5 A Batch Learning Vector Quantization Algorithm for Nearest Neighbour Classification

Abstract- We introduce in this chapter a batch learning algorithm to design the set of prototypes of 1-nearest-neighbour classifiers. Like Kohonen's LVQ algorithms, this procedure tends to perform vector quantization over a probability density function that has zero points at Bayes borders. Although it differs significantly from their online counterparts since: (1) its statistical goal is clearer and better defined; and 2) it converges superlinearly due to its use of the very fast Newton's optimization method. Experiments results using artificial data confirm faster training time and better classification performance than Kohonen's LVQ algorithms.

Index terms- Learning Vector Quantization, Newton's optimization, Nearest Neighbour Classification, Batch learning algorithms

Abbreviations- BLVQ- Batch Learning Vector Quantization, LVQ- Learning Vector Quantization, NN- Nearest Neighbour.

1. Introduction

Kohonen's LVQ algorithms (Kohonen, 1996) are statistical learning procedures that *tend to* perform VQ (Cover & Hart, 1967) over a probability density function that has zero points at Bayes borders. Thus, the labelled prototypes computed with these algorithms together with the nearest neighbour rule define an estimator of the Bayes classifier.

In this chapter, we introduce a batch learning procedure (hereafter, BLVQ) to design the set of prototypes of nearest neighbour classifiers. BLVQ relies on the same idea than Kohonen's LVQ algorithms but its generalization and convergence properties seems better due to:

1) The minimum points of its loss function are clearer defined from a statistical point of view and better approximated than Kohonen's LVQ algorithms
2) BLVQ converges superlinearly to an asymptotically stable point (e.g. a minimum) of its loss risk function since it uses the very fast Newton's optimization method. By contrast, Kohonen's LVQ algorithms employ on-line gradient descent.

In the next section the mathematical framework of our formulation is introduced. Section 3 describes the derivation of the BLVQ (Batch Learning Vector Quantization) algorithm. In Section 4, its generalization properties are analysed. Section 5 shows the experimental results of our algorithm over Kohonen's LVQ algorithms using artificial data. In section 6 some discussion is given. Finally, we end with some concluding remarks.

2. Mathematical framework

2.1. Basics of our approach.

We want to design a labelled codebook C_l for a nearest neighbour classifier that uses a distance metric $D_q(\mathbf{x},\mathbf{m}_\mathbf{j})=\left(\sum_{i=1}^{k}\left|x_i-m_{ji}\right|^q\right)^{1/q}$ where q is a parameter fixed by the user. We compute C_l in the learning phase as follows (See fig. 1):

Step 1. First, we compute an unlabeled codebook $C_{ul}=\{\mathbf{m}_\mathbf{i}, \mathrm{i}=1,\ldots,\mathrm{M}\}$ for a Nearest-Neighbour vector quantifier that uses D_q as a distance metric. C_{ul} is calculated through a vector quantization process over a probability density function that has zero points at Bayes borders. This density function is defined using the original class density functions.

Step 2. We then assign to each codevector m_i a class label with a labelling schema. The result of this process is a labelled codebook $C_l=\{(\mathbf{m}_\mathbf{i}, \text{class label}(\mathbf{m}_\mathbf{i})), \mathrm{i}=1,\ldots,\mathrm{M}\}$.

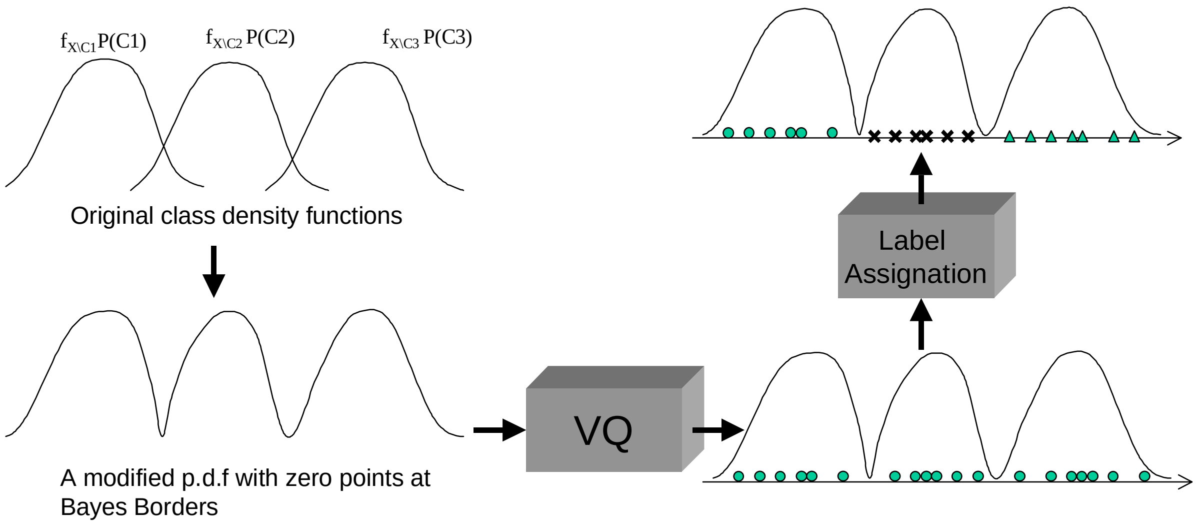

$f_{X\backslash C1}P(C1)$
$f_{X\backslash C2}P(C2)$
$f_{X\backslash C3}P(C3)$
Original class density functions
A modified p.d.f with zero points at Bayes Borders
VQ
Label Assignation

Fig.1. Our approach to the design of the codebook of NN classifiers: We perform VQ over a modified density function. This function tends to zero at Bayesian borders and is defined using the original class density functions. Once the unlabeled codevectors are computed though a VQ process, we assign them to one of the existing classes.

2.2. Optimal Design of the unlabeled codebook C_{ul}.

As we have just seen, an unlabelled codebook $C_{ul}=\{\mathbf{m}_i, i=1,...,M\}$ for a vector quantizer VQ is computed in the first step of our learning procedure. In this subsection we introduce the loss functional, called I_1[VQ], which must be minimised to compute C_{ul} in the vector quantization process. This functional is presented for the case in which the density functions are known and for any value of q. Let us start with some definitions before we present in detail how I_1 is defined.

A VQ of dimension k and size M is defined as a mapping from a k-dimensional input space X that belongs to $\Re^k$, into a codebook (or set of prototypes) $C=\{\mathbf{m}_i, i=1,...,M\}$. Associated with every codevector m_i, there is a region of influence R_i where VQ maps any input vector that falls in it to m_i. Since we use a nearest neighbour quantizer, R_i is defined by

$$R_i = \left\{ \mathbf{x} \middle| D_q(\mathbf{x},\mathbf{m_i}) = \min_{j=1..M} D_q(\mathbf{x},\mathbf{m_j}) \right\} \tag{1}$$

where $D_q(\mathbf{x},\mathbf{m_j}) = \left(\sum_{i=1}^{k} |x_i - m_{ji}|^q \right)^{1/q}$. (Note that in the case q=2, D_q is the Euclidean distance metric.) Thus the VQ mapping, denoted as VQ(x), can be expressed as

$$VQ(\mathbf{x}) = \sum_{j=1}^{M} 1(\mathbf{x} \in R_j)\mathbf{m_j} \quad where\ 1(\mathbf{x} \in R_j) = \begin{cases} 1 & si\ \mathbf{x} \in R_j \\ 0 & si\ \mathbf{x} \notin R_j \end{cases} \tag{2}$$

Now, the goal is to find a suitable measure of performance that the desired codebook C_{ul} minimises. This overall performance in vector quantization can be expressed in terms of a statistical criterion that measures the average quantization error over the total sequence of input patterns to be quantized. In our case we do not make use of the density function of the input space as usual. Instead, we employ a modified probability density function that takes zero values in Bayes borders. Consequently, the expected error of the vector quantizer is given by the functional

$$I_1[VQ] = \sum_{l=1}^{CL} \int_{B_l} D_q^q(\mathbf{x}, VQ(\mathbf{x}))g_l(\mathbf{x})d\mathbf{x} = \sum_{l=1}^{CL} \int_{B_l} D_q^q(\mathbf{x}, VQ(\mathbf{x})) \frac{f_{X|C_l}P(C_l) - f_{X|C_r}P(C_r)}{K_l} d\mathbf{x} \tag{3}$$

where CL is the number of existing classes, $\{B_l, l=1,...,CL\}$ are the Bayes regions, K_l is a constant that ensures that $\{g_l, i=1,...,CL\}$ is a density function and, C_l and C_r are the classes with the highest posterior probabilities in the Bayes region B_l. More precisely, $f_{X\backslash Cl}P(C_l)$ and $f_{X\backslash Cr}P(C_r)$ are defined by

$$\begin{aligned} f_{X|C_l}P(C_l) &= \max_{i=1..CL, \forall \vec{x} \in B_l} f_{X|C_i}(\mathbf{x})P(C_i) \\ f_{X|C_r}P(C_r) &= \max_{i \neq l, \forall \vec{x} \in B_l} f_{X|C_i}(\mathbf{x})P(C_i) \end{aligned} \tag{4}$$

where $\{f_{X\backslash Ci}, i=1,...,CL\}$ are the class density probabilities and $\{P(C_i), i=1...CL\}$ are the class priors. Since Dq is locally defined ($D_q^q(\mathbf{x}, VQ(\mathbf{x})) = \sum_{j=1}^{M} 1(\mathbf{x} \in R_j) D_q^q(\mathbf{x}, \mathbf{m_j})$), I_1 can be further developed:

$$I_1[VQ] = \sum_{l=1}^{CL} \int_{B_l} \sum_{j=1}^{M} 1(\mathbf{x} \in R_j) D_q^q(\mathbf{x}, \mathbf{m_j}) g_l(\mathbf{x})d\mathbf{x} = \sum_{l=1}^{CL} \sum_{j=1}^{M} \int_{B_l \cap R_j} D_q^q(\mathbf{x}, \mathbf{m_j}) g_l(\mathbf{x})d\mathbf{x} \tag{5}$$

2.2. Quasi -Optimal Design of the unlabeled codebook C_{ul} for D_2.

As we pointed out before, we want to compute an optimal C_{ul} that minimizes the functional $I_1[VQ]$. Let us suppose that it exists a Bayes region B_{lj} for every R_j that satisfies $B_{lj} \cap R_j \approx R_j$. Then (5) can be approximated by

$$I_2[VQ] = \sum_{j=1}^{M} \int_{R_j} D_q^q(\mathbf{x}, \mathbf{m_j}) g_{lj}(\mathbf{x}) d\mathbf{x} \tag{6}$$

If q=2, D_2 is the Euclidean distance and equation (6) admits an analytical solution to the minimization problem. The partial derivative of I_2 respect to m_i in those points where I_2 is differentiable, assuming that the conditions to interchange $\partial \bullet / \partial \mathbf{m_j}$ and $\int$ operator are satisfied (see (Bottou, 1998) for technical details), is

$$\frac{\partial I_2[VQ]}{\partial \vec{m}_j} = -2 \int_{R_j} (\mathbf{x} - \mathbf{m_j}) g_{lj}(\mathbf{x}) d\mathbf{x} \tag{7}$$

Solving the equations $\dfrac{\partial I_2[VQ]}{\partial \mathbf{m_j}} = 0$ j=1...M yields

$$\mathbf{m_j}\Big|_{opt} = \frac{\int_{R_j} \mathbf{x}\, g_{lj}(\mathbf{x}) d\mathbf{x}}{\int_{R_j} g_{lj}(\mathbf{x}) d\mathbf{x}} \qquad j = 1..M \tag{8}$$

2.3. An empirical estimator of $I_2[VQ]$

If the expected risk functional $I_2[VQ]$ was known, one could compute C_{ul} by simply applying equation (8). In practice I_2 [VQ] is unknown because the class density functions are unknown. In these cases, the only information available from the classification problem is one or several data sets $\mathbf{D_N}$={($\mathbf{x}_i$, class index ($\mathbf{x}_i$)), i=0..N-1}. Using one of these sets called the training set **T**, I_2[VQ] can be approximated by the empirical risk I_{emp2}[VQ]:

$$\begin{aligned} I_{emp2}[VQ] = \widehat{I}_2[VQ] = \sum_{j=1}^{M} \frac{1}{\widehat{K}_{lj} N_{lj}} \sum_{i=0}^{N-1} 1(\mathbf{x_i} \in R_j) 1(\mathbf{x_i} \in \widehat{C}_{lj}) \widehat{P}(\widehat{C}_{lj}) D_q^q(\mathbf{x_{ii}}, \mathbf{m_j}) - \\ - \sum_{j=1}^{M} \frac{1}{\widehat{K}_{lj} N_{rj}} \sum_{i=0}^{N-1} 1(\mathbf{x_i} \in R_j) 1(\mathbf{x_i} \in \widehat{C}_{rj}) \widehat{P}(\widehat{C}_{rj}) D_q^q(\mathbf{x_i}, \mathbf{m_j}) \end{aligned} \tag{9}$$

where $\hat{C}_{lj}$ is the first majority class of data samples that fall in Rj, $\hat{C}_{lr}$ is the second majority class of data samples that fall in Rj, N_{lj} is the number of training samples that belong to $\hat{C}_{lj}$, N_{lr} is the number of training samples that belong to $\hat{C}_{lr}$ and 1 is the indicator function.

If we use N_{lj}/N to estimate $P(C_{lj})$ and N_{rj}/N to estimate $P(C_{rj})$ then eq. (9) can be rewritten as

$$\begin{aligned}\mathrm{I}_{emp2}[VQ] = \hat{\mathrm{I}}_2[VQ] = \sum_{j=1}^{M} \frac{1}{N} \sum_{i=0}^{N-1} 1(\mathbf{x_i} \in R_j) 1(\mathbf{x_i} \in \hat{C}_{lj}) D_q^q(\mathbf{x_i}, \mathbf{m_j}) - \\ - \sum_{j=1}^{M} \frac{1}{N} \sum_{i=0}^{N-1} 1(\mathbf{x_i} \in R_j) 1(\mathbf{x_i} \in \hat{C}_{rj}) D_q^q(\mathbf{x_i}, \mathbf{m_j})\end{aligned} \tag{10}$$

Solving the equation $\frac{\partial I_{emp2}[VQ]}{\partial \mathbf{m_j}} = 0$ j=1,...,M for q=2 yields

$$\hat{\mathbf{m}}_{\mathbf{j}}\Big|_{opt} = \frac{\sum_{i=1}^{N_{Rj,\hat{C}lj}} \mathbf{x_i}\big| R_j, \hat{C}_{lj} - \sum_{u=1}^{N_{Rj,\hat{C}rj}} \mathbf{x}_u \big| R_j, \hat{C}_{rj}}{N_{R_j,\hat{C}_{lj}} - N_{R_j,\hat{C}_{rj}}} \qquad j = 1,...,M \tag{11}$$

where $N_{Rj,\hat{C}lj}$ is the number of data samples that fall in Rj and belong to class $\hat{C}_{lj}$ and $N_{Rj,\hat{C}rj}$ is the number of data samples that fall in Rj and belong to $\hat{C}_{rj}$. It is easy to show that equation (11) is the empirical estimator of the optimal solution given in equation (8).

3. The Batch Learning Vector Quantization Algorithm

In this section, we introduce the BLVQ algorithm that addresses the problem of designing a codebook for an Euclidean nearest neighbour classifier. The algorithm is divided into two steps: it performs Newton's optimization over $I_{emp2}[VQ]$ and then it assigns class labels to codevectors.

3.1. Step 1: Vector Quantization

The process of vector quantization simply consist of minimizing the functional $I_{emp2}[VQ]$ with an optimization algorithm. We have chosen the Newton's optimization method since the Hessian matrix can be easily computed. Hence the update equation has the following form:

$$\mathbf{m}[n+1] = \mathbf{m}[n] - \mathbf{H}^{-1}[n] \frac{\partial \mathrm{I}_{emp2}[VQ[n]]}{\partial \mathbf{m}[n]} \tag{12}$$

$$\text{where} \quad \mathbf{m} = \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_M \end{bmatrix}, \quad \mathbf{H}^{-1} = \left[\frac{\partial^2 I_{emp2}[VQ]}{\partial \mathbf{m}_i \partial \mathbf{m}_j} \right]$$

Solving $\dfrac{\partial I_{emp2}[VQ[n]]}{\partial \mathbf{m}[n]}$ and H yield

$$\frac{\partial I_{emp2}[VQ]}{\partial \mathbf{m}_j} = \frac{1}{\widehat{K}_{lj} N} \left(\sum_{i=1}^{N_{R_j,\widehat{C}_{lj}}} \left(\mathbf{x_i} \middle| R_j, \widehat{C}_{lj} - \mathbf{m}_j \right) - \sum_{u=1}^{N_{R_j,\widehat{C}_{rj}}} \left(\mathbf{x}_u \middle| R_j, \widehat{C}_{rj} - \mathbf{m}_j \right) \right) \tag{13}$$

$$\frac{\partial^2 I_{emp2}[VQ]}{\partial \mathbf{m}_j \partial \mathbf{m}_i} = \begin{cases} \dfrac{1}{\widehat{K}_{lj} N} \left(N_{R_j,\widehat{C}_{lj}} - N_{R_j,\widehat{C}_{rj}} \right) \mathbf{I} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where I is the kxk identity matrix

As we see from inspecting eq.(14), H is a diagonal matrix, so the iterative equation to minimize I_{emp2} is

$$\mathbf{m}_j[n+1] = \left. \frac{\sum_{i=1}^{N_{R_j,\widehat{C}_{lj}}} \mathbf{x_i} \middle| R_j, \widehat{C}_{lj} - \sum_{u=1}^{N_{R_j,\widehat{C}_{rj}}} \mathbf{x_u} \middle| R_j, \widehat{C}_{rj}}{N_{R_j,\widehat{C}_{lj}} - N_{R_j,\widehat{C}_{rj}}} \right|_{\mathbf{C} = \mathbf{C}[n]} \tag{15}$$

where C[n] is the codebook at time n.

An important remark here is the convergence speed of the algorithm. Let {m[n]} a sequence generated by algorithm (15) which convergence to a point m[∞]. As $\lim_{n\to\infty} \mathbf{H}[n] = \mathbf{H}(\mathbf{m}[\infty])$ then $\mathbf{m}[n] \to \mathbf{m}[\infty]$ superlinearly and usually quadratically (See section 1.4. in reference (Hestenes, 1980)).

In summary, the vector quantization process using the Newton's optimization over the functional I_{emp2} is shown below.

Input:* T** *(training set)*, ***rlen *(running length)*

1. *Initialize C_{ul}*
2. *Compute* $\mathrm{T}_{\mathrm{R}_j[n]} = \{\mathbf{x}_i | \mathrm{R}_j[n], \text{class label}(\mathbf{x}_i | \mathrm{R}_j[n])\}$

   Where $\mathrm{R}_j[n] = \left\{\mathbf{x} \middle| \mathrm{D}_2(\mathbf{x}, \mathbf{m}_j[n]) = \min_{i=1,\ldots,M} \mathrm{D}_2(\mathbf{x}, \mathbf{m}_i[n])\right\}$

   *And **x**\\$R_j[n]$ are the training samples that fall in region R_j [n]*
3. *Compute $C_{lj}[n]$ (majority class of training samples that fall in $R_j[n]$) and $C_{rj}[n]$ (second majority class of training samples that fall in $R_j[n]$)*
4. *Update C[n] with the following equation only if* $N_{R_j,\widehat{C}_{lj}} \neq N_{R_j,\widehat{C}_{rj}}$ *:*

$$\mathbf{m_j}[n+1] = \left. \frac{\sum_{i=1}^{N_{R_j,\widehat{C}_{lj}}} \mathbf{x_i} \middle| \mathrm{R}_j, \widehat{\mathrm{C}}_{lj} - \sum_{u=1}^{N_{R_j,\widehat{C}_{rj}}} \mathbf{x_u} \middle| \mathrm{R}_j, \widehat{\mathrm{C}}_{rj}}{\mathrm{N}_{\mathrm{R}_j,\widehat{\mathrm{C}}_{lj}} - \mathrm{N}_{\mathrm{R}_j,\widehat{\mathrm{C}}_{rj}}} \right|_{\mathbf{C} = \mathbf{C}[n]}$$

5. *n=n+1*
6. *If (n> rlen) then goto 2*
7. *End*

3.2. Step 2: Assignation of Labels

After vector quantization is done, the class label of each codevector must be computed. We propose this labelling schema:

***Input: T** (Training set), **C** (Unlabeled codebook)*

1. *Compute* $T_{R_j} = \{\mathbf{x_i} | R_j, \textit{class label}(\mathbf{x_i} | R_j)\}$

   Where $\mathrm{R}_j = \left\{\mathbf{x} \middle| \mathrm{D}_2(\mathbf{x}, \mathbf{m_j}) = \min_{i=1,\ldots,M} \mathrm{D}_2(\mathbf{x}, \mathbf{m}_i)\right\}$

   *And **x**\\R_j are the training samples that fall in region R_j*
2. *Class label {**m**$_i$}= majority class of data samples in* T_{R_j} *(If the two major classes have the same number of training samples, choose one of them randomly)*

4. Generalization properties of the BLVQ algorithm

4.1. The effect of Finite resources

Given a powerful enough hypothesis space, e.g. a sufficient number of codevectors, one could expect to solve any classification problem since Bayesian borders can be approximated by a series of (locally defined) hyperplanes that are formed with two nearest codevectors of different classes. Practically, since we deal with finite resources and imperfect optimization procedures, generalization performance is degraded by three main factors:

1. A first cause of degradation is due to the fact that we use I_2[VQ] instead of I_1[VQ] to derive our algorithm. Since I_1[VQ]≈I_2[VQ] when codebook size grows indefinitely, this source of error reflects the fact we deal with a finite number of parameters (a finite codebook size M) to approximate the Bayes classifier. In the case of $M\rightarrow\infty$, large-sample results (Cover & Hart, 1967) of nearest neighbor classifiers bound the best misclassification rate of our overall system by $E_b \le E_{1-NN} \le E_b\left(2-\left(CL/\left(CL-1\right)\right)E_b\right)$ where E_b denote the misclassification rate of the Bayes classifier.
2. Another source of degradation comes from the fact that we search, during learning, a minimum of I_{emp2}[VQ], an empirical estimator of I_2[VQ] constructed using a finite set of observations (of size N). General theorems (Vapnik, 1982) bound the error between the empirical functional and the expected functional and thus the error between the minimum point of I_{emp2}[VQ] and I_2[VQ]. This error decreases as N grows and increases if the capacity of the learning machine grows. In this case, the number of parameters of the codebook (Mxk) can approximate the capacity of our system. Thus this kind of error is in conflict with the above error since if we increase M, I_2[VQ] is closer to I_1[VQ] but, at the same time, the distance between I_{emp2}[VQ] and I_2[VQ] increases.
3. Finally, the third source of error is caused by the optimization procedure since it searches a local minimum of I_{emp2}[VQ] instead of a global minimum. To guarantee that the iterative algorithm convergence to a global minimum point of I_{emp2}[VQ] a good initialization procedure is needed.

4.2. Relation of the equilibrium points of the BLVQ with the classification accuracy.

In nearest neighbour classification, the input pattern is assigned to the class of its nearest prototype. If we use BLVQ, these prototypes are computed from minimising I_{emp2}. What is then the classification accuracy that we can expect from the 1-NN classifier that uses these prototypes? If BLVQ minimises the classification error in the training set then, according to (Devroye et al., 1996) §19, the generalisation error could be bounded for the resulting 1-NN classifier. From differential calculus (Apostol, 1967), we know if $\left\{\mathbf{m}_j^*,\ j=1,\ldots,M\right\}$ is a minimum point of the function I_{emp2} (and a equilibrium point of BLVQ) then the eigenvalues of its hessian matrix evaluated at this point are all positive. In consonance with equations 13 and 14, the equilibrium points of BLVQ induces a 1-NN classifier whose Voronoi regions {Rj, j=1,...,M} force the training data that fall in them to the following constraint: $N_{Rj,\hat{C}lj}>N_{Rj,\hat{C}rj}$ where $N_{Rj,\hat{C}lj}$ is the number of data samples that fall in Rj and belong to the majority class in Rj $\hat{C}_{lj}$ and $N_{Rj,\hat{C}rj}$ is the number of data samples that fall in Rj and belong to the second majority class $\hat{C}_{rj}$. Hence, the equilibrium points do not guarantee a minimisation of the training error. However if we consider classification as a problem of estimating Bayes borders, the attractors of BLVQ are placed to induce hyperplanes that only use training data of those two classes which are majority in Voronoi regions and, accordingly, presumably affect the Bayes decision boundaries.

5. Experimental Results

In this section, we compare Kohonen's LVQX algorithms (where X is 1, 2 and 3) and our proposal using an artificial problem due to the fact that comparative studies on real-life data tend to be less informative than those based on artificial data. Since underlying class densities are unknown and generally we deal with a particular data set (one random sample) drawn from these probability distributions, it is difficult to determine the causes that produces performance differences between methods. One might expect that the classification performance of our procedure was better than LVQ algorithms in problems where several class densities are overlapped since estimation of C_{lj} and C_{rj} in LVQ algorithms is worse done. Thus, we study one artificial problem that reflects this behaviour.

5.1 The Artificial problem

In the proposed artificial problem (g3c) three classes moderately overlap two-on-two. Data is not homogeneous and is generated from two (equally probable) normal distributions for each class (see figure 2).

5.1.1 Simulations

Ten independent training, test and validation sets (of the same size, N_{g3c}=1200) are produced from the g3c problem. Then one hundred runs for the classification problem, training set, learning algorithm and different codebook sizes (6, 12, 24, 48 and 96) have been made. Error rates for each codebook size are computed over a total of 1000N classifications.

We have initialized the codebooks using LVQ_PAK's eveninit program (Kohonen et al., 1995). Then we have applied every learning algorithm until classification error in validation set increases or stands. (We have monitored this error every 2 epochs. In the case of LVQX algorithms, every time we monitored the error, we restart the value of the step size α). Optimal parameters of LVQ algorithms have been estimated using the validation set.

5.1.2 Results

The artificial classification problem was chosen to reflect the deficiencies of LVQ algorithms. Since their on-line estimation of C_{lj} and C_{rj} is worse than our batch estimation, one could expect better performance of the BLVQ algorithm when classes overlap. Empirical experiments confirm this hypothesis. In figure 3a. we can see the average test misclassification rate that have been computed over 1000N samples. Our procedure always outperforms LVQ algorithms. The increase of classification performance was 0.54% to 1.62 %, 0.35% to 1.52 % and 0.63% to 3% in comparison with LVQ1, LVQ2 and LVQ3 respectively.

Simulations also reflect better convergence speed of BLVQ. Figure 3b. shows the average speedup in the execution time of the BLVQ algorithm over Kohonen's LVQ algorithms. This speedup, denoted by Sx (where x is 1, 2, and 3), is computed with a division between the training time of LVQx by the training time of BLVQ. Our algorithm (BLVQ) is in average: 23.4 to 9.7 times faster than LVQ1, 24.5 to 11.5 times faster than LVQ2 and 26.7 to 12 times faster than LVQ3. This happens due to:

1) LVQ algorithms are on-line gradient descent algorithms, so they exhibit poor convergence because they do not benefit from Newton's effect as our algorithm does.
2) BLVQ employs fewer operations than LVQ algorithms to make one pass through data (see section 6).

6. Discussion

6.1. BLVQ-based NN Classification.

The BLVQ algorithm is divided in two main steps. The first step consists in performing a particular kind of supervised clustering to obtain a codebook of unlabelled prototypes. The interest of performing this supervised clustering process is that the algorithm place prototypes in regions where training data is with the exception of regions where the density of training data belonging to different classes is the same. (If the training set size were large enough, these regions would agree with Bayes borders). The second step executes a simple schema labelling algorithm that assign class labels to them based on a majority vote among the training data that falls in the Voronoi cells. Since some of the prototypes computed with BLVQ could be arbitrary close to Bayes borders then the BLQ-based Euclidean 1-NN classifier might achieve good classification accuracy. However, as we have shown in section 4.2, the equilibrium points of BLVQ do not guarantee a minimisation of the classification error in the training set. Consequently we cannot use theoretical results of method (Devroye et al., 1996) §19 that bound the generalisation error (e.g. the expected differences between the minimum achieved in the training set and the real). However, since the BLVQ-based 1-NN classifier is a data-dependent partitioning method, the consistency of this classifier (that is, its convergence to the Bayes classifier as M, N$\rightarrow\infty$) could be studied using the mathematical tools developed in (Devroye et al., 1996) §21. Other clustering algorithm like the K-means (Gerscho & Gray, 1992), followed by a labelling schema, allow that the resulting 1-NN classifier was consistent (Devroye et al., 1996) §21.5. Nevertheless, it remains as an open question if BLVQ can ensure it too.

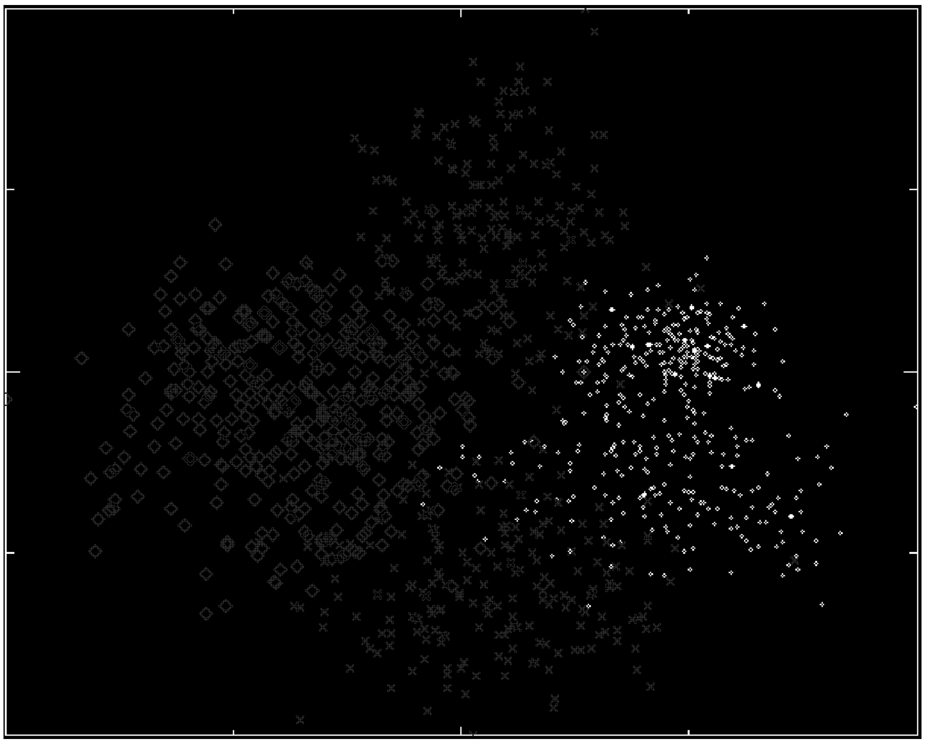


Fig.2. Simulated data used in experiments.

6.2. Kohonen's LVQ algorithms vs. BLVQ.

Kohonen's LVQ algorithms are on-line learning algorithms that tend to convergence to the minimum points of I_{emp2} [VQ] (equation 15). This is notoriously evident in the LVQ1 algorithm since it exactly converges to equation 15 for binary classification (CL=2) (see e.g.(LaVigna, 1990)). On the other hand, LVQ2 and LVQ3 are heuristic modifications of LVQ1 that have been designed with the idea of estimating each C_{lj} and C_{rj} on-line and their exact statistical convergence seems very difficult to precise (Kohonen, 1996). By contrast, BLVQ uses the whole training data to estimate $\{(C_{lj},C_{rj}),\ j=1...M\}$ so these estimates are more robust. Besides, as we have just mentioned above, the minimum points of BLVQ are clearly defined and coincide with those optimal points that Kohonen had in mind when he designed his LVQ algorithms (Kohonen, 1996).

The BLVQ algorithm employs the Newton optimization method to minimise its loss function (I_{emp2}). Instead, Kohonen's LVQ algorithms uses the gradient descent method (at least in LVQ1 (LaVigna, 1990) and LVQ2 (Bottou, 1998)). Consequently, our algorithm converges faster than LVQ algorithms (e.g. in fewer epochs). Furthermore, the execution of our algorithm does not involve more operations than LVQ algorithms. The assignation of the whole training data into one of the Voronoi regions (step 1.2) is also done in one epoch of LVQ algorithms. (One epoch of LVQ algorithms is equivalent to an iteration of the BLVQ). It happens the same with the determination of class label's training data. However, the estimation of $C_{lj}[n]$ and $C_{rj}[n]$ (step 1.3) is only performed in our algorithm but it is a mere counting of class labels. The most remarkable difference between BLVQ and LVQ are steps 1.5 and 2. Step 2 can be considered as a computational effort of one epoch more, since data assignation to Voronoi regions is a time-consuming operation. However, step 1.5 requires fewer operations than the updates of one epoch of LVQ algorithms. In table 1, we display the number of operations computed in one epoch of the simplest LVQ algorithm (LVQ1) and in step 1.5 of the BLVQ algorithm. (Note that the number of operations in one epoch of LVQ2 and LVQ3 easily exceeds LVQ1's since they can update two codevectors for each step.) Since $M<<N$, BLVQ reduces the number of sums by more than a half and the number of multiplication by a factor N/M. Besides, another parameter determines the differences in the training time: the number of epochs to reach a local minimum of the validation error that is reduced in BLVQ since it employs the Newton's optimization method. Hence, BLVQ decreases the training time in comparison with LVQ algorithm, due to 1) the Newton's effect in its dynamics and 2) the reduced number of operations that are needed to update the codevectors.

Algorithm	Number of sums	Number of multipl./divisions
LVQ1	2Nk	Nk
BLVQ	<(N+M)k	Mk

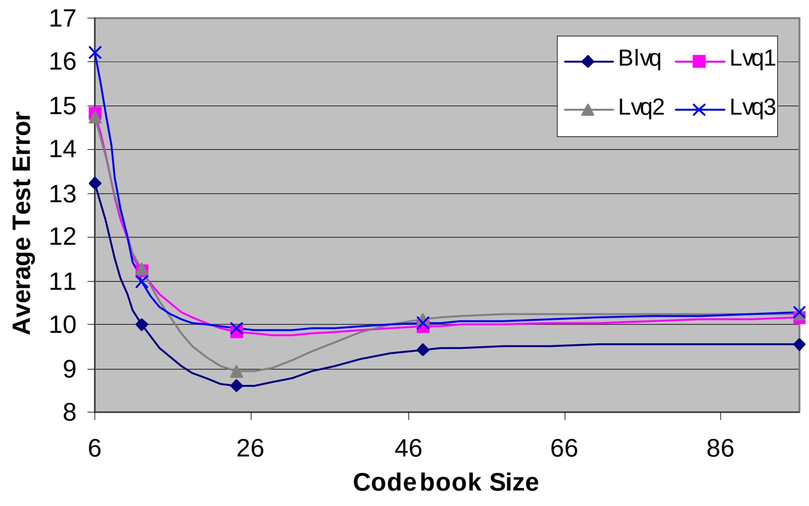
Table 1. Number of operations computed in one epoch of the LVQ1 algorithm and in step 1.5 of the BLVQ algorithm. (N is the number of training samples, M is the number of codevectors and k is the input space dimension).

7. Conclusion

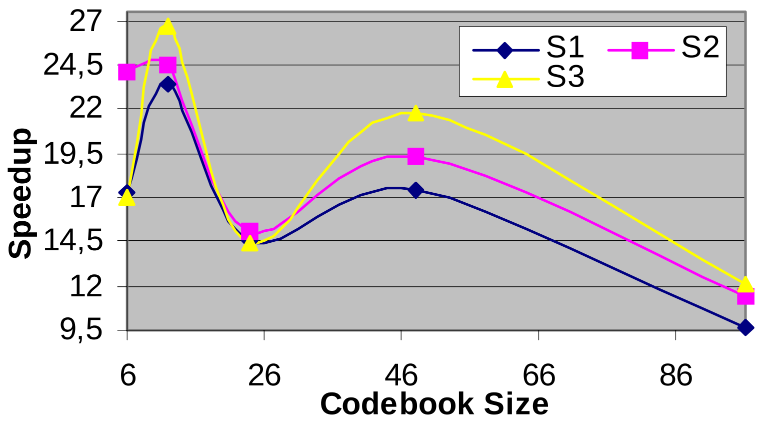
A batch learning procedure to design condensed nearest neighbour classifiers has been introduced. It tends to perform vector quantization (using a D_q metric) over a probability density function that has zero points at Bayes borders. If we choose the Euclidean distance as a metric and perform Newton's optimization over I_{emp2} then the resulting algorithm, called BLVQ, resembles a modified batch K-means algorithm that takes into account class densities. BLVQ shares several properties with batch K-means as their statistical convergence to (class) centroids in a superlinear way.

As other learning systems, three main factors contribute to the generalization error of BLVQ. First cause comes from the fact that we deal with finite codebook sizes. Second source of error arises from minimizing an estimation the expected loss function I_2 constructed from finite observations (of size N). And finally another error stems from searching a local minimum point of I_{emp2} instead of a global minimum point. The first two factors are mutually dependent so a balance between the codebook size M and the training set size N must be done to ensure good generalization.

Experimental results using a simulated classification problem shows the potential of the BLVQ algorithm since it achieves an increase of 0.35% to 3% in the classification accuracy and a speedup in the training time of 9.7 to 26.7 times.



a)



b)

Fig.3. Experimental Results: a) average test error computed over 1200000 test samples and b) average speedup in the training time of the BLVQ algorithm over Kohonen's LVQ algorithms (Sx denotes the speedup over the LVQx algorithm where x is 1,2, and 3).

Empirical evaluation of BLVQ and Kohonen's LVQ algorithms has been presented.

References

Apostol, T. M. (1967). Calculus, Multi-variable Calculus and Linear Algebra with Applications to Differential Equations and Probability, Waltham, MA: Blaisdell Publishing Company.

Bottou, L. (1998). Online Learning and Stochastic Approximation. David Saal (Ed.). Online Learning and Neural Networks. Cambridge, UK: Cambridge University Press.

Cover, T.M. & Hart, P.E. (1967). Nearest Neighbor Pattern Classification, IEEE Transactions on Information Theory, 13, 21-27.

Devroye, L., Györfi, L. & Lugosi, G. (1996). A Probabilistic Theory of Pattern Recognition, Berlin: Springer-Verlag.

Gersho, A. & Gray, R. M. (1992). Vector Quantization and Signal Compression, Boston, MA: Kluwer Academic Publishers.

Hestenes, M. (1980). Conjugate Direction Methods in Optimization, Berlin, New York: Springer-Verlag.

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., & Torkkola, K. (1995). LVQ_PAK. The Learning Vector Quantization Program Package. Version 3.1, Helsinki: Helsinki University of Technology, Laboratory of Computer and Information Science.

Kohonen, T. (1996). Self-organizing Maps, 2nd Edition, Berlin: Springer-Verlag.

Lavigna, A. (1990). Nonparametric classification using learning vector quantization. Ph. D. Dissertation, University of Maryland.

Vapnik, V. (1982). Estimation of Dependencies based on Empirical Data, New York: Springer-Verlag.