Experiments with Supervised Fuzzy LVQ

Christian Thiel, Britta Sonntag, and Friedhelm Schwenker

Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany
christian.thiel@uni-ulm.de

Abstract. Prototype based classifiers so far can only work with hard labels on the training data. In order to allow for soft labels as input label and answer, we enhanced the original LVQ algorithm. The key idea is adapting the prototypes depending on the similarity of their fuzzy labels to the ones of training samples. In experiments, the performance of the fuzzy LVQ was compared against the original approach. Of special interest was the behaviour of the two approaches, once noise was added to the training labels, and here a clear advantage of fuzzy versus hard training labels could be shown.

1 Introduction

Prototype based classification is popular because an expert familiar with the data can look at the representatives found and understand why they might be typical. Current approaches require the training data to be hard labeled, that is each sample is exclusively associated with a specific class. However, there are situations where not only hard labels are available, but soft ones. This means that each object is assigned to several classes with a different degree. For such cases, several classification algorithms are available, for example fuzzy Radial Basis Function (RBF) networks [1] or fuzzy-input fuzzy-output Support Vector Machines (SVMs) [2], but none of them allows for an easy interpretation of typical representatives. Thus, we decided to take the well-known Learning Vector Quantisation (LVQ) approach and enhance it with the ability to work with soft labels, both as training data and for the prototypes. The learning rules we propose are presented in section 2.

A related approach is the Soft Nearest Prototype Classification [3] with its extension in [4], where prototypes also are assigned fuzzy labels in the training process. However, the training data is still hard labeled. Note that also some variants of fuzzy LVQ exist, for example those of Karayiannis and Bezdek [5] or in [6], that have a hard labeling of the prototypes, and a soft neighbourhood function, exactly the opposite of our approach in this respect.

To assess the classification power of the suggested fuzzy LVQ, we compare its performance experimentally against LVQ1, and study the impact of adding various levels of noise to the training labels.

L. Prevost, S. Marinai, and F. Schwenker (Eds.): ANNPR 2008, LNAI 5064, pp. 125–132, 2008.

2 Supervised Fuzzy LVQ

Like the original Learning Vector Quantisation approach proposed by Kohonen [7], we employ r prototypes, and learn the locations of those. But, our fuzzy LVQ does not work with hard labels, but soft or fuzzy ones. It is provided with a training set M

$$\begin{aligned} &M = \{(x^\mu, l^\mu) |\ \mu = 1, \ldots, n\} \\ &x^\mu \in \mathbb{R}^z : \text{the samples} \\ &l^\mu \in [0;1]^c,\ \sum_{i=1}^{c} l_i^\mu = 1 : \text{their soft class labels} \end{aligned} \tag{1}$$

where n is the number of training samples, z their dimension, and c the number of classes. Feeding a hitherto unseen sample x^{new} to the trained algorithm, we get a response l^{new} estimating a soft class label, of the form described just above.

As second major difference to the original LVQ network, each prototype $p^\eta \in \mathbb{R}^z$ is associated with a soft class label pl^η, which will influence the degree to which the prototypes (and their labels) are adapted to the data during training.

Note that the techniques presented can be applied to various prototype based learning procedures, for example LVQ3 or OLVQ.

The basic procedure to use the algorithm for classification purposes is split into three stages: initialisation, training and answer elicitation. The initialisation part will be explained in section 3, while obtaining an answer for the new sample x^{new} is quite simple: look for the prototype p^η closest to x^{new} (using the distance measure of your choice, we employed the Euclidean distance), and answer with its label pl^η. The training of the network is the most interesting part and will be described in the following.

2.1 Adapting the Prototypes

The basic training procedure of LVQ adapts the locations of the prototypes. For each training sample, the nearest prototype p^* is determined. If the label $pl^*$of the winner p^* matches the label l^μ of the presented sample x^μ, the prototype is shifted into the direction of the sample. When they do not match, the prototype is shifted away from the sample. This is done for multiple iterations.

Working with fuzzy labels, the similarity between two labels should no longer be measured in a binary manner. Hence, we no longer speak of a *match*, but of a similarity S between labels. Presenting a training sample x^μ, the update rule for our fuzzy LVQ is:

$$p^* = p^* + \theta \cdot (\ S(l^\mu, pl^*)\ - e) \cdot (x^\mu - p^*) \tag{2}$$

Still, the prototype is shifted to or away from the sample. The direction now is determined by the similarity S of their labels: if S is higher than the preset threshold e, p^* is shifted towards x^μ, otherwise away from it. The learning rate θ controls the influence of each update step.

As similarity measure S between two fuzzy labels, we opted to use alternatively the simple scalar product (component wise multiplication, then summing up) and the S_1 measure introduced by Prade in 1980 [8] and made popular again by Kuncheva (for example in [9]):

$$S_1(l^a, l^b) = \frac{||l^a \cap l^b||}{||l^a \cup l^b||} = \frac{\sum_i min(l_i^a, l_i^b)}{\sum_i max(l_i^a, l_i^b)} \tag{3}$$

2.2 Adapting the Labels

The label of the prototypes does not have to stay fixed. In the learning process, the winners' labels pl^* can be updated according to how close the training sample $s^\mu \in M$ is. This is accomplished with the following update rule:

$$pl^* = pl^* + \theta \cdot (l^\mu - pl^*) \cdot exp(-\frac{||x^\mu - p^*||^2}{\sigma^2}) \tag{4}$$

The scaling parameter σ^2 of the exponential function is set to the mean Euclidean distance of all prototypes to all training samples. Again, θ is the learning rate.

3 Experimental Setup

The purpose of our experiments was twofold: Assessing the classification power of fuzzy LVQ, and its ability to stand up to noise added to the labels of its training data, all in comparison with standard LVQ1.

As test bed, we employed a fruits data set coming from a robotic environment, consisting of 840 pictures from 7 different classes (apples, oranges, plums, lemons,... [10]). Using results from Fay [11], we decided to use five features: *Colour Histograms* in the RGB space. Orientation histograms on the edges in a greyscale version of the picture (we used both the *Sobel* operator and the *Canny* algorithm to detect edges). As weakest features, colour histograms in the black-white opponent colour space $APQBW$ were calculated, and the mean color information in HSV space. Details on these features as well as references can be found in the dissertation mentioned just above.

The fruits data set originally only has hard labels (a banana is quite clearly a banana). As we need soft labels for our experiments, the original ones had to be fuzzified, which we accomplished using two different methods, K-Means and Keller. In the fuzzy K-Means [12] approach, we clustered the data, then assigned a label to each cluster centre according to the hard labels of the samples associated with it to varying degrees. Then, each sample was assigned a new soft label as a sum of products of its cluster memberships with the centres' labels. The fuzzifier parameter of the fuzzy K-Means algorithm, which controls the smoothness or entropy of the resulting labels, was chosen by hand so that the correspondence between hardened new labels and original ones was around 70%. The number of clusters was set to 35. In the second approach, based on work

by Keller [13], the fuzzyness of a new soft label l^{new} can be controlled very straightforwardly, and using a mixture parameter $\alpha > 0.5$ it can now be ensured that the hardened l^{new} is the same as the original class C^{orig}:

$$l_i^{new} = \begin{cases} \alpha + (1-\alpha) \cdot \frac{n_i}{k} & \text{, if } i = C^{orig} \\ (1-\alpha) \cdot \frac{n_i}{k} & \text{, otherwise} \end{cases}$$

The fraction counts what portion of the k nearest neighbours are of class i. In our experiments, $\alpha = 0.51$ and $k = 5$ were used.

All results were obtained using 5-fold cross validation. Hardening a soft label was simply accomplished by selecting the class i in the label with the highest value l_i, and our accuracy measure compares the hardened label with the original hard one.

Finding a suitable value for the threshold e in equation 2, which controls at what similarity levels winning prototypes are attracted to or driven away from samples, is not straightforward. We looked at intra- versus inter-class similarity values[1], whose distributions of course overlap, but in our case seemed to allow a good distinction at $e = 0.5$.

The 35 starting prototypes for the algorithms were selected randomly from the training data. With a learning rate of $\theta = 0.1$ we performed 20 iterations. One iteration means that all training samples were presented to the network and prototypes and labels adjusted. The experiments were run in online mode, adjusting after every single presentation of a sample.

As mentioned, we also wanted to study the impact of noise on the training labels on the classification accuracy. To be able to compare the noise level on hard and fuzzy labels, noise was simply added to the hard labels, imparting corresponding noise levels to the soft labels derived from the noised hard ones. The procedure for adding noise to the hard labels consisted simply of randomly choosing a fraction (0% to 100%, the noise level) of the training labels, and randomly flipping their individual label to a different class.

4 Results and Interesting Observations

The experiments on the five different features gave rather clear answers. As had to be expected, if no noise is present, the original LVQ1 approach performs best[2]. Then, depending on the feature, at a noise level between 10% and 30%, its accuracy drops rather sharply, and one of the fuzzy LVQ approaches wins (see figure 1). Right after LVQ1 drops, the fuzzy approach with the Keller-initialisation and scalar product as similarity measure remains most stable. Adding more noise, the approach initialised with K-Means and similarity measure S_1 now becomes the clear winner with the highest classification accuracy. This means that once there is a not insignificant level of noise on the (training) labels, a fuzzy LVQ approach is to be preferred over the basic hard LVQ one.

[1] Intra- and inter-class determined with respect to the original hard labels.

[2] Keep in mind that the correspondence between hardened new labels and original ones was only around 70%, giving the LVQ1 approach a headstart.

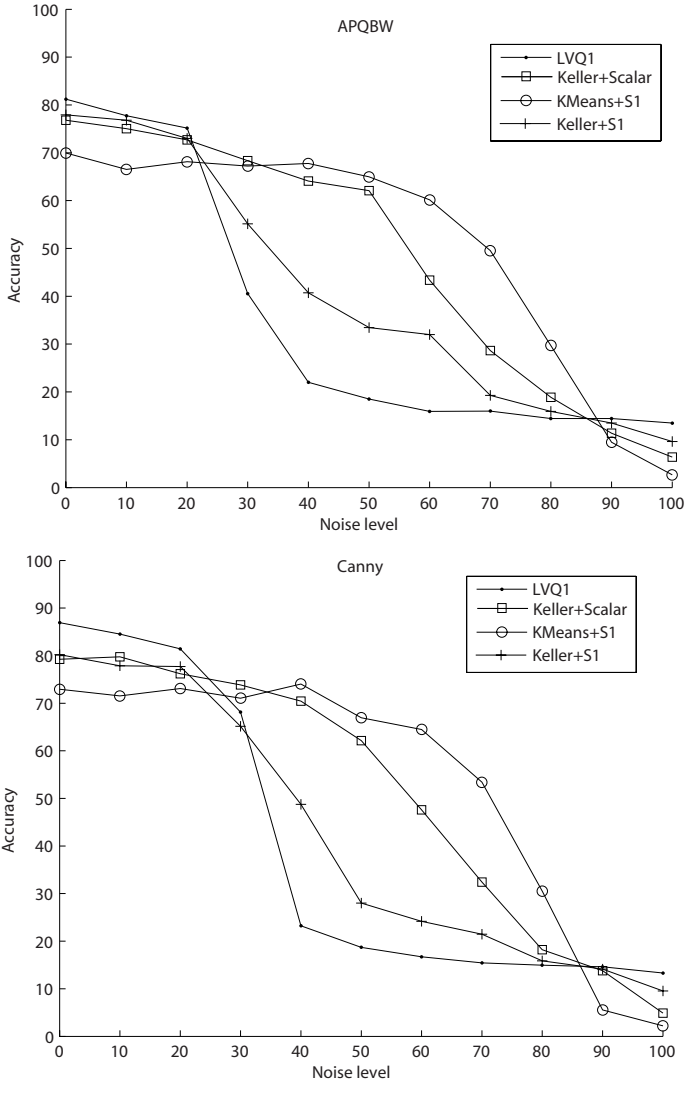

Fig. 1. Plotting the performance (accuracy given in %) of the original LVQ1 algorithm and our fuzzy LVQ approach when adding noise (noise level given in %). The fuzzy LVQ is plotted with different fuzzy label initialisation techniques (K-Means, Keller) and distance measures on the labels (S_1 and scalar product). Results given for two features, *APQBW* and *Canny*. The combination of K-Means and scalar product is omitted from this graph for clarity, as it is very similar to the one with Keller initialisation.

The poor performance of the fuzzy LVQ with Keller fuzzification and S_1 similarity measure has a simple explanation: setting the Keller mixing parameter α to 0.5 and $k := 5$, the possible values for intra- [0.34;1] and inter-class similarities [0;0.96] overlap largely. A similar explanation, experimentally obtained

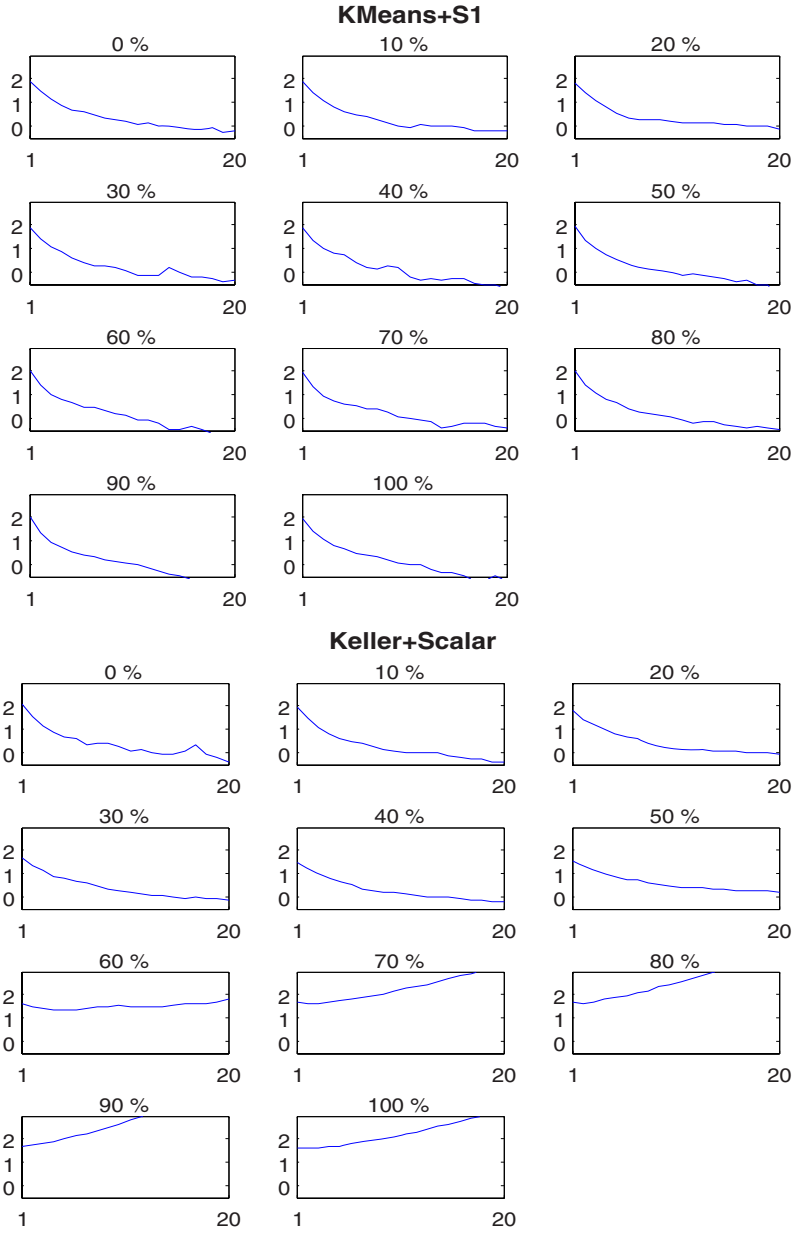

Fig. 2. Showing how much the location of the prototypes changes from iteration to iteration (x-axis, 20 training epochs), depending on how much noise is present (0 % to 100%). Location change is the sum of quadratic distances between the prototypes' position before and after each iteration, logarithmised to base 10. Plots given for Canny feature, and two fuzzy LVQ algorithm variants: K-Means coupled with S_1 and Keller coupled with scalar product.

this time, holds for K-Means fuzzification and the scalar product as distance measure. Initially, the same accuracy as with the S_1 measure is achieved, but this does not hold once noise is added (results not present in figure 1 for reasons of readability).

For higher noise levels, the winning approach is a fuzzy LVQ with K-Means label fuzzification, and S_1 similarity measure. This shows nicely the effect we were hoping to achieve with the process of fuzzifying the labels; the clustering of the training data, and usage of neighbouring labels for the soft labels, encodes knowledge about the label space into the labels itself. Knowledge, which can then be exploited by the fuzzy LVQ approach.

Examining how the labels of the prototypes changed from iteration to iteration (equation 4) of the fuzzy LVQ, we found that they remain rather stable after the first 10 rounds.

Looking closer at how the locations of the prototypes change (compare figure 2, and equation 2) across iterations, we could make an interesting observation. When no noise was added, the movement of the prototypes went down continuously with each iteration. But as soon as we added noise to the labels, the situation changed. The tendency of the volume of the movement was not so clear any more, for some algorithms it would even go up after some iterations before going down again. Reaching a noise level of 40 to 60 percent, the trend even reversed, and the movements got bigger with each iteration, not stabilising any more. The only exception here was the Fuzzy LVQ with K-Means initialisation and S_1 as similarity measure, which explains why it performs best of all variants on high noise levels.

The non-settling of the prototype locations also solves the question why, at a noise level of 100%, the original LVQ has an accuracy of 14%, which is exactly the random guess. It turned out that the cloud of prototypes shifts far away from the cloud of samples, in the end forming a circle around the samples. One random but fixed prototype is now the closest to all the samples, leading to the effect described.

5 Summary

We presented a prototype-based classification algorithm that can take soft labels as training data, and give soft answers. Being an extension of LVQ1, the prototypes are assigned soft labels, and a similarity function between those and a training sample's label controls where the prototype is shifted. Concerning the classification performance, in a noise free situation, the original approach yields the best results. This quickly changes once noise is added to the training labels, here our fuzzy approaches are the clear winners. This is due to information about the label-distribution inherently encoded in each of the soft labels. A finding which seems to hold for other hard-vs-soft scenarios, too, so we are currently investigating RBFs and SVMs in a multiple classifier systems scenario.

References

1. Powell, M.J.D.: Radial basis functions for multivariate interpolation: A review. In: Mason, J.C., Cox, M.G. (eds.) Algorithms for Approximation, pp. 143–168. Clarendon Press, Oxford (1987)
2. Thiel, C., Scherer, S., Schwenker, F.: Fuzzy-Input Fuzzy-Output One-Against-All Support Vector Machines. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part III. LNCS (LNAI), vol. 4694, pp. 156–165. Springer, Heidelberg (2007)
3. Seo, S.: Clustering and Prototype Based Classification. PhD thesis, Fakultät IV Elektrotechnik und Informatik, Technische Universität Berlin, Germany (November 2005)
4. Villmann, T., Schleif, F.M., Hammer, B.: Fuzzy Labeled Soft Nearest Neigbor Classification with Relevance Learning. In: Fourth International Conference on Machine Learning and Applications, pp. 11–15 (2005)
5. Karayiannis, N.B., Bezdek, J.C.: An integrated approach to fuzzy learning vector quantization andfuzzy c-means clustering. IEEE Transactions on Fuzzy Systems 5(4), 622–628 (1997)
6. Wu, K.L., Yang, M.S.: A fuzzy-soft learning vector quantization. Neurocomputing 55(3), 681–697 (2003)
7. Kohonen, T.: Self-organizing maps. Springer, Heidelberg (1995)
8. Dubois, D., Prade, H.: Fuzzy Sets and Systems: Theory and Applications. Academic Press, London (1980)
9. Kuncheva, L.I.: Using measures of similarity and inclusion for multiple classifier fusion by decision templates. Fuzzy Sets and Systems 122(3), 401–407 (2001)
10. Fay, R., Kaufmann, U., Schwenker, F., Palm, G.: Learning Object Recognition in a NeuroBotic System. In: Groß, H.M., Debes, K., Böhme, H.J. (eds.) 3rd Workshop on SelfOrganization of AdaptiVE Behavior SOAVE 2004. Fortschritt-Berichte VDI, Reihe 10, vol. 743, pp. 198–209. VDI (2004)
11. Fay, R.: Feature Selection and Information Fusion in Hierarchical Neural Networks for Iterative 3D-Object Recognition. PhD thesis, University of Ulm, Germany (2007)
12. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–298. University of California Press (1967)
13. Keller, J., Gray, M., Givens, J.: A Fuzzy K Nearest Neighbor Algorithm. IEEE Transactions on Systems, Man and Cybernetics 15(4), 580–585 (1985)