Regression Learning Vector Quantization

Mihajlo Grbovic
Department of Computer and Information Sciences
Temple University
Philadelphia, USA
e-mail: mihajlo.grbovic@temple.edu

Slobodan Vucetic
Department of Computer and Information Sciences
Temple University
Philadelphia, USA
e-mail: vucetic@ist.temple.edu

***Abstract*— Learning Vector Quantization (LVQ) is a popular class of nearest prototype classifiers for multiclass classification. Learning algorithms from this family are widely used because of their intuitively clear learning process and ease of implementation. In this paper we propose an extension of the LVQ algorithm to regression. Just like the LVQ algorithm, the proposed modification uses a supervised learning procedure to learn the best prototype positions, but unlike LVQ algorithm for classification, it also learns the best prototype target values. This results in the effective partition of the feature space, similar to the one the K-means algorithm would make. Experimental results on benchmark datasets showed that the proposed Regression LVQ algorithm performs better than the nearest prototype competitors that choose prototypes randomly or through K-means clustering, classification LVQ on quantized target values, and similarly to the memory-based Parzen Window and Nearest Neighbor algorithms.**

Keywords-regression, learning vector quantization

I. INTRODUCTION

Learning Vector Quantization (LVQ) [8] is a simple, universal, and efficient classification algorithm. It belongs to a class of prototype-based learning algorithms such as nearest neighbor, Parzen window, kernel perceptron, and support vector machine algorithms. LVQ is defined by a set of P prototypes $\{(m_j, c_j),\ j = 1 \ldots P\}$, where m_j is a K-dimensional vector in the feature space, and c_j is its class label. Given an unlabeled data point x_u, its class label y_u is determined as the class c_q of its nearest prototype m_q,

$$y_u = c_q,\ q = \arg\min_j d(x_u, m_j), \tag{1}$$

where d is a distance measure (e.g. Euclidean).

The training of LVQ starts with placing the prototypes at some initial positions in the feature space. LVQ algorithm then sequentially scans the training data points and updates the prototypes. There are several different LVQ algorithms that deal with prototype updates in different ways [1, 8].

LVQ2 [8] has been shown to achieve consistently good accuracy and is commonly used as a representative of the LVQ algorithms. Given a training data point (x, y), three conditions have to be met for LVQ2 to update its prototypes: 1) Class of the prototype closest to x has to be different from y, 2) Class of the next closest prototype has to be equal to y, and 3) x must satisfy the "window rule" by falling near the hyperplane at the midpoint between the closest (m_j) and the second closest prototype (m_k). These two prototypes are then modified as

$$\begin{aligned} m_j(t+1) &= m_j(t) - \alpha(t)\,(x - m_j(t)) \\ m_k(t+1) &= m_k(t) + \alpha(t)\,(x - m_k(t)), \end{aligned} \tag{2}$$

where t counts how many updates have been made, and $\alpha(t)$ is a monotonically decreasing function of time. Let d_j and d_k be the distances between x and m_j and m_k. Then, the "window rule" is satisfied if $min(d_j / d_k, d_k / d_j) > s$, where s is a constant commonly chosen between 0.4 and 0.8.

Equation (1) corresponds to a hard decision based solely on the nearest-neighbor prototype, as is done in 1-Nearest Neighbor algorithm. While hard decisions are typical of LVQ, several soft LVQ versions have been proposed [16], that predict based on a weighted vote from the near prototypes. Soft versions of LVQ are reminiscent of Parzen window and Support Vector Machines algorithms.

LVQ is traditionally used for classification. In this paper, we propose several LVQ algorithms suitable for regression. We pose Regression LVQ as an optimization problem with two different cost functions. The first cost function leads to Regression LVQ with hard predictions and the second to LVQ with soft predictions. To design an LVQ learning algorithm, we explored two optimization approaches, one being gradient-based and another expectation-maximization- (EM-) based. The gradient-based approach allows both batch and stochastic (online) updates of prototypes, while the EM-based is naturally suited for the batch mode of operation. Finally, we show that a simplification of one of the resulting algorithms leads to a regression variant of the popular LVQ2

There are several main justifications for using LVQ algorithms in regression. One is that it allows a clear mechanism for designing a prototype-based regressor on a fixed prototype budget. Unlike classification, where methods such as condensing [5] have been proposed for nearest neighbor classifiers, there is no clear alternative for building budget-based nearest neighbor and Parzen window algorithms for regression. Another justification is that regression LVQ has a relatively low training cost with linear time and constant memory scaling with training size. This makes it very attractive for the online learning applications. Finally, some regression problems require prototypes for representational purposes.

We should also discuss the difference between soft and hard predictions in prototype-based algorithms. First, hard predictions are suboptimal in the sense that they are a special

case of soft predictions where the whole weight of decision is given to the nearest prototype. It has long been known that soft predictions have a smoothing effect that is useful for approximating regression functions. However, there are several situations where hard predictions are necessary. Two examples are decentralized estimation [12] and meta-learning in distributed data mining [10, 11]. In both applications, instead of sending raw data or predictions to the fusion center, it might be more appropriate to send index of the nearest prototype. This can be useful when there are communication channel and energy constrains in the fusion system, or when there are the data privacy concerns.

II. METHODOLOGY

The starting point in the design of Regression LVQ (RLVQ) algorithms is to introduce probability $p(j|x)$ of assigning observation x to prototype j that is dependent on their (Euclidean) distance. Let us assume that the probability density $p(x)$ of x can be described by a mixture model

$$p(x) = \sum_{j=1}^{P} p(x|j)p(j), \tag{3}$$

where P is the number of components, $p(j)$ is the prior probability that a data point is generated by a particular component and $p(x|j)$ is the conditional probability that component j generates particular data point x.

Let us represent the conditional density function $p(x|j)$ with the normalized exponential form

$$p(x|j) = K(j)\cdot\exp(f(x,m_j)), \tag{4}$$

and consider a Gaussian mixture with $K(j) = (2\pi\sigma_p^2)^{-1/2}$ and $f(x,m_j) = -(x-m_j)^2/2\sigma_p^2$. In this case, component j is completely represented by its mean m_j and standard deviation σ_p. Therefore, we can describe *component j* as *prototype j* and we will follow this convention through the rest of the paper. We should observe that it was assumed that all prototypes have the same standard deviation (*width*) σ_p. Additionally, we will assume that each prototype has the same prior, $p(j) = 1/P$. Given this, using the Bayes' rule we can write the assignment probability as

$$p(j|x) = \frac{\exp(-(x-m_j)^2/2\sigma_p^2)}{\sum_{l=1}^{P}\exp(-(x-m_l)^2/2\sigma_p^2)}. \tag{5}$$

Starting from equation (5), in sections II.*A* and II.*B*, we are proposing two cost functions that naturally lead to hard and soft versions of regression LVQ.

A. Hard RLVQ

To develop cost function for Hard RLVQ we propose the following prediction model: data point x is assigned to a single prototype j^* by chance following the assignment probabilities $p(j|x)$ and its label is predicted as label q_{j^*} of prototype j^*. The expected Mean Squared Error (MSE) of such a model can be expressed as

$$D_1 = \sum_{i=1}^{N}\sum_{j=1}^{P} p(j|x_i)\cdot(y_i - q_j)^2 = \sum_{i=1}^{N}\sum_{j=1}^{P} g_{ij}\cdot e_{ij}, \tag{6}$$

where $p(j|x_i)$ is from equation (5). For the compactness of notation, we defined $g_{ij} \equiv p(j|x_i)$ and $e_{ij} \equiv (y_i - q_j)^2$.

To see why D_1 from (6) is suitable as cost function for Hard RLVQ, consider the case when $\sigma_p \to 0$. Then, the assignment probability $p(j|x_i)$ for the nearest prototype is one, while the assignment probability for other prototypes is zero. This is exactly the MSE of Hard LVQ. The reason why cases when σ_p is nonzero are interesting has to do with the training procedure, as will be described shortly.

The objective of LVQ learning is to estimate the unknown model parameters, namely prototype positions m_j and target values q_j, $j = 1\ldots P$. This is done by minimizing the cost function D_1 with respect to the parameters. To do this we propose to use the gradient descent method. For this, we have to calculate derivatives $\partial D_1/\partial m_k$ and $\partial D_1/\partial q_k$ for $k = 1\ldots P$. Interestingly, $\partial D_1/\partial q_k = 0$ can be expressed in the closed form and, therefore, prototype labels could be updated exactly. The resulting learning rules for Hard RLVQ are

$$\begin{aligned} m_k^{t+1} &= m_k^t - \alpha(t)\sum_{i=1}^{N}(e_{ik} - \sum_{j=1}^{P} g_{ij}e_{ij})\frac{g_{ik}(x_i - m_k^t)}{\sigma_p^2} \\ q_k^{t+1} &= \sum_{i=1}^{N} g_{ik}\cdot y_i \Big/ \sum_{i=1}^{N} g_{ik}, \end{aligned} \tag{7}$$

where t is the iteration number and $\alpha(t)$ is the learning rate. At $t = 0$, the procedure starts by selecting at random (or as the first, in the on-line case) P training points as initial prototypes. Then, prototype labels are calculated as in (7) and the procedure is iterated. Learning is terminated when the loss function D_1 stops improving (e.g. by more than 10^{-5}).

An important issue to be addressed is choice of parameter σ_p. Setting it to a value near zero would hurt the learning process and possibly lead to poor RLVQ model, as observed in the related optimization problems [13, 16]. Therefore, we initially set σ_p to a large value and slowly anneal it towards zero. The update rule can be derived using the gradient descent method $\sigma_p^2(t+1) = \sigma_p^2(t) - \alpha(t)\cdot\partial D/\partial\sigma_p^2$. However, we experimentally determined that this could lead to instabilities in the learning process. A much better approach is to anneal σ_p^2 using a specific schedule. Following the suggestion from [16], the annealing was performed using the schedule $\sigma_p^2(t+1) = \sigma_p^2(0)\cdot\sigma_T/(\sigma_T+t)$, where σ_T is the parameter that controls how fast is the annealing. For batch versions it was set to $\sigma_T = 5$ by default, while for stochastic versions we used $\sigma_T = 5N$.

Hard RLVQ algorithm summarized in (7) is batch-mode – it requires a pass through whole training set to perform a single iteration. It can easily be converted to the stochastic (online) mode that updates model after each training point, as

$$\begin{aligned} m_k^{t+1} &= m_k^t - \alpha(t)\cdot(e_{ik} - \sum_{j=1}^{P} g_{ij}\cdot e_{ij})\cdot g_{ik}\frac{x_i - m_k^t}{\sigma_p^2} \\ q_k^{t+1} &= q_k^t - \alpha(t)\cdot 2g_{ik}(q_k^t - y_i). \end{aligned} \tag{8}$$

We can observe that the main difference is in the way prototype targets are updated – here, we could not use the closed-form update of (7) because it would require a pass through whole training data set, thus defeating the purpose of stochastic updating.

B. Soft RLVQ

To develop cost function for Soft RLVQ, we use the prediction model that assigns data point x to the prototypes probabilistically and predicts based on the weighted average of the prototype labels. Assuming this mixture model, we can express the posterior probability $p(y|x)$ as

$$p(y \mid x) = \sum_{j=1}^{P} p(j \mid x)\, p(y \mid j), \tag{9}$$

that was obtained by observing the conditional independence between x and y given j, $p(y|x, j) = p(y|j)$. For $p(j|x)$ we use the Gaussian distribution from (5). For the probability of generating target y by prototype j, $p(y|j)$, we also assume Gaussian error model with mean $(y - q_j)$ and standard deviation σ_y. The resulting cost function D_2 for Soft RLVQ can be written as the negative log-likelihood

$$\begin{aligned} L_2 &= \prod_{i=1}^{N} P(y_i \mid x_i) = \prod_{i=1}^{N}\sum_{j=1}^{P} g_{ij} \cdot N(y_i - q_j, \sigma_y^2) \\ D_2 &= -\ln L = -\sum_{i=1}^{N} \ln \sum_{j=1}^{P} g_{ij} \cdot N(y_i - q_j, \sigma_y^2), \end{aligned} \tag{10}$$

where, for the compactness of notation, we defined $g_{ij} \equiv p(j|x_i)$ and $e_{ij} \equiv N(y_i - q_j, \sigma_y{}^2)^2$.

Let us first derive the gradient descent rule for minimization of D_2, similar to the stochastic gradient algorithm (8),

$$\begin{aligned} m_k^{t+1} &= m_k^t - \alpha(t) \cdot \frac{\sum_{j=1}^{P} g_{ij} \cdot e_{ij} - e_{ik}}{\sum_{j=1}^{P} g_{ij} \cdot e_{ij}}\, g_{ik}\, \frac{x_i - m_k^t}{\sigma_p^2} \\ q_k^{t+1} &= q_k^t - \alpha(t) \cdot \frac{g_{ik} \cdot e_{ik}}{\sum_{j=1}^{P} g_{ij} \cdot e_{ij}}\, \frac{(y_i - q_k^t)}{\sigma_y^2}. \end{aligned} \tag{11}$$

Comparing to the learning rule for q_k in (8) we see that (11) uses a weighting term that takes into account how well prototype j does relative to the other prototypes. The batch version of update rules in (11) can be obtained using the batch gradient descent in a straightforward manner.

In the proposed Soft LVQ version, we use the same annealing schedules for $\sigma_p{}^2$ and α. The main difference is that we do not wish $\sigma_p{}^2$ to drop to a value near zero, but instead to set it to an appropriate positive value. To achieve this, we resort to using the validation data set and continue to decrease $\sigma_p{}^2$ as long as D_2 value is being improved on the validation data set. Soft RLVQ uses $\sigma_p{}^2$ value achieved at the termination of training.

Compared to Hard RLVQ, Soft RLVQ has $\sigma_y{}^2$ as an additional parameter. In our algorithm, parameter $\sigma_y{}^2$ is set as the mean squared error of the current RLVQ model.

Given an unlabeled data point x_u, its target value y_u is predicted in a soft way, using the gating function, as a weighted average of the prototype target values q_j

$$y_u = \sum_{j=1}^{P} g_{uj} \cdot q_j. \tag{12}$$

As an alternative to the gradient descent algorithm for minimization of D_2 we explored the Expectation-Maximization (EM) approach [3]. The resulting update steps for the algorithm parameters (not shown due to length constraint) are similar but applicable only to batch mode.

C. Regression LVQ 2

Starting from Hard RLVQ and simplifying the assignment probabilities such that only two closest prototypes are consulted each time the prediction is needed, we will demonstrate that the resulting prototype updates closely resemble LVQ 2 learning rule from (2). Given a training data point x_i with label y_i, let us denote the closest prototype as m_k and the second closest prototype as m_j. We will neglect the remaining prototypes. This is acceptable as an approximation of the Hard LVQ, because assignment probabilities of the two closest prototypes are the highest.

Let us consider two extreme scenarios. In the first, m_k and m_j are in similar proximity to x_i. As a consequence, their assignment probabilities will be approximately the same, $g_{ik} = g_i = 0.5$. The update rule from equation (7) could then be expressed as

$$\begin{aligned} m_k^{t+1} &= m_k^t - \alpha(t)(e_{ik} - e_{ij}) \cdot (x_i - m_k^t) \\ m_j^{t+1} &= m_j^t + \alpha(t)(e_{ik} - e_{ij}) \cdot (x_i - m_j^t), \end{aligned} \tag{13}$$

where $\sigma_p{}^2$ has been neglected for simplicity (it could be incorporated in the learning rate parameter anyway). The update rule is very similar to LVQ2 update in (2). The only difference is the $(e_{ik} - e_{ij})$ term which makes the amount of update dependent on the difference in errors. It has an intuitive interpretation. If errors of the 2 prototypes are similar, their positions will not be updated. If error of k-th prototype is larger, it is moved away from data point x_i and j-th prototype is moved towards it. This is exactly the intuition used in LVQ2. The difference from LVQ2 is the case when error of k-th prototype is smaller. There, RLVQ2 is moving the k-th prototype towards and j-th prototype away from the data point. Unlike it, LVQ2 would stay put.

The second scenario is when the closest prototype is much closer than the second closest, which makes $g_{ij} \approx 0$. Following update rule (7), it could be approximated that

$$\begin{aligned} m_k^{t+1} &= m_k^t \\ m_j^{t+1} &= m_j^t. \end{aligned} \tag{14}$$

This result is analogous to the very successful "window rule" of LVQ2. Therefore, in our application of RLVQ2, we

use exactly the "window rule" used in LVQ2 to decide whether the prototype positions should be updated.

In addition to update of prototype positions, in the Regression LVQ case we have to update the prototype target values as well. We derive stochastic update rule from (8) and batch update rule from (7) as

$$\begin{aligned} \textit{stohastic}: \quad & q_k^{t+1} = q_k^t + \alpha(t)\cdot(y_i - q_k^t) \\ & q_j^{t+1} = q_j^t + \alpha(t)\cdot(y_i - q_j^t), \end{aligned} \tag{15}$$

$$\textit{batch}: \quad q_k^{t+1} = \frac{\sum_{i=1}^{n_k} y_i}{n_k}, \quad q_j^{t+1} = \frac{\sum_{i=1}^{n_j} y_i}{n_j}. \tag{16}$$

Given an unlabeled data point x_u, its target value y_u is determined as the target value q_c of its nearest prototype m_c

$$Y_u = q_c,\ c = \arg\min_j d(x_u, m_j). \tag{17}$$

Finally, in Fig. 1 we summarize the resulting RLVQ2.1 algorithm in the stochastic mode. Value of s is chosen between 0.4 and 0.8.

Receive a new training point (x_i, y_i)

Find two closest prototypes m_k and m_j
if $\min(d_{ik}/d_{ij}, d_{ij}/d_{ik}) > s$ // the window rule
use (13) and (15)
else
use (14) and (15)

Figure 1. Stochastic RLVQ2.1 algorithm

It is interesting to note that, in contrast to LVQ 2.1, Hard RLVQ was derived via optimization of a specific cost function. Interestingly, as we showed, it can be reduced to a simple heuristic in a very natural way.

D. Time Complexity

Time complexity of each run of Regression LVQ through all training examples is $\Theta(N{\cdot}P{\cdot}M)$, where N is the number of training points, M is the number of features, and P is the number of prototypes. Time complexity of calculating the assignment probabilities g_{ij} of each point x_i, is $\Theta(2{\cdot}P{\cdot}M)$, and of calculating the error e_{ij} each prototype makes is $\Theta(P{\cdot}M)$. Time complexity of target value prediction for an unlabeled point is $\Theta(P{\cdot}M)$.

While all proposed RLVQ algorithms have same time complexity, RLVQ 2 algorithm works the fastest because it does not have to deal with soft assignments. It also has the fewest hyperparameters, since it only uses α from the basic classification LVQ2 and does not use σ_p^2 or σ_y^2 needed for Soft RLVQ and Hard RLVQ algorithms.

III. Experimental Results

All proposed versions of Regression LVQ algorithm were evaluated on various regression tasks, both real-life and synthetic. We were interested in comparing the proposed Hard and Soft Regression LVQ algorithms and Regression LVQ2 with several benchmark algorithms. Only results of the stochastic Regression LVQs are shown dues to lack of space. Compared with the batch versions, stochastic versions resulted in slightly higher accuracies, as was expected [2].

A. Benchmark Regression Algorithms

In the following, we describe regression algorithms used for benchmarking of proposed Regression LVQ algorithms.

K-Nearest Neighbor algorithm is a traditional prototype-based algorithm that works both for classification and regression. Prototypes are typically all training points. There, K corresponds to the number of neighbors consulted in making the prediction. In regression scenario, prediction is given as the average label of K nearest neighbors.

Parzen Window Regression is an algorithm that behaves as a soft version of the Nearest Neighbor algorithm. It makes the prediction as the weighted average of prototype labels, where the weight is determined based on the kernel distance from the query data point. The smoothing strength of the weighting is decided by the kernel width parameter.

K-Means Regression is based on selection of prototypes using standard K-means algorithm [19] that partitions training observations into P clusters. The prototypes are defined as the cluster centers. Given such a set of prototypes, both nearest neighbor and Parzen window algorithm can be used to provide predictions. This approach ensures fixed prototype budget, consistent with the RLVQ algorithms. Therefore, it is appropriate for benchmarking.

Random Prototypes is a simple approach where P prototypes are selected at random from the training data. Again, both nearest neighbor and Parzen window algorithms can be used with such prototypes.

Quantization LVQ uses classification to solve regression problems. It quantizes the target variables into M discrete values and treats them as M classes. Then it performs LVQ2.1 on M classes using P prototypes.

B. Data Description

We evaluated Regression LVQ algorithms on several benchmark regression datasets from StatLib, Delve and UCI ML Repositories. Data sets we used can be divided in two groups: time series data sets (Table II) and the rest (Table I).

TABLE I. Non-time series data sets

Data Set	*Source*	*Training Size*	*Test Size*	*Features*
Abalone	UCI	1897	949	8
Air NM10	StatLib	333	167	7
Air NO2	StatLib	333	167	7
Body Fat	StatLib	168	84	14
Boston	UCI	337	169	13
Concrete	UCI	687	343	8
CPU small	Delve	5461	2731	12
Forest	UCI	345	172	9
Houses	StatLib	13760	6880	8
MG	[4]	923	462	6
Motor	StatLib	1454	727	6
MPG	UCI	198	99	7
Space	StatLib	2071	1036	6
Tecator	StatLib	172	43	122

TABLE II. TIME SERIES DATA SETS

Data Set	*Source*	*Training Size*	*Test Size*	*Lag*
Balloon	StatLin	1331	665	6
Ocean Shear	StatLib	4580	2290	5
River Flow	StatLib	628	314	4
Santa Fe	[17]	9991	2482	9
Sun Spots	StatLib	1878	939	3
Yield	StatLib	1655	827	9

C. Regression on a fixed budget

The proposed Regression LVQ algorithms used moderate prototype budgets of 100 on all but the 3 smallest data sets (there, budget was set to 50). The same prototype budgets were used for K-Means Regression and Random Prototypes benchmark algorithms. For the K-Nearest Neighbor we explored $K = 1$, 2 and 3. The kernel width for Parzen window was optimized using cross-validation.

Each of the Regression LVQ versions used the same hyperparameter value $\alpha_0 = 0.04$ and the update step $\alpha(t) = \alpha_0 \cdot \alpha_T/(\alpha_T+t)$, where t is the time and $\alpha_T = 5N$, where N is the size of the training set. The hyperparamether σ_p was initialized as the value of within-data variance and updated using the schedule described in Section II depending on the type of the algorithm (Soft or Hard). Hyperparamether σ_y used by the Soft RLVQ was always set to the standard deviation of the squared error made by the prototypes.

In the cases where the data set was not explicitly divided into training and test sets by the authors, we randomly selected 2/3 of the data set for training and 1/3 for testing. This process was repeated 10 times and the average R^2 values on the test sets are reported in Table III. In addition to showing the R^2 accuracies on each data set, in Table III we report how many times each algorithm was among Top 2 and Bottom 2 performers, as well as the average rank of each algorithm.

Based on the Top 2 and average rank scores we can conclude that all three versions of Regression LVQ on a fixed budget perform similarly to Non-budget Parzen window and KNN algorithms. Budget versions of 1NN and Parzen window were less accurate then Budget Regression LVQ algorithms. Additionally, K-means clustering selection of prototypes was more successful than random prototype selection in both Budget 1NN and Budget Parzen window methods. Finally, Quantization LVQ performed poorly and did not show up as a successful idea.

Comparing of Regression LVQ algorithms shows that Soft RLVQ is slightly better than Hard RLVQ. Interestingly, despite its simplicity, RLVQ2 was quite competitive.

To illustrate the behavior of the proposed algorithms in the highly resource constrained scenarios, another set of experiments was performed where the budget algorithms were restricted to a very tight budget (results not shown). As expected, the resulting R^2 accuracies were lower than when larger budgets are used. Comparing with K-means and random prototype selection approaches for budget maintenance, we observed an even larger dominance of

TABLE III. PERFORMANCE RESULTS ON A FIXED BUDGET. COMPARISON WITH BUDGET AND NON-BUDGET ALGORITHMS

Data Set	P	Hard RLVQ	Soft RLVQ	RLVQ 2	Budget QLVQ M=32	K-means Regression		Random Prototypes		Non-budget K-NN			Non-budget Parzen
		Hard	*Soft*	*Hard*		1-*NN*	*Parzen*	1-*NN*	*Parzen*	*K=1*	*K=2*	*K=3*	
Abalone	**100**	0.497	0.471	0.489	0.219	0.540	0.528	0.200	0.468	0.350	0.492	0.538	0.561
AirNM10	**100**	0.220	0.253	0.203	-0.585	0.080	0.185	-0.348	0.071	0.208	0.112	0.164	0.241
AirNO2	**100**	0.447	0.491	0.453	0.088	0.343	0.420	0.098	0.358	0.272	0.400	0.438	0.494
Balloon	**100**	0.151	0.153	0.148	0.119	0.117	0.125	-0.378	-0.128	-0.331	-0.055	0.022	0.148
Balloon_{res}	**100**	0.109	0.111	0.106	-0.321	0.065	0.082	-0.638	-0.023	-0.349	-0.168	-0.081	0.131
Boston	**100**	0.741	0.749	0.679	0.350	0.553	0.612	0.387	0.497	0.677	0.710	0.682	0.703
Body Fat	**50**	0.751	0.796	0.718	0.697	0.703	0.796	0.674	0.754	0.768	0.822	0.88	0.809
Concrete	**100**	0.711	0.666	0.694	0.326	0.503	0.559	0.302	0.485	0.604	0.635	0.644	0.646
CPU	**100**	0.952	0.949	0.947	0.909	0.947	0.953	0.905	0.929	0.955	0.964	0.968	0.969
Forest	**100**	0.056	-0.033	0.041	-0.406	-0.608	-0.583	-4.626	-0.400	-2.863	-1.018	-1.104	-0.022
Houses	**100**	0.534	0.567	0.641	0.433	0.510	0.552	0.085	0.231	0.548	0.645	0.679	0.676
MG	**50**	0.721	0.726	0.713	0.493	0.694	0.714	0.464	0.647	0.541	0.648	0.681	0.705
Motor	**100**	0.011	0.016	0.007	-0.054	0.009	0.008	-0.094	0.004	0.096	0.109	0.112	0.026
MPG	**100**	0.889	0.876	0.852	0.783	0.843	0.862	0.764	0.815	0.820	0.856	0.862	0.866
Ocean	**100**	0.996	0.995	0.994	0.989	0.995	0.995	0.989	0.990	0.997	0.997	0.998	0.997
River	**100**	0.836	0.859	0.852	0.737	0.838	0.850	0.724	0.811	0.774	0.819	0.837	0.837
Santa Fe	**100**	0.937	0.969	0.903	0.788	0.896	0.921	0.795	0.821	0.986	0.989	0.989	0.989
Space	**100**	0.526	0.564	0.485	0.147	0.519	0.551	0.095	0.448	0.468	0.555	0.577	0.611
Sun Spot	**100**	0.847	0.859	0.848	0.819	0.848	0.857	0.731	0.839	0.739	0.806	0.824	0.859
Tecator	**50**	0.740	0.706	0.669	0.459	0.659	0.675	0.484	0.571	0.762	0.852	0.862	0.819
Yield	**100**	0.324	0.317	0.302	0.032	0.286	0.301	0.077	0.196	0.306	0.347	0.354	0.357
Top 2		**6**	**9**	**3**	**0**	**1**	**0**	**0**	**0**	**1**	**4**	**9**	**9**
Bottom 2		**0**	**0**	**0**	**16**	**0**	**0**	**21**	**1**	**4**	**0**	**0**	**0**
Average Rank		**4.33**	**3.47**	**5.66**	**10.66**	**7.52**	**5.76**	**11.66**	**9.14**	**7.42**	**5.33**	**4.23**	**2.76**

RLVQ algorithms. This is expected, because proper prototype selection becomes ever more important as the budget decreases.

IV. Related Work

The idea of soft partition of the feature space for LVQ was introduced in [16] and resulted in derivation of Soft LVQ algorithm. The authors introduced two variants of Learning Vector Quantization using Gaussian mixture assumption about prototypes. The learning rule was derived by minimizing an objective function based on a likelihood ratio using the gradient descent method. The form of the resulting learning algorithm resembles the traditional LVQ algorithms. It was shown that soft partition can lead to better classification as compared to the traditional LVQ algorithms. However, Soft LVQ is limited to performing classification tasks and cannot directly be extended to perform regression.

Deterministic Annealing algorithm is a soft prototype partition algorithm first introduced for clustering [14] and then extended for vector quantization [12, 15], classification [9] and regression [13]. The general idea behind Deterministic Annealing (DA) is to find the optimal prototype positions (and target values in the regression case) by minimizing the objective cost function subject to a constraint on the entropy of the solution which is weighted by a "temperature" term that anneals during the training procedure. Our cost function D_1 can be derived from this DA cost function by omitting the entropy constraint. The intuition behind annealing the "temperature" is that it can avoid shallow local minima and produces a hard solution at the limit of zero temperature. Its drawback is that, since the cost function is defined and minimized at each temperature, the solution cannot be found in an on-line manner and can take quite long to produce due to the slow annealing process.

Mixture of local experts [6, 9 and 7] is a supervised learning procedure that decomposes a complex task into a number of simpler learning problems. The model consists of a number of specialized predictors and a gating function that decides how to combine them to make a final prediction. By using the gradient descent [6] or EM [7, 9] method each expert learns to handle a subset of the complete data set and the gating function learns how to combine the expert's predictions. This procedure can be viewed as an associative version of competitive learning. RLVQ resembles the mixture of experts approach if we treat prototypes as experts and use their distances from a query point to calculate the gating weights. In fact, the cost function D_2 proposed in Soft RLVQ closely resembles the cost function used in the mixture of experts [6].

V. Conclusion

In this paper we presented a regression extension of the popular LVQ algorithm for classification. After a thorough study of soft partition regression and expert related algorithms, we realized that the expert networks can be replaced by much more easily implemented and intuitively clearer LVQ prototypes. As a result, we proposed three different versions of LVQ algorithm for regression, one of them using a soft partition of feature space and the other two using the hard partition. The experimental results showed that all versions of our algorithm are very efficacious and that they achieve considerable improvement in prediction accuracy when compared to other prototype based methods and perform similar to memory unconstrained methods.

Acknowledgment

This work was supported by the U.S. National Science Foundation under Grant IIS-0546155.

References

[1] Biehl M., Ghosh A., Hammer B., Dynamics and generalization ability of LVQ algorithms, *The Journal of Machine Learning Research*, vol. 8, pp. 323-360, 2007.

[2] Bottou L., Stochastic learning, *Advanced Lectures on Machine Learning* pp. 153-174, 2003.

[3] Dempster A. P., Laird N. M. and Rubin D. B., Maximum likelihood from incomplete data via the EM algorithm, *J. Royal Statistical Society B*, vol. 39, pp. 1-38, 1977.

[4] Flake G. W. and Lawrence S., Efficient SVM regression training with SMO. *Machine Learning*, vol 46, pp. 271-290, 2002.

[5] Hart P. E., The condensed nearest neighbor rule, *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515–516, 1968.

[6] Jacobs R. A., Jordan M. I., Nowlan S.J and Hinton G.E., Adaptive mixtures of local experts, *Neural Computation*, vol. 3, pp. 79-87, 1991.

[7] Jordan M. I., Jacobs R. A., Hierarchical mixtures of experts and the EM algorithm, *Neural Computation*, vol. 6, pp. 181-214,1994.

[8] Kohonen T., The Self-organizing Map, *Proceedings of the IEEE*, vol 78, pp. 1464-1480, 1990.

[9] Miller D., Rao A. V., Rose K., and Gersho A., A global optimization technique for statistical classifier design, *IEEE Trans. Signal Processing*, vol. 44, pp. 3108–3122, 1996.

[10] Prodromidis A., Chan P., Stolfo S., Advances in Distributed and Parallel Knowledge Discovery, MIT Press Cambridge, MA, 2000.

[11] Prodromidis A., Chan P., and Stolfo S., Meta-learning in distributed data mining systems: Issues and approaches. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, Cambridge, MA, 2000.

[12] Rao A., Miller D., Rose K. and Gersho A., A generalized VQ method for combined compression and estimation, *IEEE Intern. Conf. on Acoustics Speech and Sig. Proc.*, vol. 4, pp. 2032-2035, 1996.

[13] Rose K., Deterministic Annealing for Clustering, Compression, classification, regression and related optimization problems, *Proc. IEEE*, vol 86, pp. 2210–2239, 1998.

[14] Rose K., Gurewitz E., Fox G., A deterministic annealing approach to clustering, *Pattern Recognition Letters*, vol 11, pp. 589-594, 1990.

[15] Rose K., Gurewitz E., Fox G., Vector quantization by deterministic annealing, *IEEE Transactions on Information Theory*, vol 38, pp. 1249-1257, 1992.

[16] Seo S. and Obermayer K.. Soft learning vector quantization, *Neural Computation*, vol. 15, pp. 1589-1604, 2003.

[17] Weigend A. S., Mangeas M. and Srivastava A. N., Nonlinear gated experts for time series: discovering regimes and avoiding overfitting, *International Journal of Neural Systems*, vol. 6, pp. 373-399, 1995.