. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, pages 423–9. MIT Press, Cambridge, MA, USA, 1996.

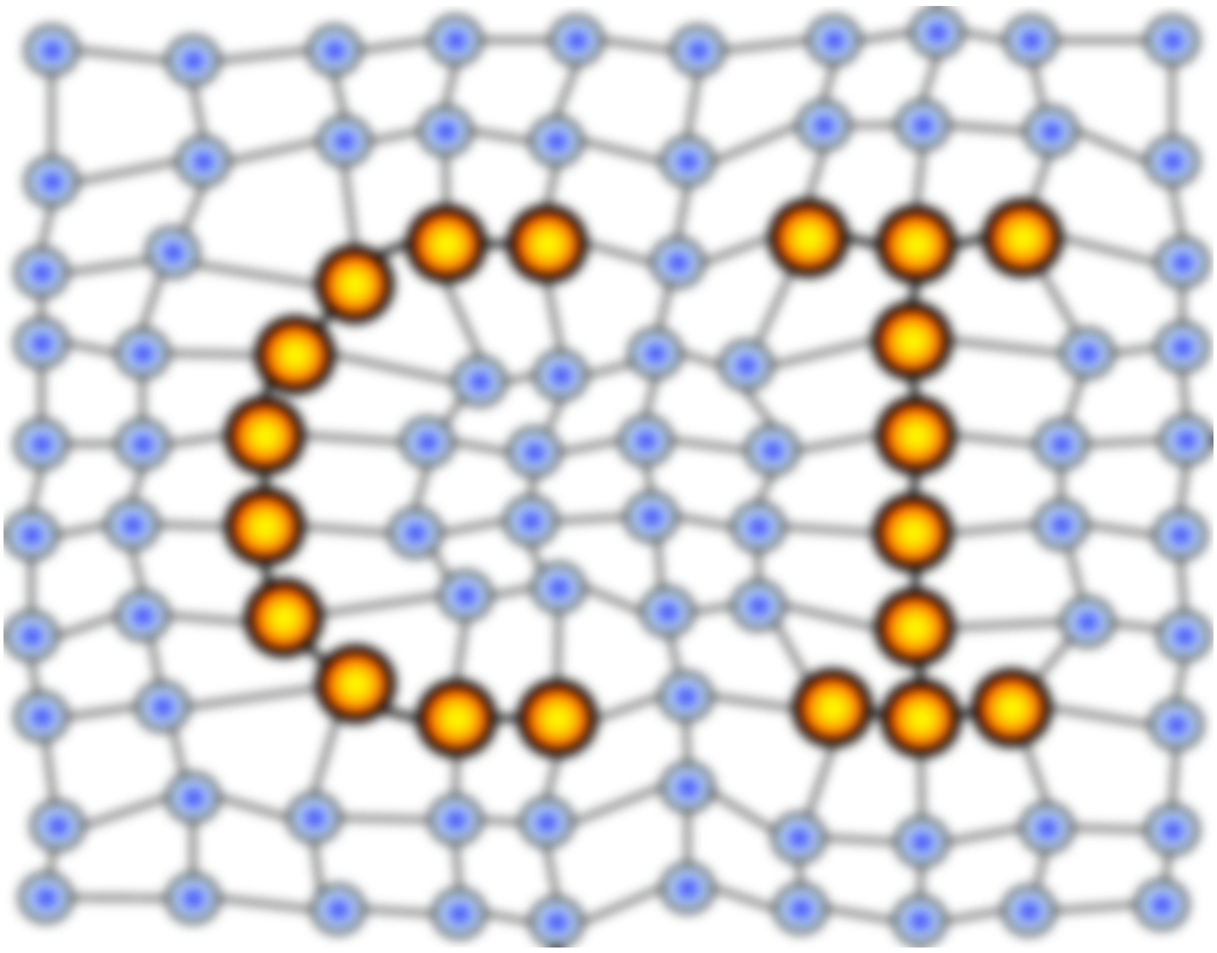
[6] P. Schneider, M.Biehl, and B.Hammer. Matrix adaptation in discriminative vector quantization. Technical Report IFI-08-08, Department of Informatics, TU Clausthal, Germany, 2008.

[7] S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Computation*, 15:1589–1604, 2003.

[8] T. Villmann, M. Strickert, C. Brüß, F.-M. Schleif, and U. Seiffert. Visualization of fuzzy information in fuzzy-classification for image segmentation using MDS. In M. Verleysen, editor, *Proc. Of European Symposium on Artificial Neural Networks (ESANN'2007)*, pages 103–108, Brussels, Belgium, 2007. d-side publications.

Machine Learning Reports

Report 02/2009



Impressum

Machine Learning Reports ISSN: 1865-3960

▽ Publisher/Editors

Prof. Dr. rer. nat. Thomas Villmann & Dr. rer. nat. Frank-Michael Schleif
University of Applied Sciences Mittweida
Technikumplatz 17, 09648 Mittweida - Germany•
http://www.uni-leipzig.de/˜compint

▽ Acknowledgments
We would like to thank the reviewers for their time and patience.