An efficient and scalable data compression approach to classification

Claudia Diamantini
Computer Science Institute, University of Ancona
via Brecce Bianche
60131 Ancona, Italy
diamanti@inform.unian.it

Maurizio Panti
Computer Science Institute, University of Ancona
via Brecce Bianche
60131 Ancona, Italy
panti@inform.unian.it

ABSTRACT

Learning algorithms are effective means of inducing predictive models of a phenomenon starting from a set of instances of the phenomenon itself. However, the impressive growth of the amount of stored data makes scalability of both the learning and classification procedures a compelling requisite for their effective application in data mining tasks, at least as important as accuracy of the induced model. In this paper we show the features of a stochastic gradient algorithm for the minimization of the average misclassification risk performed by a Labeled Vector Quantizer, both in terms of scalability and accuracy. The performance are compared with those of other related algorithms, often adopted in data mining, on both artificial and real data experiments.

1. INTRODUCTION

Database Management Systems give simple and fast access to data, but they provide limited resources to discover high-level information hidden into the data. For instance, they can not give answer to queries like: "give me all the transactions whose likelihood of being fraudulent exceeds 0.75" [6], or "is this client reliable for a loan?". To answer this kind of questions, a predictive model of the domain is needed, that is, a classification rule which allows to classify data into one of a specified set of classes (for instance, "fraudulent", "legal", or "reliable", "unreliable" in the examples above), on the basis of the values of a set of relevant data attributes $x = \{x_1, \ldots, x_n\}$, hereafter called features. The classification rule should be designed as accurate as possible, in such a way that the minimum number of classification errors is made. This problem has been undertaken for a long time in disciplines like statistics, pattern recognition and machine learning, and their achievements form a solid theoretical background also for the developing of methodologies and tools that help in the analysis of, and information extraction from, databases. Non parametric classification methods has been developed in pattern recognition, such as the *nearest neighbor* method [2], whose classification accuracy often turns out to compete in data mining applications with other, more sophisticated, ones [5; 7]. However, this approach requires to process the entire training set in order to classify a new datum, thus its application to large databases turns out to be unpractical. The high cost of non parametric methods was recognized as early as the first studies in pattern recognition, where data reduction techniques were introduced. The aim of data reduction is to select from the whole training set the subset of data which allows the minimum degradation in performance. Hart [9] introduced an algorithm for the nearest neighbor method, based on the observation that these data are exactly those falling near the decision border. This heuristic was in some sense discovered again, in more recent studies in data mining and machine learning [1; 14; 15]. These approaches allows to reduce the cost of classification, but they introduce a cost for running the reduction algorithm, and an accuracy vs efficiency tradeoff. Furthermore, very often such algorithms turn out to be themselves too expensive to be applied to large databases, so an important research topic in data mining is the development of techniques in order to improve scalability [16; 13; 11]. In this paper, we want to bring to the attention of researchers the features of a complementary approach to data reduction, i.e. *data compression*. We introduce a learning algorithm for Vector Quantizer (VQ) architectures [8], based on the principle of average misclassification risk minimization. Although average misclassification risk is universally considered as the right measure to evaluate a classification rule, most of the learning paradigms known in the literature do not use this learning criterion, adopting other related measures instead, such as the well known Mean Square Error used in the back-propagation algorithm (see e.g. [10]) or Structural Risk [17], or heuristics [1; 9; 12]. The proposed algorithm, first presented in [4], is the first complete and general gradient-descent solution to average misclassification risk minimization, based on stochastic approximation theory. Stochastic gradient algorithms are efficient methods for learning [10, §4.6], as they perform local computations based on only one training sample for each iteration. Local computations are simple to implement on parallel architectures, while the use of one sample per iteration allows to keep data on hard disks, with no accuracy vs efficiency tradeoff typical of data reduction techniques. Average misclassification risk minimization guarantees optimal classification performance. These facts ensure the efficiency and effectiveness of the learning process. Furthermore, VQ architectures proves particularly well suited to the classification task, allowing to strongly compress the information contained in the training set. The particular VQ architecture adopted, that of nearest neighbor, allows to design a simple nearest neighbor classification rule based on a very small number of elements with respect to the training set size. These characteristics ensure the efficiency of the classification process. In the present paper, the performance of the method will be demonstrated on both artificial data and real data taken from the UCI ML repository, and compared with those of the classical Nearest Neighbor (1-NN) method, Support Vector Machines (SVM) [17] and the IB2 algorithm of the Instance Based Learning family [1].

2. THE BAYES VECTOR QUANTIZER

In the statistical approaches to classification, data are described by a continuous random vector $\mathbf{x} \in \mathcal{R}^n$ (feature vector) and classes by a discrete random variable $\mathbf{c} \in \mathcal{C} = \{c_1, c_2, \ldots, c_C\}$. For each class c_i, the distribution of data in the feature space is described by the conditional probability density function (cpdf) $p_{\mathbf{x}|\mathbf{c}}(x|c_i)$. The cumulative probability density function of the random vector $\mathbf{x}$ is $p_{\mathbf{x}}(x) = \sum_{i=1}^{C} P_{\mathbf{c}}(c_i) p_{\mathbf{x}|\mathbf{c}}(x|c)$, where $P_{\mathbf{c}}(c_i)$ is the a priori probability of class c_i. The predictive accuracy of a classification rule $\Phi : \mathcal{R}^n \to \mathcal{C}$ is evaluated by the *average misclassification risk*

$$R(\Phi) = \int R(\Phi(x)|x) p_{\mathbf{x}}(x) dV_x, \tag{1}$$

where dV_x denotes the differential volume in the x space and $R(c_j|x)$ is the risk in deciding for class c_j when a particular x is observed. $R(c_j|x)$ is defined as

$$R(c_j|x) = \sum_{i=1}^{C} b(c_i, c_j) P_{\mathbf{c}|\mathbf{x}}(c_i|x). \tag{2}$$

In (2), $b(c_i, c_j) \geq 0$ expresses the cost of an erroneous classification, i.e. the cost of deciding in favor of class c_j when the true class is c_i, with $b(c_i, c_i) = 0 \; \forall i$. If $b(c_i, c_j) = 1 \; \forall i, j, i \neq j$ the average misclassification risk turns to the simpler error probability. $P_{\mathbf{c}|\mathbf{x}}(c_i|x)$ can be derived from $P_{\mathbf{x}|\mathbf{c}}(x|c_i)$ by the Bayes theorem. The development of most of the learning algorithms and non parametric methods for classification starts from the result that the best possible classification of a feature vector consists in mapping it to the class with the minimum conditional risk (2) (Bayes rule):

$$\Phi_B(x) = \min_{c \in \mathcal{C}}^{-1} \{R(c|x)\}, \tag{3}$$

trying to overcome the limits of applicability of this optimal rule, related to the the fact that cpdfs involved in are in general unknown. Thus one of the main efforts is that of obtaining estimates of such functions of x on the basis of the training set. However it is recognized that accurate cpdfs estimation does not necessarily lead to good classification performance [7]. In this paper we take a different approach, based on the observation that a classification rule $\Phi : \mathcal{R}^n \to \mathcal{C}$, being a total and surjective function, induces a partition of the feature space $\mathcal{R}^n$ into C regions $R_1, \ldots, R_C$ (decision regions), where R_i is the set points which are preimages of class c_i in $\mathcal{R}^n$. Starting from this observation, we propose an algorithm to adapt an initial labeled partition, where labels represent classes, towards the optimal partition induced by the Bayes rule. Notice that, in this way, the function of x we try to approximate is directly $\Phi_B(x)$, which, under the hypothesis of piecewise continuity of cpdf, is a piecewise constant function. We encode a labeled partition by a Labeled nearest neighbor Vector Quantizer. A *nearest neighbor Vector Quantizer* of size M is a mapping

$$\Omega : \mathcal{R}^n \to \mathcal{M},$$

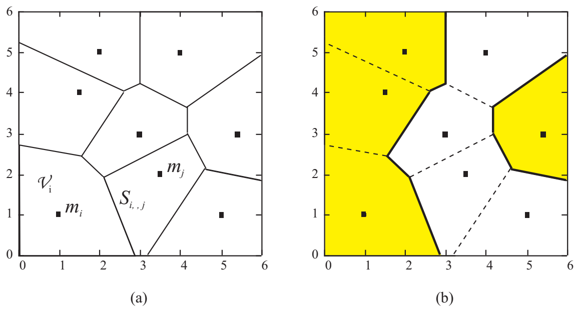

Figure 1: (a) A nearest neighbor VQ of size 8 in $\mathcal{R}^2$ and (b) A possible labeled partition induced by labeling the VQ.

where $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$, $m_i \in \mathcal{R}^n, m_i \neq m_j$, which defines a partition of $\mathcal{R}^n$ into M regions $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_M$, such that

$$\mathcal{V}_i = \{x \in \mathcal{R}^n : d(x, m_i) < d(x, m_j), \; j \neq i\}.$$

$\mathcal{V}_i$ is the *Voronoi region* of *code vector* m_i and d is some distance measure. If the Euclidean measure is adopted, then Voronoi region boundaries turns out to be piecewise linear. In particular, the boundary $\mathcal{S}_{i,j}$ between two regions $\mathcal{V}_i$ and $\mathcal{V}_j$ is a piece of the hyperplane equidistant from m_i and m_j (see Figure 1(a)). In the following we will always refer to nearest neighbor VQs with Euclidean distance. The VQ is extended with a further mapping $\Lambda : \mathcal{M} \to \mathcal{C}$, which assigns a label from $\mathcal{C}$ to each code vector. We will call this extended VQ a Labeled VQ (LVQ). An LVQ can be used to define a classification rule: let $l_i = \Lambda(m_i) \in \mathcal{C}$ denote the label assigned to m_i. The decision taken by the LVQ when x is presented at its input is

$$\Phi_{LVQ}(x) = \Lambda \circ \Omega(x) = l_i, \text{ if } x \in \mathcal{V}_i \tag{4}$$

In practice, the classification is performed by finding in $\mathcal{M}$ the code vector at minimum distance from x, and then by declaring its label. Thus an LVQ implements a simple nearest neighbor rule, and each point of a Voronoi region $\mathcal{V}_i$ is implicitly labeled with the same label of m_i. Figure 1(b) shows an example of labeled partition induced by an LVQ. Notice that, even if each Voronoi region is convex, we can construct non-convex and non connected decision regions as well. Notice also that boundaries between two Voronoi regions with the same label (the dashed lines in Figure 1(b)) do not contribute to the definition of decision region boundaries (decision boundaries). The adoption of the simple Euclidean distance limits us to piecewise linear decision boundaries. With other, more complex distance measures non linear boundaries could be obtained as well. In order to develop an algorithm to find an optimal approximation of the Bayes partition, a crucial observation is that the average risk of Φ_{LVQ} depends only on the labeling function Λ and on the mutual position of code vectors m_i, which determines the form of the integration regions. Thus, keeping the labeling function Λ fixed, and under the continuity hypothesis for cpdfs, average risk is differentiable w.r.t. $\mathcal{M}$. The gradient of $R(\mathcal{M})$ w.r.t. the generic m_i has been derived for the first time in [4], and has the form

$$\nabla_i R(\mathcal{M}) = \sum_{j=1}^{C} \sum_{q=1, q\neq i}^{M} \frac{b(c_j, l_q) - b(c_j, l_i)}{\| m_i - m_q \|} P_{\mathbf{c}}(c_j) \times$$
$$\times \int_{\mathcal{S}_{i,q}} (m_i - x) p_{\mathbf{x}|\mathbf{c}}(x|c_j) dS_x,$$

where dS_x denotes the differential surface in the x space. Almost surprisingly, the variation of the risk w.r.t. code vector m_i depends only on what happens on boundary surfaces between $\mathcal{V}_i$ and each neighbor region $\mathcal{V}_q$ (obviously $\mathcal{S}_{i,q}$ vanishes if m_i and m_q are not neighbors), and only in the case that $b(c_j, l_q) \neq b(c_j, l_i)$. In the case of error probability this means simply that $l_i \neq l_q$, i.e. that the boundary surface between $\mathcal{V}_i$ and $\mathcal{V}_q$ actually represents a part of the decision boundary. Thus, this result formalizes and generalizes the intuition of Hart [9] and of the so called "boundary hunting" methods [1; 17]. The use of a stochastic Parzen estimate for $p_{\mathbf{x}|\mathbf{c}}(x|c_j)$, and some approximations introduced for sake of simplicity, for which we refer the interested readers to [4], leads to a class of stochastic gradient algorithms for the minimization of $R(\mathcal{M})$. Here we consider the algorithm, called Bayes VQ (BVQ), obtained when a uniform window of side Δ is adopted as the Parzen window. Assuming that the labeled training set $\mathcal{L} = \{(t_1, u_1), \ldots, (t_T, u_T)\}$ is given, where $t_i \in \mathcal{R}^n$ is the feature vector and $u_i \in \mathcal{C}$ is its class, at each iteration the algorithm considers a labeled sample randomly picked from the training set. If the sample turns out to fall near the decision boundary, then the position of the two code vectors determining the boundary is updated, moving the code vector with the same label of the sample towards the sample itself and moving away that with a different label. More precisely, the k-th iteration of the BVQ algorithm is:

BVQ Algorithm - k-th iteration

1. randomly pick a training pair $(t^{(k)}, u^{(k)})$ from $\mathcal{L}$;
2. find the code vectors $m_i^{(k)}$ and $m_j^{(k)}$ nearest to $t^{(k)}$;
   /* note: certainly such vectors are neighbors! */
3. $m_t^{(k+1)} = m_t^{(k)}$ *for* $t \neq i, j$;
4. compute $t_{i,j}^{(k)}$, the projection of $t^{(k)}$ on $\mathcal{S}_{i,j}^{(k)}$ [1];
5. if $\| t^{(k)} - t_{i,j}^{(k)} \| \leq \Delta/2$ then

$$m_i^{(k+1)} = m_i^{(k)} - \gamma^{(k)} \frac{b(u^{(k)}, l_j) - b(u^{(k)}, l_i)}{\| m_i - m_j \|} (m_i^{(k)} - t_{i,j}^{(k)})$$
$$m_j^{(k+1)} = m_j^{(k)} + \gamma^{(k)} \frac{b(u^{(k)}, l_j) - b(u^{(k)}, l_i)}{\| m_i - m_j \|} (m_j^{(k)} - t_{i,j}^{(k)})$$

   else $m_t^{(k+1)} = m_t^{(k)}$ *for* $t = i, j$.

Figure 2 illustrates the behavior of BVQ, considering two equiprobable classes (called black (B) and white (W) class) and error probability as the performance measure. In this case, the point P, located where cpdfs coincide is, by definition, the optimal Bayes decision boundary. Figure 2 (a) shows a set of samples of the W and B classes represented by small white and black dots respectively, whose distribution in the feature space follows class statistics. In Figure 2 (b) is depicted an LVQ of size three, with two code vectors labeled as white and one code vector labeled as black, and the induced labeled partition. There, are also indicated the boundaries $\mathcal{S}_{1,2} = \frac{m_1+m_2}{2}$ and $\mathcal{S}_{2,3} = \frac{m_2+m_3}{2}$, the window, represented by the shaded area, centered around $\mathcal{S}_{2,3}$ and

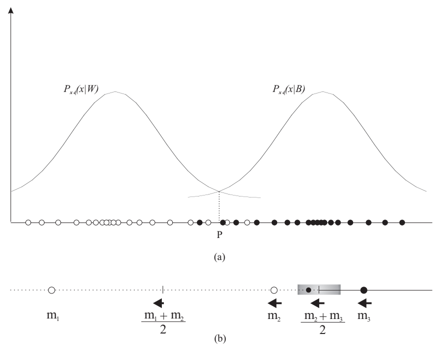

Figure 2: A graphical representation of the adaptation step performed by BVQ.

a training sample falling inside the window. Notice that perturbations of $\mathcal{S}_{1,2}$ do not change the decisions taken in its around, which are always in favor of the white class. In fact, samples falling near $\mathcal{S}_{1,2}$ are not used to update the code vectors. Note however that the updating of m_2 and m_3 indirectly influences also $\mathcal{S}_{1,2}$. Repeated iterations of BVQ algorithm move $\mathcal{S}_{2,3}$ from its initial position toward left, since at the beginning black samples are more frequent than white samples. This drifting continues until a point is reached where the frequency of black and white samples falling inside the window is the same. In the asymptotic case, the number of samples is arbitrary large, and the optimal value of Δ is zero, so this point is exactly the point P where cpdfs coincide. In the finite case, P can be only approached, since $\Delta > 0$. However, more samples we have, smaller values of Δ can be set, more accurate approximations of the Bayes decision boundary can be found. Experiments in [3; 4] show that the BVQ is capable to construct a locally optimal linear approximation of the Bayes decision boundary, and that its classification performance approaches Bayes risk. There, it is also discussed the main differences between BVQ and LVQ2.1, a very similar algorithm by Kohonen [12], showing that the formal derivation of BVQ from the average risk prevents him from the stability problems documented for LVQ2.1.

3. EXPERIMENTAL EVIDENCES

In order to show the advantages of BVQ, we compare it to the nearest neighbor rule (1-NN), SVM [17] and IB2 [1]. We first propose a set of experiments on synthetic data, which best allow to control the training process and to evaluate the results. The performance on real data is also considered, on a set of experiments taken from the UCI Machine Learning repository.

3.1 Synthetic dataset

The aim of this set of experiments is to show both the classification performance of the method and the independence of training and classification costs from the training set size. To this end, we consider the problem of discriminating between two multivariate Gaussian sources, whose mean vectors are zero in all dimensions. The two classes are distin-

[1]Such projection is a function of the two code vectors and $t^{(k)}$ only.

guishable because the two covariance matrices are different. We consider $n = 2$, $Pr(c_1) = Pr(c_2)$, and we assume that both covariance matrices have the form $\sigma^2 I$, I being the 2×2 identity matrix. We set $\sigma_1^2 = 1$, $\sigma_2^2 = 0.01$, so that the Bayes error probability is equal to 0.027 and the Bayes border is a circle centered in the origin with radius 0.305. The experimental setting is based on six training sets of increasing size generated by the same seed, in such a way that each training set contains all the samples of the smaller one plus others. On each training set we trained different LVQs with increasing number of code vectors. The problem of local minima suffered by "greedy" methods afflicts BVQ as well. This can be alleviated by a proper initialization of code vectors, but in this case some domain knowledge should be given. In each experiment, the LVQ was initialized simply by the first M labeled vectors of the training set. The parameter Δ was set to 0.1897, so that the variance of the uniform window is $3*10^{-3}$. This value turns out to be the optimal value in the Parzen method. The number of training iterations was fixed to 40000. As will be clear later, this is a very big number of iterations, which is sufficient but not necessary to the BVQ to reach a good minimum in each experiment. The decreasing law for the step size was set to $\gamma^{(k)} = \gamma^{(0)} j_k^{-0.51}$, where j_k denotes the number of non-null updatings until iteration k and $\gamma^{(0)}$ is an initial value experimentally determined. After the training, the error probability is measured on a new testing set of 10^4 observations, generated by a different seed. The same procedure was repeated for three different seeds and the results averaged. Table 1 reports the average error probabilities.

Table 1: Error probability vs number of code vectors (rows) and size of the training set (columns) for the Bayes VQ.

	100	200	400	800	1600	3200
2	0.296	0.299	0.295	0.298	0.300	0.296
4	0.148	0.120	0.112	0.104	0.104	0.101
8	0.082	0.053	0.038	0.033	0.036	0.033
16	0.072	0.039	0.033	0.028	0.029	0.028

Table 2: Number of effective iterations of the Bayes VQ.

	100	200	400	800	1600	3200
2	2393.3 (1197)	1698.7 (129)	2123.7 (712)	2234.3 (254)	2021.7 (123)	1104.7 (121)
4	1012.3 (721)	1493.7 (567)	2167 (399)	2087 (389)	1984.3 (279)	1288 (323)
8	1058 (542)	1051.3 (373)	1060.3 (624)	1420 (584)	1364.7 (511)	914.7 (247)
16	746.7 (78)	1210 (235)	944.3 (195)	1016.3 (395)	1029.3 (117)	725 (250)

From the table, we can see that the training set size influences performance, as is natural, and that, for each training set size, increasing the VQ size allows to enhance performance. Nevertheless, we can also appreciate that the best VQ size is 16 independently of the training set size. This number of code vectors allows to virtually achieve Bayes error probability from training set size 400 on. Hence, we can conclude that the optimal number of code vectors is independent of the training set size. Table 2 shows for each experiment the average number of non null updatings in the first row and, in the second row in brackets, the number of non null updatings which was approximately necessary to the algorithm to converge. These values represents approximately the number of updatings after which the estimated error probability settles around the final error probability and should be considered for qualitative evaluations only. We can observe a great variability in the number of iterations, even for experiments with the same number of code vectors. This variability depends from the stochastic nature of the method, and in particular from the variability of the initial value of the step size, and from the initial position of code vectors. Nevertheless, observing the table, we can conclude that the method rapidly converge to the optimum with a number of iterations which, substantially, does not depends from the training set size.

For comparison, in Table 3, we report the error probability and size of the SVM, IB2 and 1-NN classifiers. The size is expressed by the number of vectors used in the respective classification rules. These results were obtained by using SVM^{light}[2] and IBL[3] on the three training sets, and averaging the results calculated on the testing set, as before. As we can see, only SVM allows to reach an error performance comparable with BVQ, but with a number of support vectors much greater than the number of code vectors of BVQ. Furthermore, for all the methods the size of the classifier is shown to monotonically increase with the training set size. In order to stress the BVQ perfomance on large data sets,

Table 3: Error probability and size (in brackets) of the SVM, IBL and 1-NN classifiers.

	100	200	400	800	1600	3200
SVM	0.041 (14)	0.034 (21.3)	0.033 (36.3)	0.03 (61)	0.028 (119)	0.028 (246.3)
IB2	0.072 (15.7)	0.073 (19.7)	0.07 (34)	0.075 (57.7)	0.087 (118.7)	0.075 (251.7)
1-NN	0.082 (100)	0.053 (200)	0.038 (400)	0.033 (800)	0.036 (1600)	0.033 (3200)

the algorithm has been run on three training sets of 100.000 samples each, with the parameter Δ set to 0.06. The average error probabilities, estimated on a testing set of 100.000 samples, are reported in Table 4, while in Table 5 the average number of effective iterations is considered. As we can see,

Table 4: Error probability of the Bayes VQ on the 100.000 data set.

2	4	4	16
0.299	0.103	0.035	0.029

Table 5: Number of effective iterations of the Bayes VQ on the 100.000 data set.

2	4	4	16
1233.3 (40)	1472.3 (643.3)	964.3 (582)	624 (200)

the performance of the algorithm does not change on this large size experiment. The LVQ of size 16 is still the best and it still allows to reach an error probability very close to the Bayes error probability. As to the number of effective iterations, they refer to a 40.000 iterations long run. In this way some training sample has not been presented to the algorithm. Although more iterations would allow to present to the algorithm all the samples, the results reported indicates that this is not necessary to reach good error probabilities.

[2] http://www-ai.cs.unidortmund.de/svm_light
[3] http://www.aic.nrl.navy.mil/~aha/software/list.html

This can be explained by observing that the richness of the training set is nevertheless exploited during the training, since samples are randomly chosen from the pool of data, and since this allows a smaller window size.

3.2 Experiments with real data

In order to confirm the previous findings on real data and on higher dimensional spaces, we compare the learning algorithms on a set of experiments from the UCI Machine Learning repository[4] (see Table 6). Table 7 shows the error probabilities achieved by SVM, 1-NN and IB2, together with the size of each classifier in brackets. Since 1-NN does not discard any training sample, the size of this classifier corresponds to the training set size. The results are taken from [15]. They correspond to average measures obtained by a 10-fold cross validation method. Thus, in order to compare the results, the same experimental procedure was adopted for the BVQ. Before applying the BVQ, a normalization of data is performed, in such a way that each feature ranges in the same interval. This is a invertible transformation of data which allows to give equal importance to each vector component during the learning. In Table 8 we report, for each experiment, the optimal value of the parameter Δ and the best error performance achieved by BVQ with LVQs of varying size. Some comments on this table are in order. First, we have to report a phenomenon typical of VQ architectures, called the "dead neuron problem" by Kohonen. In practice, it can happens that a code vector is never used to classify input data. The removal of such code vectors would not modify the error probability for the data at hand. Thus the number of code vectors reported is always greater than or equal to the true number of used code vectors. Second, we can observe a non monotonic trend of error probability vs number of code vectors for the australian, diabetes and liver data sets. This somewhat counter intuitive result can be explained by the intrinsic "jitter" of stochastic methods, which make the system to wander around the optimum, by the dead neuron problem and by the dependency of the result from initialization and labeling of code vectors.

Table 6: The data sets from the UCI Repository used in experiments.

Data set	Size	Dimension
Diabetes	768	8
German	1000	24
Mushroom	8124	22
Australian	690	14
Liver-Disorders	345	6
Ionosphere	351	34

Table 7: Error probability and size of SVM, 1-NN and IB2 classifiers on the UCI repository data sets.

Data set	SVM	1-NN	IB2
Australian	0.1537 (203.9)	0.185 (603)	0.2642 (151.5)
Diabetes	0.2292 (401.7)	0.3048 (691.2)	0.3505 (253.5)
German	0.249 (487)	0.331 (900)	0.388 (338.7)
Ionosphere	0.0543 (167.1)	0.1543 (315.9)	0.1314 (54.7)
Liver	0.3132 (209.7)	0.3765 (310.5)	0.4201 (121.3)
Mushroom	0.0 (437.3)	0.0 (7311)	0.0041 (25.6)

Table 8: Error probability vs number of code vectors for the BVQ on the UCI repository data sets. Values in boldface denote the best error probability for each experiment.

Data sets	2	4	8	16	32
Australian $\Delta = 0.346$	0.1493	0.1478	0.1449	0.1493	**0.1435**
Diabetes $\Delta = 0.49$	0.2396	0.2358	**0.2265**	0.2358	0.2422
German $\Delta = 1.549$	0.291	0.290	0.277	0.270	**0.253**
Ionosphere $\Delta = 1.897$	0.1454	0.1398	0.1343	0.1231	**0.112**
Liver $\Delta = 0.154$	0.3338	0.3219	**0.286**	0.3038	0.3033
Mushroom $\Delta = 1.897$	0.2254	0.2129	0.100	0.0242	**0.0138**

Turning to the comparison of results in Tables 7 and 8, the BVQ almost always reach a better error probability than the other methods, except for the mushroom experiment, where it is the worst. In this experiment, samples are described by purely categorical features, hence cpdfs are not piecewise continuous, so the basic assumption for the applicability of the method is not satisfied. As a consequence, we can report a long number of iterations needed to converge, and a great sensitivity of the algorithm to the initialization of code vectors and to the value of the parameter $\gamma^{(0)}$. On the other hand, the algorithm proves to work nicely if at least some feature turns out to be continuous, as it is the case for the other experiments. Apart from the special mushroom case, BVQ performance are surpassed only by SVM on the ionosphere data set. For this experiment, we failed to find a number of code vectors for which a comparable error probability could be obtained. This fact can be related to the small training set size, especially compared with the dimension of the feature space. Thus, from the perspective of very large databases, this result is not very serious. On the other hand, the advantage of BVQ in memory requirements is striking. As we can see, two code vectors are sufficient to the BVQ to obtain an error probability lower than all the others for the australian data set, they are sufficient to reach a lower error probability than 1-NN in all the experiments and a lower error probability than IB2 in all but the ionosphere experiment. The highest memory requirements for the BVQ are on the german data set, where 32 code vectors are needed to reach an error probability comparable with that of SVM, which needs 487 vectors out of 900 training samples. Furthermore, the artificial experiments presented in the previous section state that the optimal size of the BVQ classifier depends only from the geometry of the problem (number of classes, shape of decision regions), while for the other methods the classifier size depends on the the training set size.
The size of BVQ, IB2 and 1-NN classifiers allows also to directly compare their computational cost, as they all are nearest neighbor methods. For instance, on the german data set the BVQ allows to classify a sample by calculating only 2 distances, against the 900 distance calculation of the 1-NN and the 339 distance calculation of IB2. The decision function of SVM takes a different form. SVM can directly

[4] http://www.ics.uci.edu/~mlearn/MLRepository.html

manage only two class problems[5]. Assuming that class labels are encoded by integers -1 and 1 respectively, SVM decision function is

$$\Phi_{SVM}(x) = sgn[\sum_{i=1}^{S} u_i \alpha_i K(x \cdot s_i) + b], \quad (5)$$

where, $\{(s_1, u_1), \ldots, (s_S, u_S)\} \subseteq \mathcal{L}$ is the set of Support Vectors and S is the size of the classifier. $K(\cdot)$ is either a linear or a non linear kernel (typical kernels are the gaussian and sigmoid functions). If K is a linear kernel, then this decision function requires approximately the same number of multiplications of (4). In fact, we can rearrange the last one as follows

$$\begin{aligned}\Phi_{LVQ}(x) &= \Lambda(\min_{m_i}{}^{-1}\{\| m_i - x \|^2\}) \\ &= \Lambda(\min_{m_i}{}^{-1}\{\| m_i \|^2 - x \cdot m_i\}).\end{aligned}$$

Hence, if we store the square norm of code vectors, which equals the cost of storing the weights α_i in SVM, both the decision functions require a number of inner products equal to the size of the classifier. In the general case, however, the SVM decision function is computationally heavier, since we have to add the cost of the non linear kernels calculus. In the above experiments the kernel adopted is always gaussian, except for the mushroom experiment, where it is linear. Concluding, comparing the size of BVQ and SVM classifiers we can establish an even greater computational advantage of the former over the latter than the advantage observed over 1-NN and IB2.

4. CONCLUSIONS

In the paper we showed that compression capabilities of the VQ architecture can be exploited to find a simple and effective classification rule by a very small number of vectors, when we train it with the stochastic gradient algorithm BVQ. Comparison to other classification techniques have been presented on both artificial and real data sets. The results show the advantage of BVQ over the other methods when only classification performance are considered, and a striking advantage when both classification performance and costs are taken into account. These results put the BVQ algorithm as a promising technique for the induction of predictive models from huge amounts of data, especially in the light of the fact that it performs local computations based on only one sample for each iteration. Local computations are simple to implement on parallel architectures, while the use of one sample per iteration allows to keep data on hard disk, with no accuracy vs efficiency tradeoff typical of data reduction techniques. Furthermore, although in the experiments we concentrate on error probability as the performance measure, the BVQ is a general algorithm for the minimization of average risk. Thus the introduction of cost matrices in the formulation of the classification problem can be supported as well. This feature is important for practical applications, where some classification errors are often considered more serious than others (for instance, evaluating a client reliable for a loan when it is unreliable can be more dangerous for a bank than evaluating a reliable client unreliable). Directions of research include the study of methods to further improve performance, by finding better initialization strategies of code vectors and developing non greedy versions of the algorithm to escape local minima, and the introduction of more general distance measures. Also of interest is the study on the exploitation of geometric characteristics of VQs in order to extract symbolic classification rules from it.

5. ACKNOWLEDGEMENTS

The authors wish to thank the anonymous referees which, with their comments, helped in improving the quality of the manuscript.

6. REFERENCES

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[2] T. M. Cover and P. E. Hart. Nearest neighbor Pattern Classification. *IEEE Trans. on Information Theory*, 13(1):21–27, Jan. 1967.

[3] C. Diamantini and A. Spalvieri. Vector quantization for minimum error probability. In *Proc. Int. Conf. on Artificial Neural Networks*, volume 2, pages 1091–1094, May 1994.

[4] C. Diamantini and A. Spalvieri. Quantizing for Minimum Average Misclassification Risk. *IEEE Trans. on Neural Networks*, 9(1):174–182, Jan. 1998.

[5] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[6] U. M. Fayyad. Editorial. *Data Mining and Knowledge Discovery*, 1(1):5–10, 1997.

[7] J. H. Friedman. On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[8] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[9] P. E. Hart. The condensed nearest neighbor rule. *IEEE Trans. Information Theory*, 4:515–516, May 1968.

[10] S. Haykin. *Neural Networks: A Comprehensive Foundation, 2nd Ed.* Prentice Hall, New Jersey, 1999.

[11] T. Joachims. Making Large-Scale SVM Learning Practical. In B. Scholkopf, C. J. Burges, A. J. Smola, editor, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1999.

[12] T. Kohonen. The self organizing map. *Proc. of the IEEE*, 78(9):1464–1480, Sept. 1990.

[13] F. J. Provost and V. Kolluri. A survey of methods for scaling up inductive learning algorithms. Tech. Rep. ISL-97-3, Intelligent System Lab., Dept. of Comp. Science, Univ. Pittsburg, 1997.

[5]problems with $C > 2$ classes have to be managed by designing C different classifiers, each separating one class from the rest [14]

[14] B. Schoelkopf, C. Burges, and V. Vapnik. Extracting Support Data for a Given Task. In U. Fayyad and R. Uthurusamy, editors, *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[15] N. A. Syed, H. Liu, and K. K. Sung. A Study of Support Vectors on Model Independent Example Selection. In S. Chaudhuri and D. Madigan, editors, *Proc. 5th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 272–276, New York, 1999. ACM Press.

[16] N. A. Syed, H. Liu, and K. K. Sung. Handling Concept Drift in Incremental Learning with Support Vector Machines. In S. Chaudhuri and D. Madigan, editors, *Proc. 5th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 317–321, New York, 1999. ACM Press.

[17] V. Vapnik. *Statistical Learning Theory*. J. Wiley and Sons, New York, 1998.