Recognition of Handwritten Digits by Combining Independent Learning Vector Quantizations

Tin Kam Ho

AT&T Bell Laboratories
600 Mountain Avenue, 2C-548C
Murray Hill, NJ 07974, USA

Abstract

Classifiers derived by learning vector quantization (LVQ) have well defined decision regions that can be combined to construct a more accurate classifier. A given point may be included in a number of decision regions associated with different LVQ-classifiers. The relative densities of classes in each region can be combined to obtain a final classification. The method allows useful inferences from small training sets, which is needed for problems involving large variations within each class. In an application of this method to the recognition of handwritten digits, we show that the classifier can be improved almost monotonically without suffering from over-adaptation to the training data.

1 Introduction

The lack of sufficient training data is often an obstacle to building classifiers for patterns with large variations. The problem can be partly remedied by using multiple classifiers simultaneously, so that diverse extrapolations are possible. The success in combining the power of multiple classifiers depends critically on a comparable and consistent representation of the class decisions. One such representation is the rank-order of classes [4].

Certain classifiers use a discriminant function that is consistent and comparable, and thus permits direct combination. An example is the measure used in [5], where classifiers are identified with their decision regions and created stochastically. In this paper we discuss a similar technique on trainable classifiers. We illustrate the technique using as an example classifiers trained by Kohonen's learning vector quantization (LVQ) algorithm [8].

2 Learning vector quantization

A classifier obtained by learning vector quantization is a variant of Kohonen's self-organizing networks. The essence of the method is in deriving a set of *reference vectors* by iterative matching and adaptation to a training set. Following the convention, we represent such reference vectors as *weight vectors* each associated with a *node* in an array with no lateral connections. We will refer to such an array as an *LVQ-array*. An LVQ-array can be used as a classifier when an input sample is matched to the reference vectors and assigned to the class of the closest vector. It is the same as a nearest-neighbor classifier except that the reference vectors are used as prototypes. An LVQ-classifier is obtained by a training procedure described below.

Suppose that we are tackling an m-class ($m \geq 2$) recognition problem in a d-dimensional feature space F. We are given a *training set* $X = x_1, x_2, ..., x_p$ and a *testing set* $Y = y_1, y_2, ..., y_q$, where X and Y are disjoint. Each point in X is labeled with a class.

Let L be an LVQ-array with n nodes $v_1, v_2, ..., v_n$ ($n \geq m$). A weight vector w_j of dimension d is associated with each node v_j. Let the values of w_j at iteration $t = 0$ be initialized by randomly chosen points in X. At each iteration $t > 0$, the weights are modified in the following way. For each point x in X, we compute a distance $d(x, w_j)$ from x to each weight vector w_j, and find the node $v_{j'}$ for which $d(x, w_{j'})$ is the minimum. $w_{j'}$ is then updated by

$$w_{j'}(t+1) \leftarrow w_{j'}(t) + \alpha(t)(x - w_{j'}(t)),$$

where $0 < \alpha(t) < 1$. As t increases, $\alpha(t)$ approaches zero so that the weights converge. This is called an *unsupervised* learning rule because the weights are updated the same way regardless of the class of the input.

After the weights converge, we recompute the distances $d(x, w_j)$ and assign each x to the node to which it is closest. After all points in X are assigned, for each node v_j, we count the number of points of each class that have been assigned to it, and label the node with the class that has the largest count. The labeled array

can then be used as a classifier.

Alternatively, the weight vectors can be updated by a *supervised* learning rule that tends to locate them near the class boundaries. Supervised learning requires each node be associated with a class label during the entire training process. At $t = 0$, each node is initialized and labeled with the class of a randomly chosen point in X. Then at each iteration $t > 0$, every point x is compared to each node and the the closest node $v_{j'}$ is determined. If the class of x is the same as the class of $v_{j'}$, $w_{j'}$ is updated by

$$w_{j'}(t+1) \leftarrow w_{j'}(t) + \alpha(t)(x - w_{j'}(t)),$$

otherwise, $w_{j'}$ is updated by

$$w_{j'}(t+1) \leftarrow w_{j'}(t) - \alpha(t)(x - w_{j'}(t)),$$

and the class of $v_{j'}$ is updated by the majority of the classes of the points that have been assigned to it at that iteration. After the weights converge, the array together with the class labels can then be used as a classifier in the same way as before.

Like most minimum-distance classifiers, the behavior of an LVQ-classifier is difficult to analyze. One reasonable conjecture is that the accuracy of the classifier depends on its complexity, that is, the number of nodes used, on the basis that its decision boundaries can better match irregular and intermixing class-conditional distributions. Therefore one plausible way to improve the performance is to add more nodes to the array.

However, in most practical applications, one knows little about the class-conditional distributions to chose an optimal number of nodes. Furthermore, there is no guarantee that the accuracy of a classifier will necessarily improve with increasing complexity. This is especially true for a classifier built by adaptive training that minimizes its errors on the training set. Given a training set of limited size, the problem of over-adaptation may cause the classifier to perform poorly on unseen test data. For an LVQ-classifier, this occurs when there are too many reference vectors so that each is modified by only a few training samples. Their final locations are too close to those of the training samples so that extrapolation to unseen cases is unreliable.

In what follows, we will give an alternative method that extends an LVQ-classifier systematically so that its performance improves without suffering from over-adaptation. We will show that the performance of such a classifier can be improved to equal or exceed that of a conventional nearest-neighbor classifier where all training samples are used as prototypes.

3 Combination based on relative class densities

The final locations of the reference vectors are affected by their initialization as well as by modifications in the training process. By changing the initial conditions, different classifiers can be produced with the same training process [3].

Suppose that the weights are initialized by randomly chosen points in the training set X. For an array of n nodes and a training set with p samples, C_p^n potentially different classifiers can be generated with equal likelihood. Interestingly, experimental results show that the overall classification accuracy is insensitive to the initial conditions (see Section 4). However, despite the similarity in the overall accuracy, for each individual sample, the decisions of each classifier are potentially different.

Geometrically, the reference vectors in an LVQ-array determine a Voronoi tessellation that partitions the feature space into different classes. Each node defines a decision region that includes all the points in the feature space that are closer to it than to any other node. The decision of the classifier is identical for all the points inside this region. In other words, the node may be considered as a representative having all the attributes of the points surrounding it. A node therefore represents a subset of the feature space with constant class membership, and it can be considered as a *model* in the analysis in [6].

Let m be the number of classes, and n be the number of nodes in a given LVQ-array. Let s be the number of LVQ-arrays that are independently initialized and trained. For a given point x in F, let $v_{j'}^k(x)$ be its nearest-neighbor among all nodes v_j $(j = 1, ..., n)$ in array k $(k = 1, ..., s)$. Given this, let the probability that x belongs to class c be denoted by $P(c|v_{j'}^k(x))$.

$$P(c|v_{j'}^k(x)) \;=\; \frac{P(c, v_{j'}^k(x))}{\sum_{l=1}^{m} P(c_l, v_{j'}^k(x))}$$

can be estimated by the fraction of class c points over all points in X that are assigned to $v_{j'}^k(x)$. We call $P(c|v_{j'}^k(x))$ the *relative density* of class c in $v_{j'}^k(x)$ [1].

We propose a discriminant function

$$g_c(x) \;=\; \frac{1}{s}\sum_{k=1}^{s} P(c|v_{j'}^k(x)),$$

and the decision rule that assigns x to class c for which $g_c(x)$ is the maximum. The discriminant gives a new

[1] this measure is the main difference between this method and the one proposed in [6] [7].

classifier that combines the power of the s-many LVQ-arrays. The classifier can be extended by adding more arrays that differ from the previous ones by the initial conditions.

When s different LVQ-arrays are used, each point x is matched to s many nearest-neighbors. By using the discriminant function $g_c(x)$, we are drawing inferences about x from s potentially different directions where its nearest-neighbors are located. Unlike the method of *Parzen windows*, the sizes of the individual decision regions (the kernels) do not decrease monotonically as we add more arrays, so that there is no need for prohibitively many training samples for the estimates to remain reliable. However, it is important that the nearest-neighbors are distributed evenly in all directions of x, and this is an effect we intend to achieve by using randomized initial conditions. The accuracy of the classifier improves as more arrays are added, because more neighbors to a point x are referred to so that the relative densities are better approximated.

4 Experimental results

The strength of the method is best seen in problems where a small training set is unlikely to represent fully the variations of the samples. The recognition of handwritten digits is one such problem, and is therefore chosen for our experiments.

A collection of handwritten digits contained in the NIST (National Institute of Standards and Technology) Special Database 3 was used for both training and testing. The digits were extracted from images of a set of forms designed to collect handwriting samples. The images were scanned at 300 pixels per inch and binarized.

The database is organized by the serial number of the forms from which the digits were extracted. The first sample of each digit from each form was used in our test. Samples from the first 500 forms were used in training and those from the second 500 forms were used in testing. Some digits in certain forms are missing from the database, so that the training and testing sets contain 4997 and 4975 samples respectively.

A feature vector of 512 dimensions was derived for each digit as follows. The digit images were size-normalized to 16x16 pixels, and then convolved with a set of 32 structural feature masks that were defined on windows of 7x7 pixels [1]. A threshold associated with each mask was applied to the convolution responses to detect the presence of that particular feature in a 7x7 neighborhood centered at each pixel. The image area was then partitioned into 16 nonoverlapping regions each of size 4x4. The counts of positive responses to each feature in each region were represented in a vector of 512 (32x16) dimensions. Euclidean distance was used in all the matchings.

A series of tests were performed using LVQ-arrays of different sizes. We compare the results of expanding an LVQ-classifier first by adding in more nodes, and then by adding in more arrays of the same size. Up to 20 arrays of the same size were combined. The choice of the number 20 was simply by the limit of available computing resources. Each array was randomly initialized and trained using the same parameters ($\alpha(0) = 0.3$, $\Delta\alpha = -0.05$) until the weights converge. Both the unsupervised and the supervised learning rules were tested.

We first used each array as an individual classifier and applied it on the training set. Table 1 shows their accuracies when the supervised learning rule was used. Arrays of the same number of nodes yield similar levels of classification accuracy with only minor variations. The accuracy increases with the increase in the number of nodes.

Table 1: Performance of individual classifiers on training data (% correct).

	n: number of nodes in each array					
array k	100	200	300	400	500	1000
1	92.88	94.80	95.56	95.72	95.80	97.02
2	92.14	95.14	95.48	95.88	96.40	96.88
3	93.50	94.38	95.02	95.06	95.88	96.94
4	93.16	94.40	94.78	95.88	96.70	97.02
5	91.09	94.64	95.64	96.42	95.58	96.96
6	92.44	94.76	95.96	95.64	96.56	97.04
7	93.40	94.68	95.12	95.28	95.58	97.16
8	91.31	94.54	95.06	96.02	95.90	97.00
9	92.40	95.22	95.24	96.00	96.12	96.50
10	91.33	94.36	95.92	95.46	96.00	96.96
11	93.52	93.70	95.50	95.86	96.04	97.04
12	91.19	94.64	95.54	95.78	96.10	96.72
13	91.88	94.60	94.76	96.14	96.08	96.58
14	91.33	94.88	95.54	95.48	96.48	96.76
15	92.48	95.14	95.14	95.64	96.06	96.88
16	92.54	94.02	95.34	95.88	96.10	96.98
17	91.47	94.94	95.64	95.50	95.92	96.88
18	92.86	94.46	95.24	95.80	96.16	97.00
19	92.44	94.06	95.48	95.86	96.06	97.16
20	91.11	94.16	94.86	96.18	95.90	97.26

The arrays together with the combination algorithm were then applied to classify the points in both the training and testing sets. The results are shown in Tables 2 and 3, respectively. In all the tests the classifiers trained by the supervised learning rule were found better than those by the unsupervised learning rule. For simplicity we report only the results of the former.

From Tables 2 and 3 one can observe the two-way improvement of performance: by increasing the number of nodes in each array, and by adding more arrays

Table 2: Performance of combined classifier on training data (% correct).

s: no. of arrays	n: no. of nodes in each array					
	100	200	300	400	500	1000
1	92.88	94.80	95.56	95.72	95.80	97.02
2	94.80	96.04	97.02	97.36	97.30	98.32
3	95.06	96.28	96.68	97.22	97.10	98.36
4	95.58	96.68	97.02	97.34	97.64	98.52
5	95.24	96.72	97.00	97.52	97.54	98.48
6	95.50	96.86	97.32	97.54	97.76	98.68
7	95.50	96.86	97.34	97.42	97.84	98.70
8	95.64	97.16	97.30	97.72	97.88	98.66
9	95.82	97.04	97.32	97.60	97.86	98.72
10	95.78	97.04	97.36	97.78	97.98	98.86
11	95.74	97.18	97.38	97.74	97.88	98.86
12	95.88	97.08	97.50	97.86	97.88	98.80
13	95.92	97.06	97.50	97.84	97.88	98.78
14	95.94	97.10	97.44	97.86	98.06	98.74
15	96.02	97.08	97.54	97.84	98.06	98.80
16	95.86	97.22	97.54	97.90	98.12	98.72
17	95.90	97.20	97.60	97.82	98.14	98.74
18	95.90	97.28	97.58	97.82	98.16	98.78
19	96.02	97.28	97.62	97.86	98.16	98.80
20	96.02	97.30	97.56	97.88	98.10	98.84

Table 3: Performance of combined classifier on testing data (% correct)

s: no. of arrays	n: no. of nodes in each array					
	100	200	300	400	500	1000
1	91.38	93.41	93.29	93.47	93.49	93.79
2	92.48	94.05	94.91	94.33	94.61	94.61
3	93.13	95.04	95.16	95.06	95.26	95.34
4	93.59	95.26	95.48	95.44	95.76	95.82
5	93.33	95.22	95.58	95.42	95.70	95.90
6	93.93	95.34	95.84	95.52	95.90	96.00
7	94.07	95.58	95.92	95.54	95.98	95.86
8	94.35	95.56	95.90	95.64	96.04	95.94
9	94.49	95.76	95.88	95.88	96.00	96.14
10	94.51	95.68	96.08	95.98	96.18	96.20
11	94.67	95.76	96.00	96.02	96.14	96.22
12	94.65	95.82	96.12	96.04	96.04	96.26
13	94.57	95.78	96.00	96.12	96.06	96.16
14	94.75	95.80	96.18	96.16	96.12	96.24
15	94.61	95.80	96.22	96.10	96.20	96.32
16	94.47	95.78	96.30	96.02	96.20	96.30
17	94.57	95.74	96.26	96.12	96.24	96.26
18	94.73	95.78	96.12	96.12	96.22	96.30
19	94.67	95.84	96.26	96.10	96.30	96.38
20	94.81	95.86	96.16	95.92	96.26	96.26

to the classifier. Notice that improvements due to increasing number of nodes in each array do not carry over to the testing set. Table 4 shows the differences between corresponding entries in Tables 2 and 3. Notice that the discrepancy increases with the number of nodes, as a result of over-adaptation. On the other hand, the performance of the combination using our method shows *nearly-monotonic* improvement with increases in the number of arrays on *both* training and testing sets, regardless of the complexity (number of nodes) of the individual arrays.

Table 4: Differences between training and testing set performances (% correct)

s: no. of arrays	n: no. of nodes in each array					
	100	200	300	400	500	1000
1	1.50	1.39	2.27	2.25	2.31	3.23
2	2.32	1.99	2.11	3.03	2.69	3.71
3	1.93	1.24	1.52	2.16	1.84	3.02
4	1.99	1.42	1.54	1.90	1.88	2.70
5	1.91	1.50	1.42	2.10	1.84	2.58
6	1.57	1.52	1.48	2.02	1.86	2.68
7	1.43	1.28	1.42	1.88	1.86	2.84
8	1.29	1.60	1.40	2.08	1.84	2.72
9	1.33	1.28	1.44	1.72	1.86	2.58
10	1.27	1.36	1.28	1.80	1.80	2.66
11	1.07	1.42	1.38	1.72	1.74	2.64
12	1.23	1.26	1.38	1.82	1.84	2.54
13	1.35	1.28	1.50	1.72	1.82	2.62
14	1.19	1.30	1.26	1.70	1.94	2.50
15	1.41	1.28	1.32	1.74	1.86	2.48
16	1.39	1.44	1.24	1.88	1.92	2.42
17	1.33	1.46	1.34	1.70	1.90	2.48
18	1.17	1.50	1.46	1.70	1.94	2.48
19	1.35	1.44	1.36	1.76	1.86	2.42
20	1.21	1.44	1.40	1.96	1.84	2.58

The accuracy of the combined classifier continues to increase with additional arrays despite minor local fluctuations. Arguably, in practice, one may not be able to expand the classifier to perfect accuracy within feasible complexity. Nevertheless, the behavior of the classifier permits a tradeoff between the desired accuracy and the computational cost. We emphasis that such a choice is not always available in other methodologies because of the problem of over-adaptation. The advantage of the combined classifier in maintaining generalization power is similar to what we have observed with another type of stochastic models [7].

Given that the training set is fixed, such a nearly-monotonic improvement cannot be achieved using conventional nearest-neighbor (NN) or k-nearest-neighbor (KNN) classifiers. For comparison purposes, we tested an NN classifier and a KNN classifier with k=3 on the same data using all the training samples as prototypes. The same features and distance measure were used. The correct rates are 95.50% for NN and 96.02% for KNN respectively. These are below the 96.18% achieved by the combination of 10 500-node LVQ-arrays, while the computational costs are roughly equal.

5 Conclusions

A method is given that creates an accurate classifier by combining less accurate classifiers obtained by LVQ. The combination is based on a discriminant that averages relative class densities in the decision re-

gions associated with each classifier. Randomly generated initial conditions allows some independence between the classifiers. The effectiveness of the method is demonstrated in an application to handwritten digit recognition. It is shown that the combined classifier can be improved nearly-monotonically without suffering from over-adaptation to the training data.

We have shown that the combined classifier could perform better than traditional nearest-neighbor classifiers. Yet exactly when this happens in a general context remains uncertain. Also, the method in its current form does not rigorously enforce independence between the classifiers. Ideally, the individual classifiers should be selected in a way that optimizes their combined performance within a limit of complexity. Finally, the method of relative class densities can be used to combine other types of stochastic models. Necessary conditions for improvements in classification accuracy are studied in [2].

Acknowledgements

The author is thankful to Eugene and Evelyn Kleinberg for stimulating discussions on the theory of stochastic modeling that motivated this work. Helpful comments from Henry Baird, David Ittner, and Hing-Kai Hung are also much appreciated.

References

[1] H.S. Baird, H.P. Graf, L.D. Jackel, W.E. Hubbard, A VLSI Architecture For Binary Image Classification, in J.C. Simon, ed., *From Pixels to Features* , North Holland, 1989, 275-286.

[2] R. Berlind, Doctoral Dissertation, Department of Mathematics, SUNY at Buffalo, in preparation.

[3] L.K. Hansen, P. Salamon, Neural Network Ensembles, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-12**, 10, October 1990, 993-1001.

[4] T.K. Ho, *A Theory of Multiple Classifier Systems And Its Application to Visual Word Recognition*, Doctoral Dissertation, Department of Computer Science, SUNY at Buffalo, 1992, 175-183.

[5] E.M. Kleinberg, Stochastic Discrimination, *Annals of Mathematics and Artificial Intelligence*, **1**, 1990, 207-239.

[6] E.M. Kleinberg, The Theory of Stochastic Modeling in Pattern Recognition, in preparation.

[7] E.M. Kleinberg, T.K. Ho, Pattern Recognition by Stochastic Modeling, *Proceedings of the Third International Workshop on Frontiers of Handwriting Recognition*, Buffalo, May 1993.

[8] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, 1984. 3rd ed., 1989.