Approximation of the Gradient of the Error Probability for Vector Quantizers

Claudia Diamantini, Laura Genga, and Domenico Potena

Dipartimento di Ingegneria dell'Informazione
Università Politecnica delle Marche - Ancona
{diamantini, genga, potena}@dii.univpm.it

Abstract. Vector Quantizers (VQ) can be exploited for classification. In particular the gradient of the error probability performed by a VQ with respect to the position of its code vectors can be formally derived, hence the optimum VQ can be theoretically found. Unfortunately, this equation is of limited use in practice, since it relies on the knowledge of the class conditional probability distributions. In order to apply the method to real problems where distributions are unknown, a stochastic approximation has been previously proposed to derive a practical learning algorithm. In this paper we relax some of the assumptions underlying the original proposal and study the advantages of the resulting algorithm by both synthetic and real case studies.

1 Introduction

Vector Quantization is the process of mapping continuous or discrete vectors produced by a source, into a finite set of symbols, called the codebook, that can be transmitted or stored using a finite number of bits [1]. Because of its strong mathematical background, accompanied with simplicity and easy of use, Vector Quantization finds application in diverse domains like signal detection, image and video compression, image segmentation, face recognition, speech recognition and clustering, among others. Since introduction in the '70s, researchers continue to study mathematical properties and variants of Vector Quantization, its application to novel data structures, domains and problems [2,3] and to develop learning algorithms for the effective design of the codebook [4,5,6], or to improve efficiency [7]. In the application of Vector Quantization to the classification problem, most of the techniques adopt loss functions different from the misclassification risk to define the learning algorithm, the main reason being that the traditional empirical estimate of the risk is not differentiable. An exception is [8], where a continuous stochastic estimate is exploited to derive the gradient with respect to the codebook. However, in order to be applicable, the general theoretical form of the gradient has been approximated, giving rise to a simple and cheap algorithm called BVQ. This algorithm compares favorably with state of the

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

art techniques [9], but it shows some limitations on high-dimensional spaces and on discrete data. Arguing that this behavior is related to the bias introduced by the approximation when data are sparse, in this paper we present a variant of the algorithm that exploits a finer approximation, improving performance.

In the next section we first present the details about Vector Quantizers architectures used for classification and describe the essential steps that lead to the derivation of the BVQ algorithm. Then the new approximation and solutions to deal with technical issues related to it are presented. In section 3 we show experiments on artificial and real datasets that analyze the behavior and performances of the two algorithms. Finally in Section 4 we draw some conclusions and future work.

2 Bayes Vector Quantizer

This section is devoted to illustrate the theoretical principles underlying the Bayes Vector Quantizer (BVQ) algorithm, a stochastic gradient descent algorithm aimed at minimizing the average misclassification risk performed by a Labeled Vector Quantizer (LVQ). In particular, in the following we start describing the basics of statistical decision theory.

Let boldface characters denote random variables, and let (**x**, **c**) be a random variable pair taking values from $\mathbb{R}^n \times C$, where the continuous random vector **x** is the observation (or feature) vector, while the discrete random variable $\mathbf{c} \in C = \{c_1, c_2, \ldots, c_C\}$ is its class. Classes are statistically characterized by conditional probability density functions (cpdf) $p_{\mathbf{x}|\mathbf{c}}(x|c_i)$, measuring the probability that $\mathbf{x} = x$ given that the class observed is $\mathbf{c} = c_i$. Also, a priori probabilities $P_{\mathbf{c}}(c_i)$, i =1, …, C are given. From cpdf, a-posteriori probabilities class can be derived by the Bayes theorem:

$$P_{\mathbf{c}|\mathbf{x}}(c_i \mid x) = \frac{p_{\mathbf{x}|\mathbf{c}}(x \mid c_i) \cdot P_{\mathbf{c}}(c_i)}{p_{\mathbf{x}}(x)} \tag{1}$$

where $p_{\mathbf{x}}(x)$ is the probability density function of the random vector **x**.

In order to ease notation, boldface subscripts will be dropped when this will not make confusion among random variables.

On the basis of the a-posteriori probabilities in (1), it is possible to introduce a *decision rule* (or *classification rule*), that is a mapping $\Phi(\mathbf{x})$: $\mathbb{R}^n \rightarrow C$, expressing the decision taken to classify the sample **x** being observed. A sample **x** is assigned to the class c_i, for which $p(x|c_i)$ is maximum. The decision rule partitions the observation space into as many decision regions as the classes in C. To evaluate the quality of this decision rule we can resort to the average misclassification risk (or, in short, average risk), that is defined as

$$\int R(\Phi(x) \mid x) p(x) dVx \tag{2}$$

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

where $R(\Phi(x)|x)$ is defined as *conditional risk*, that is the risk, or cost, of choosing a given class following the analysis of a given sample, and is expressed by

$$R(c_i \mid x) = \sum_{j=1}^{c} b(c_j \mapsto c_i) P(c_j \mid x) \tag{3}$$

where $b(c_j \mapsto c_i)$ are the elements of a cost matrix B, representing the cost of deciding in favor of class c_i when the true class is c_j. If $b(c_j \mapsto c_i) = 1$ for i ≠ j and $b(c_j \mapsto c_i) = 0$ for i = j, then average risk turns to the well-known 0-1 loss, or error probability.

The *Bayes rule* is the classification rule Φ_B that minimizes the average risk.

Before describing the BVQ algorithm, we briefly introduce the main concepts of the model underlying it, namely Labeled Vector Quantizer. A *nearest neighbor Vector Quantizer with Euclidean distance* (VQ) is a mapping $\Omega: \mathbb{R}^n \longrightarrow \mathcal{M}$ where $\mathcal{M}=\{m_1, m_2,..., m_M\}$ is called *codebook* and vectors m_i are called *code vectors*. The relation Ω partitions the $\mathbb{R}^n$ space into M Voronoi regions, where the i-th region is defined as

$$V_i = \left\{ x \in \mathbb{R}^n : \|x - m_i\|^2 \leq \left\|x - m_j\right\|^2 \forall\, j \neq i \right\} \tag{4}$$

The set of aforementioned regions defines the n-dimensional Voronoi diagram of the codebook $\mathcal{M}$.

A *Labeled Vector Quantizer (LVQ)* is a VQ equipped by a further mapping which assigns a label from *C* to each code vector. The decision border of an LVQ is piecewise linear. Having a classification rule based on a LVQ, the equation (2) becomes:

$$R(\Lambda(\Omega)) = \sum_{j=1}^{C} \sum_{i=1}^{M} b(c_j \mapsto l_i) \int_{V_i} P(c_j \mid x) p(x) dV_x \tag{5}$$

where $l_i \in C$ is the label of the code vector m_i. Note that the average risk depends on Voronoi regions, which in turn depends on the mutual position of code vectors; thus, it is possible to modify the average risk by modifying their positions. A principled way to do this is to adopt gradient descent techniques:

$$\begin{aligned} & m_i^{(k+1)} = m_i^{(k)} - \gamma^{(k)} \nabla_i R^{(k)}(\Phi^{(k)}) \\ & i = 1,2,\ldots,M; \quad k = 0,1,.... \end{aligned} \tag{6}$$

where $m_i^{(k)}$ is the code vector m_i at the k-th iteration of the algorithm, $\gamma^{(k)}$ is the step size and $\nabla_i R(\Omega)$ is the i-th component of the gradient of the average risk. At each iteration of the algorithm the code vectors are updated (and hence the decisional rule) on the basis of the k-th measurement of the average risk gradient.

To compute the gradient, it is important to note that, when *p(x)* is smooth, the average risk is differentiable with respect to the position of code vectors, hence the analyt-

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

ical form of the gradient can be obtained [8]. The gradient with respect to the code vector m_i takes the form:

$$\nabla_i R(\Omega) = \sum_{j=1}^{C} \sum_{q=q_1}^{q_N} \frac{b(c_j \mapsto l_q) - b(c_j \mapsto l_i)}{\|m_i - m_q\|} \times \int_{S_{i,q}} (m_i - x) p(c_j \mid x) p(x) dS_x \quad (7)$$

where $q_1 \ldots q_n$ are the indexes of code vectors that are adjacent to m_i, and $S_{i,q}$ is the decisional surface (i.e. a (n-1) dimensional hyperplane) separating Voronoi regions of m_i and m_q. In order to obtain the gradient when class distribution are unknown (as usual in real classification problems), we refer to the Parzen estimate [10, Chap. 6]. Hence the gradient becomes:

$$\nabla_i \boldsymbol{R}(\Omega) = \sum_{q=q_1}^{q_N} \frac{b(\boldsymbol{h} \mapsto l_q) - b(\boldsymbol{h} \mapsto l_i)}{\|m_i - m_q\|} \times \int_{S_{i,q}} (m_i - x) w(x - \boldsymbol{z}) dS_x \quad (8)$$

where $w(x)$ is the kernel or Parzen window, $\mathbf{z}$ is a sample randomly chosen from the training set and $\mathbf{h}$ is its class label. The derivation of the gradient formula is not reported due to page limitation. Interested readers can refer to [8].

The Bayes Vector Quantizer algorithm uses a particular form of $w(x)$ that allows a simple and cheap implementation. In particular, we consider a hypercubic window, that is $w(x) = \Delta^{-n}$ over a n-dimensional hypercube of side Δ centered on the origin and $w(x) = 0$ elsewhere. In such a way, the integral in (8) assumes a non-zero value for each point on the decision border which is less than $\Delta/2$ from the projection of $\mathbf{z}$ over the border; in other words $\nabla_i R(\Omega) \neq 0$ for all the pairs m_i, m_q such that $S_{i,q}$ falls into the window centered in $\mathbf{z}$ (i.e. $\exists x \in S_{i,q}: w(x - \boldsymbol{z}) = \Delta^{-n}$). At each iteration of the algorithm, only code vectors in these pairs are updated, while other code vectors remain unchanged.

The main issue is to check whether the piece of decision border $S_{i,q}$ falls in the window. In order to solve this issue, we could compute the n-dimensional Voronoi diagram or its dual, namely the Delaunay diagram [11, 12]. Although the use of these diagrams allows a precise calculation of the gradient, their definitions makes the algorithm highly inefficient. Furthermore, experimental results have not shown a significant improvement in performance with respect to implementations that we will present in this work. Hence, we left this direction in place of approximate solutions. To this end, note that if we consider only the piece of border nearest to $\mathbf{z}$ (i.e. the one related to the first two code vectors nearest to $\mathbf{z}$), the issue of checking whether $S_{i,q}$ falls in the window becomes to check whether the distance between the point $\mathbf{z}$ and the line $S_{i,q}$ is less than $\Delta/2$. Hence, the BVQ2 version has been introduced, where at each iteration only the two nearest code vectors are updated [8].

In this version the formula (8) becomes:

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

$$\begin{cases} \nabla_1 R(\Phi) = \dfrac{(b(\boldsymbol{h} \mapsto l_2) - b(\boldsymbol{h} \mapsto l_1)) \cdot (m_1 - z_{1,2})}{\Delta \| m_1 - m_2 \|} \\ \nabla_2 R(\Phi) = \dfrac{(b(\boldsymbol{h} \mapsto l_1) - b(\boldsymbol{h} \mapsto l_2)) \cdot (m_2 - z_{1,2})}{\Delta \| m_1 - m_2 \|} \end{cases} \tag{9}$$

where $\mathbf{z}_{1,2}$ is the projection of the point $\mathbf{z}$ on the surface $S_{1,2}$ separating the first two code vectors nearest to $\mathbf{z}$, namely m_1 and m_2. The assumption underlying BVQ2 is that the window intersects only one piece of the decisional border at a time. This assumption can be considered true since the unbiasedness of Parzen estimate requires that the span of the window is reduced as the training set grows. BVQ2 has experimentally shown comparable or better results than those obtained by well-known and effective classification algorithms; furthermore, from computational point of view BVQ2 turns out to be an advantageous choice [8, 9].

When the training sample falls close to a vertex of the Voronoi diagram, that is the window intersects more than one piece of the decision border, then the approximation error introduced in (9) becomes not negligible.

In these cases we can improve the classifier performances by updating more than two code vectors, achieving a better approximation of (8). In particular, in this work we introduce BVQ3, a version of the BVQ algorithm which updates up to 3 code vectors at each iteration. In this case, the components of the gradient, for the first three code vectors nearest to $\mathbf{z}$ (m_1, m_2 and m_3 respectively), are:

$$\begin{cases} \nabla_1 R(\Phi) = \dfrac{(b(\boldsymbol{h} \mapsto l_2) - b(\boldsymbol{h} \mapsto l_1)) \cdot (m_1 - z_{1,2})}{2\Delta \| m_1 - m_2 \|} + \dfrac{(b(\boldsymbol{h} \mapsto l_3) - b(\boldsymbol{h} \mapsto l_1)) \cdot (m_1 - z_{1,3})}{2\Delta \| m_1 - m_3 \|} \\ \nabla_2 R(\Phi) = \dfrac{(b(\boldsymbol{h} \mapsto l_1) - b(\boldsymbol{h} \mapsto l_2)) \cdot (m_2 - z_{1,2})}{2\Delta \| m_1 - m_2 \|} + \dfrac{(b(\boldsymbol{h} \mapsto l_3) - b(\boldsymbol{h} \mapsto l_2)) \cdot (m_2 - z_{2,3})}{2\Delta \| m_2 - m_3 \|} \\ \nabla_3 R(\Phi) = \dfrac{(b(\boldsymbol{h} \mapsto l_1) - b(\boldsymbol{h} \mapsto l_3)) \cdot (m_3 - z_{1,3})}{2\Delta \| m_1 - m_3 \|} + \dfrac{(b(\boldsymbol{h} \mapsto l_2) - b(\boldsymbol{h} \mapsto l_3)) \cdot (m_3 - z_{2,3})}{2\Delta \| m_2 - m_3 \|} \end{cases} \tag{10}$$

The pseudo code of the BVQ3 version is shown in Figure 1. Again, we have to check whether the intersection between the decision border at each iteration and the hypercubic window is not null. To this end, first we check whether the piece of decision border formed by the first two code vectors close to $\mathbf{z}$ are in the window (step 4.3). This step is the same as in BVQ2 version. Please note that, if these two code vectors belong to the same class, then the surface dividing their Voronoi regions is not a piece of decision border. In this case, whatever the class $\mathbf{h}$ of $\mathbf{z}$, it turns out that $b(\boldsymbol{h} \mapsto l_1) = b(\boldsymbol{h} \mapsto l_2)$, making zero right sides of (9); and, the two code vectors are de-facto not updated. Hence, in order to speed up the execution avoiding an unnecessary check, (at the step 4.3) we introduced also the check on the classes of m_1 and m_2.

Note also that if the piece of border between m_1 and m_2 does not intersect the window, then surely no other piece of border can intersect it. Therefore, if the check in 4.3 is false, no other check is needed and the algorithm goes to next iteration. Otherwise, we have to check whether pieces of decision border formed by m_3 fall in the window, as in Figure 2.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

To this end, before all, we have to check whether the third code vector forms borders with m_1 or m_2, that is to check whether the Voronoi region of m_3 is adjacent to at least one of the region of m_1 and m_2. This further checking is needed due the fact that the existence of a Voronoi surface between the first and the third code vector or between the second and the third one is not guaranteed, while it always exists between the first two nearest code vectors. In this case, we could again use the analytical form of the Voronoi diagram, but as said above its computation is inefficient.

Let label(**x**) be the function that returns the class of **x**.
1. Set the value of Δ, γ^0, the number of iterations n_{max} and the maximum number of attempts *try_max*;
2. Initialize code vectors m_i.....m_n;
3. Set $flag_{BVQ3} = 0$;
4. For k = 1 to n_{max} do
   1. Randomly pick a training pair ($z^{(k)}$, $h^{(k)}$) from the training set;
   2. Find the first three code vectors m_1, m_2 ,m_3 nearest to $z^{(k)}$, such that
      $$\left\|m_1 - z^{(k)}\right\|^2 \leq \left\|m_2 - z^{(k)}\right\|^2 \leq \left\|m_3 - z^{(k)}\right\|^2$$
   3. If (label(m_1) ≠ label(m_2) and $\left\|z^{(k)} - z^{(k)}_{1,2}\right\| \leq \Delta/2$)
      1. If (label(m_1) ≠ label(m_3) or label(m_2) ≠ label(m_3))
         1. Set *try=0;*
         2. While ($flag_{BVQ3}=0$ and *try* < *try_max*)
            1. Set $d=z^{(k)}$ + *noise;*
            2. *try* = *try* + *1;*
            3. If $w(d - z^{(k)}) \neq 0$
               1. Find the first 3 code vectors q_1,q_2,q_3 nearest to d, such that: $\|q_1 - d\|^2 < \|q_2 - d\|^2 < \|q_3 - d\|^2$
               2. If ($q_1 = m_3$ and ($q_2 = m_1$ or $q_2 = m_2$))
                  1. update m_1, m_2 ,m_3 using the formula in (10);
                  2. $flag_{BVQ3} = 1$;
      2. else update m_1 and m_2 using the formula in (9);

Fig. 1. Pseudo-code of BVQ3 algorithm

Hence, in order to reduce complexity, the adjacency of Voronoi regions is empirically tested according to the following principle: regions V_1 (or V_2) and V_3 are adjacent if a point **d** closed to **z** exists, such that m_3 and m_1 (or m_2) are the two code vectors nearest to **d**. To implement this idea, we add a Gaussian n-dimensional noise to the training sample **z**, and then we find the neighborhood of the new point **d** = **z** + **noise** (see Figure 2). The variance of the noise has to be set to a value smaller than the size of the window. As a matter of fact, values comparable to (or bigger than) Δ would have the effect of altering the learning of BVQ, as if de-facto we extend the size of Δ. Now, if **d** falls in the window (step 4.3.1.2.3) and the adjacency of V_3 and V_1 (or V_2) is established (step 4.3.1.2.3.2), then m_1, m_2 and m_3 are updated according

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

to (10). The checking of adjacency is repeated at most *try_max* times. If the adjacency cannot be established, then only m_1 and m_2 are updated using the formula (9), as in BVQ2.

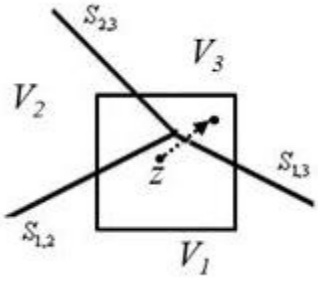

Fig. 2. Example of a point who satisfies the update conditions

We like to highlight that although the proposed BVQ3 version is, of course, more time-consuming than BVQ2, the two version have the same complexity order. As a matter of fact, the main differences are in the extraction of three neighbors instead of two, and in the checking the adjacency of third region. However, whatever the number of nearest code vectors to find, it is needed to order all code vectors. The latter check implies the computation of $^{try\_max}/_{2}$ distances on average. In the next Section, we experimentally show the advantages of BVQ3 over BVQ2.

3 Experiments

This section is devoted to experimentally evaluate the BVQ3 algorithm and to compare it with the BVQ2 version. We first examine their accuracy and ability to approximate the Bayes decision border on a synthetic experiment. Then we show the performances on real-world datasets from the UCI Machine Learning Repository [13].

In both cases the figure of merit to compare the two algorithms is the error probability, hence the cost matrix has been always set to [0 1;1 0]. In particular, for each dataset we show results obtained by using a 10-fold cross validation procedure. The same folds are used both for BVQ2 and BVQ3.

3.1 Experiment with a synthetic dataset

The first experiment has been performed on a two classes 2-D dataset. Each class has been drawn from 2-D independent Gaussian distributions, with the same a-priori probability and zero means. The two classes can be distinguished by means of their covariance matrixes, I for class c1 and 0.01·I for class c2 respectively, where I is the 2×2 identity matrix.

In order to determine the best Δ window size, we refer to the Parzen method to derive the variance that minimizes the error probability on the test set. In particular, by

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

centering a Gaussian function on each training sample, an estimated probability density function (pdf) has been obtained for both class. Based on the estimated pdfs and the classes' a priori probability, test samples have been classified according to the Bayes decision rule. Varying the variance of the Gaussian distribution we are able to obtain different estimate of data distribution and, hence, classification. The estimate that minimizes the classification error is the best approximation of the data distribution, and returns the optimal variance. For this dataset, the best value of the variance is σ^2=2.1·10-5. As in BVQ we use a hypercubic window instead of a Gaussian distribution, the value of delta is given by the following formula: $\Delta = \sqrt{12 \cdot \sigma^2} = 0.0159$.

The number of iteration was fixed at 40000, and several experiments have been performed, by varying the γ^0 value, the number of code vectors and, in the case of BVQ3, the *try_max* value. In particular, we set γ^0 values within the interval [0.00159, 0.0159] with a step of 0.02. Every experiments has been repeated by doubling the number of code vectors, ranging from 4 to 128. The *try_max* parameter has been set to 10,20 and 50.

Table 1 reports best results with respect to the number of code vectors, obtained by varying other parameters. Values are averages resulting by applying a 10-fold cross validation procedure, bold ones are the minimal errors for the two BVQ versions. In both versions, the best accuracy is achieved with 64 code vectors, where the BVQ3 outperforms the result of BVQ2. We can note that the result are close to the Bayesan error, which for this problem is 0.027. In general BVQ3 provides better or at least comparable results than BVQ2.

Table 1. Error probability on *Gaussian*

	Number of code vectors					
	4	8	16	32	64	128
BVQ2	0.0425	0.0295	0.0295	0.0285	**0.0282**	0.029
BVQ3	0.0415	0.0296	0.0291	0.0289	**0.0277**	0.0288

In order to evaluate the ability of approximating the Bayes decision we report the initial (Figure 3.(a)) and the final (Figure 3.(b)-(c)) code vector configurations reached by the two BVQ versions, corresponding to the minimum error values obtained in Table 1. In the Figure, points are code vectors and lines define the Voronoi regions of any code vectors. The bold line is the decision border, while the dotted line represents the optimal (Bayesian) decision border, which for this problem is a circle. Code vectors in the initial configuration are randomly chosen.

Notice that the 3-vector updating determines a better approximation of the border than BVQ2. Furthermore, the distribution of code vectors in BVQ3 is more regular than in the previous version.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

3.2 Experiments with real-world datasets

As mentioned before, in this section we analyze the results of the experiments conducted on 4 datasets chosen from the UCI Machine Learning repository, namely Australian, Liver, Mushroom and Ionosphere datasets. Main characteristics of these datasets are reported in Table 2.

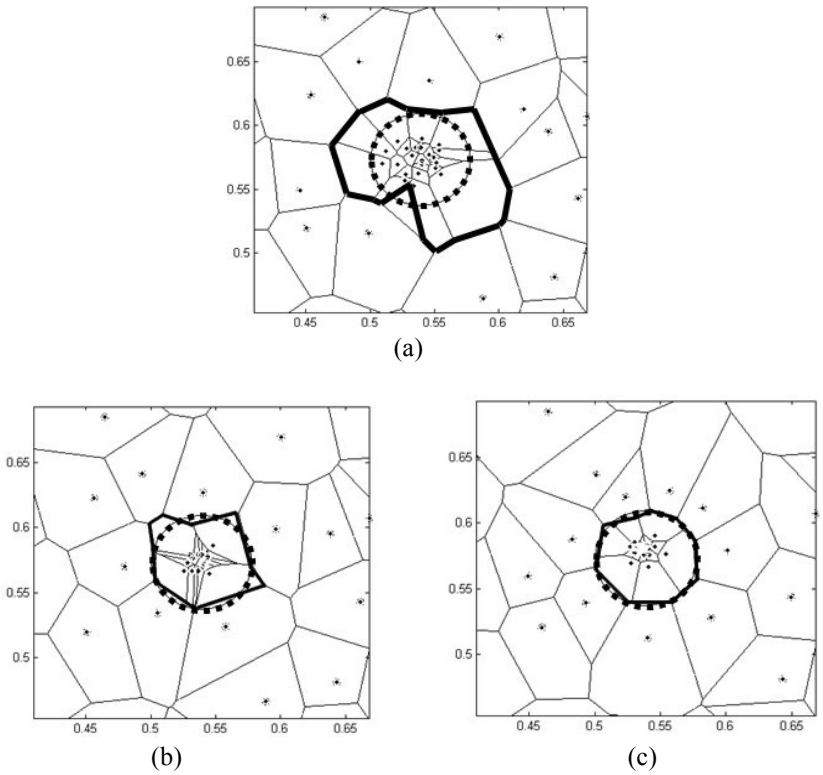

Fig. 3. Example of convergence of the algorithm for *Gaussian*: (a) Initial code-vector layout, (b) final BVQ2 configuration, and (c) final BVQ3 configuration.

As a first step, a data normalization procedure has been performed, with the aim to have all features within the [0, 1] interval, so to give equal importance to each feature during learning. To this end, we have applied the following formula:

$$x_i' = \frac{x_i - \min_i}{\max_i - \min_i} \tag{11}$$

where x_i is the original feature and x_i' is the normalized one, $\min_i$ and $\max_i$ are minimum and maximum values of the feature x_i respectively.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Table 2 reports the dataset characteristics. As for the previous experiment we use the Parzen method to calculate the Δ window size. The number of iterations has been fix to 50000, γ^0 assumes values in [0.1, 1], with a step of 0.1, and the number of code vectors in the set {4, 8, 16, 32, 64, 128}. For BVQ3, we set the *try_max* parameter in the set [10,20,30].

Table 2. Dataset characteristics: number of features (dim), number of samples (N), number of samples in each class (N_1 and N_2). Δ is the window size estimated through the Parzen method.

Dataset	dim	N	N_1	N_2	Δ
Australian	14	690	307	383	0.346
Liver	6	345	145	200	0.154
Mushroom	22	8124	4208	3916	1.897
Ionosphere	34	351	126	225	1.897

Table 3. Error probability on each *real-world dataset*

		Number of code vectors					
Dataset		4	8	16	32	64	128
Australian	BVQ2	0.1406	0.1420	0.1478	0.1420	**0.1391**	0.1522
	BVQ3	0.1478	**0.1319**	0.1449	0.1478	0.1406	0.1391
Liver	BVQ2	0.3205	0.3200	**0.3190**	0.3505	0.3505	0.3700
	BVQ3	0.3267	**0.2919**	0.2981	0.3305	0.3324	0.3348
Mushroom	BVQ2	0.1001	0.0617	0.0327	0.0211	0.0113	**0.0065**
	BVQ3	0.0996	0.0373	0.0201	0.0049	**0.0017**	0.0046
Ionosphere	BVQ2	0.1480	0.1167	0.1082	0.1254	**0.0998**	0.1111
	BVQ3	0.1426	0.1194	0.0944	0.0898	0.0944	**0.0639**

Table 3 summarizes the best results obtained in terms of average error with varying the number of code vectors. The best results for each dataset is in bold.

Results in the table confirm the results of experiments with the synthetic dataset, namely BVQ3 outperforms results of the BVQ2 version. As a matter of fact, BVQ3 returns best results in each dataset. The improvement is particularly significant in Mushroom and Ionosphere, where the gain of BVQ3 over BVQ2 is more than 36%.

Analyzing the results obtained in further detail, it turns out that in general the three-vector version performs better for all *try_max* configurations. It is however not possible to identify the best absolute configuration, as the best minimum error values are achieved for different values of *try_max* in different datasets. In particular, the best value of *try_max* is 10 for the datasets Liver and Ionosphere, 30 for Australian and Mushroom.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Concerning the computing time BVQ3 is, as said before, more time-consuming than BVQ2 : however the difference between the computing times of the two algorithms is not particularly significant, compared to the considerable improvement of classification performance achieved by BVQ3. As an example we reports in Figure 4 the trend of computing times of BVQ2 and BVQ3 calculated on the dataset *Ionosphere*: the continuous line refers to BVQ3, the dotted line to BVQ2.

We can note that two lines have the same trend and are quite close, which corresponds to two similar computing times.

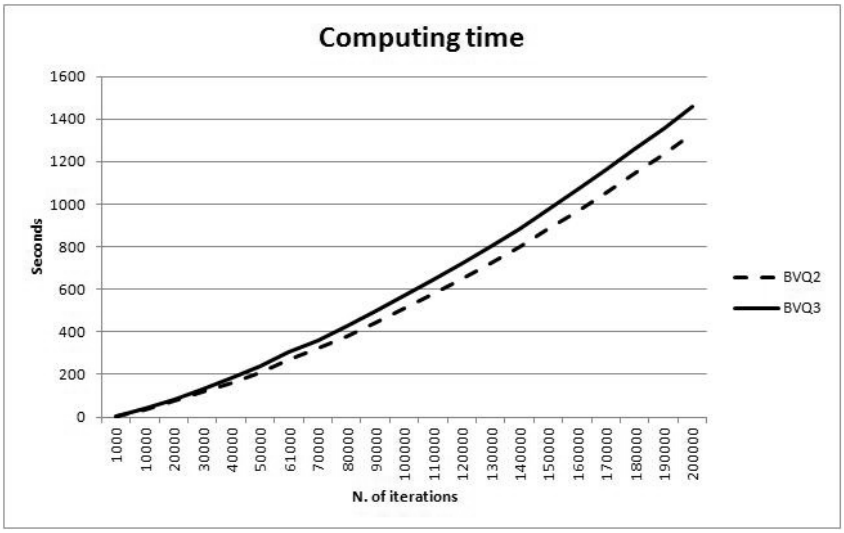

Fig. 4. Comparison between computing time of BVQ3 and BVQ2 for *Ionosphere.*

4 Conclusions

In the paper we presented a new version of the algorithm BVQ, obtained by dropping the hypothesis that at each iteration only a segment of the piece-wise linear border defined by a labeled Vector Quantizer is interested by the updating. Such hypothesis leads to an asymptotically unbiased estimate of the true gradient, since asymptotically the Parzen window involved in the estimate shrinks to a single point. However, in real cases where a finite number of training data is available the hypothesis introduce a bias that worsen the performance of the algorithm. As it is demonstrated in the paper, the recognition of cases where a training data falls close to a vertex of the Voronoi diagram, and the consequent updating of the interested code vector leads to more accurate approximation of the true decision border and to better performance without adding significant complexity.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

We plan to extend the empirical study by performing more experiments on real data, and to compare the new algorithm with other state of the art approaches, also on multi-class, unbalanced datasets.

5 References

1. Gray, R. : Vector quantization. In: Acoustics, Speech, and Signal Processing Magazine, vol. 1, no. 2, (1984).
2. Jain, B.J., Obermayer, K. : Generalized learning graph quantization. In: Proc. of the 8th International Conference on Graph-based Representations in Pattern Recognition, pp. 122-131, Springer-Verlag, Berlin, Heidelberg, (2011).
3. Sanchez, J.; Perronnin, F. : High-dimensional signature compression for large-scale image classification. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, pp.1665-1672, (2011)
4. Tsai, C., Lee, C., Chiang, M., Yang, C., : A fast VQ codebook generation algorithm via pattern reduction. In: Pattern Recognition Letters, Vol. 30, no. 7, pp. 653-660, (2009).
5. Lazebnik, S.; Raginsky, M. : Supervised Learning of Quantizer Codebooks by Information Loss Minimization. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, , vol.31, no.7, pp.1294-1309, (2009)
6. Kastner, M., Hammer, B., Biehl, M., Villmann, T. : Functional Relevance Learning in Generalized Learning Vector Quantization. Neurocomputing, in press. Pre-print available at http://www.cs.rug.nl/biehl/Publications/prepnow.html (2012)
7. Kekre, H. B., Sarode, T.K. : Fast codebook search algorithm for vector quantization using sorting technique. In: Proc. of the International Conference on Advances in Computing, Communication and Control, ACM, New York, NY, (2009)
8. Diamantini,C., Spalvieri, A. : Quantizing for Minimum Average Misclassification Risk. In: IEEE Trans. Neural Netw., vol. 9, no. 1, pp. 174–182 (1998).
9. Diamantini, C., Potena, D. :Bayes Vector Quantizer for Class-Imbalance Problem. In : *Knowledge and Data Engineering, IEEE Transactions on* , vol. 21, no. 5, pp.638-651 (2009).
10. Fukunaga,K.: Introduction to Statistical Pattern Recognition (2nd Edition). Academic Press, San Diego (1990).
11. Aurenhammer, F., Klein, R. : Voronoi Diagrams. In: Handbook of Computational Geometry, Sack,J., Urrutia,J., (eds), cap.5, pp.201-290, Elsevier Science Publishers, Amsterdam, (2000).
12. Okabe,A., Boots ,B., Sugihara, K. : Spatial Tessellations - Concepts and Applications of Voronoi Diagrams(2nd Edition). Wiley, Toronto (2000).
13. Blake,C.L., Hettich, S., Merz, C.J., Newman, D.J.: UCI Repository of Machine Learning Databases, http://kdd.ics.uci.edu/, (1998).

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,