Adaptive feature space Transformation in Generalised Matrix Learning Vector Quantization

MASTER THESIS

Johann Bernoulli Institute of Mathematics and Computer Science

University of Groningen
The Netherlands

Moses Matovu

February 18, 2010

Abstract

We propose and investigate a modification of Generalized Matrix Relevance Learning Vector Quantization (GMLVQ). In the novel approach we restrict the linear transformation to only the data set instead of transforming both the prototypes and the data like in the original GMLVQ. The method is implemented using a rectangular transformation matrix in a modified Euclidean distance measure. We analyse the performance of the modified algorithm and compare with original GMLVQ. In this paper, the method is outlined and experimental results are discussed in terms of a benchmark classification task.

1 Introduction

Several classification techniques for differentiating (or discriminating) features and patterns in data sets do exist however, they provide different classification convergence behaviours, some associated with various setbacks. Various domains and applications require more efficient algorithms that provide better performances. Thus, the reason why several variants of classifiers are being proposed, developed and deployed in various domains with the view that newer ones may provide better performance results. The need for better optimal classifiers prompts the continuity of on going research activities (and/or projects).

Our research project is based on Learning Vector Quantization (LVQ) algorithms where much emphasis is put on adapting and modifying the Generalized Matrix Learning Vector Quantization (GMLVQ) algorithm, which is introduced and discussed in [7, 14] and extended in [15, 16] to a novel approach where linear matrix transformation is only restricted to the data set instead of transforming both the data and the prototypes like it is the case in the GMLVQ algorithm. We investigate and analyse whether the modified algorithm (possibly) provides better or attains similar converging classification performances on same data sets as GMLVQ(MxN).

This paper is structured in 7 sections as follows. Section 1 gives a brief introduction and the purpose (and/or motivation) of this research; in section 2, we give general introductory remarks about the LVQ algorithms and review some of them like LVQ1, which introduced the basic idea of prototype (codebook) learning based on heuristic codebook updates; Generalised LVQ (GLVQ) algorithm, which is based on cost function using gradient descent; and relevance learning algorithms such as GRLVQ, and GMLVQ; section 3 discusses our approach of modifying GMLVQ algorithm; in section 4, we discuss and analyse the set-ups of the experiments and the data used in training both the modified algorithm and GMLVQ; in section 5, we discuss the results of the various experiments carried out; in section 7, we give conclusive remarks and recommendations; and finally, acknowledgements in section 8.

2 LVQ Algorithms

Various pattern classification algorithms do exist associated with differences in their convergence results. In order to improve on the optimization of performances and attain better convergences, extensive research has been done and is still on going. It is a result of such needs that led to the emergence of many algorithms among which are the LVQ methods, which have performed better compared to many other algorithms on high data dimensionality.

LVQ is a method for training competitive network layers in a supervised setting where a competitive layer learns to classify input vectors. The classes formed are dependant on only the distance (similarity measure) between the classes and the input vectors.

Learning in LVQ networks classifies input vectors targeting classes specified by the user [3]. The emergence of LVQ algorithms provided an alternate machine learning approach of handling the challenge of dealing with high data dimensionality associated with some classifiers. LVQ algorithms provide better performance behaviours in terms of computational cost (as regards resources and time) and the convergence. It involves an intuitive and simple though powerful classification method [2]. The method is easy to implement; a user can control the complexity of the resulting classifier; it provides classifiers that can deal with multi-class data problems; and the resultant classifier is human understandable because of the intuitive classification of data points to the class of their closet prototype [14].

In this paper, we review some of the LVQ algorithms but to a brief extent. In the next section, we discuss LVQ1, GLVQ, GRLVQ and GMLVQ, which is the foundation of this research.

2.1 Classification in LVQ

Learning Vector Quantization (LVQ) algorithms are a group of learning algorithms based on nearest prototype classification concept that was introduced and proposed by Kohonen [2, 3]. LVQ algorithms are on-line supervised versions of Vector Quantization (VQ) competitive learning classification approaches that are used extensively. Since the introduction of LVQ, a number of variants aimed at providing better performance have been proposed and developed. LVQ methods are used when there is a set of labelled input data. Classes are pre-defined. There is a set of reference vectors (prototypes), $\omega_j \in \mathbb{R}^N$, for $j = 1, 2, 3, ..., K$, where K is the number of

prototypes, that are used to approximate the different data classes. Each prototype carries a label $c(\omega_j) \in \{1, 2, ..., C\}$, which can attain up to C number of classes. Note that K = C if only one prototype per class is used during training.

Like other prototype-based algorithms, LVQ algorithms provide a good generalisation of classification for high dimensional data [10]. Classification is attained by determining the closest of the prototypes and returning the class label of the winning prototype. After determining the closet prototype (or set of prototypes) from the original given set of prototypes, the closest (set of) prototype(s) is then updated in such a way that, if its class label is the same as the label of the data sample, the prototype is attached to the data set of this class and if otherwise, it is pushed away from this data point that belongs to a different class. This is the basic idea of prototype learning.

Classification in LVQ algorithms is based on the nearest prototype metric where a set of chosen prototype vectors is used. LVQ algorithms rely on the distance measured between a data point, ξ and the class to which the nearest prototype belongs. Usually, the algorithms employ standard Euclidean metric as the similarity (distance) measure [5] but other forms of the distance measure can be used, depending on the domain of application.

If we consider a given set of training data, $(\xi_i, y_i) \in \mathbb{R}^N \times \{1, 2, ..., C\}$, for i = 1,2...,P, where N denotes the data dimensionality, P is the number of examples and C is the number of different classes, where classification of the data into C classes is required. Classification in LVQ is achieved by a winner-takes-all rule based on the concept of finding the nearest prototype such that the distance between the data point ξ and prototype ω, denoted by $d(\omega, \xi)$ has to be minimum for the winning prototype. A fixed number of vectors (prototypes), ω_i for each class is chosen, and a data point $\xi \in \mathbb{R}^N$ is then mapped to the class label $c(\xi) = c(\omega_i)$ of the closest prototype (the winner), for which $d(\omega_i, \xi) \leq d(\omega_j, \xi)$ holds for all i $\neq$ j. This is the basic approach used by all LVQ algorithms.

2.2 LVQ1

According to [2], learning in LVQ methods determines weight locations for prototypes so that the given training data sets can be mapped onto the corresponding class labels. LVQ1 builds on the idea of the standard Self-Organising Maps (SOM) introduced and discussed in [3].

Given input vectors ξ and weights (prototypes) ω_j, the main objective in LVQ1, is to determine a set of prototypes that best represent each class. LVQ1 applies labels of inputs to determine the best classification label for each prototype, ω_j. After a number of training iterations have been carried out and deemed sufficient, the learned prototypes are used in the nearest prototype classification. The algorithm checks the input classes and moves prototype ω_j accordingly at each iteration, and ω_j is updated in accordance with the winner-take-all approach. The generic form of updating these prototypes is

$$\omega_{p+1} = \omega_p - \Delta\omega_p \tag{1}$$

If the input vector ξ and the associated weight, ω_j (the winner) have the same class label, $c(\omega_j) = c(\xi)$, then they are both moved together by

$$\Delta\omega_j(t) = +\eta_\omega(t)[\xi - \omega_j(t)]$$

as in SOM [3] to give the update

$$\omega_j(t+1) = \omega_j(t) + \eta_\omega(t)\Big(\xi - \omega_j(t)\Big) \tag{2}$$

If input vector, ξ is correctly classified, the algorithm continues with the next element otherwise, if the input vector, ξ and the associated weight, ω_j (the winner) have different class labels, $c(\omega_j) \neq c(\xi)$, then they are moved apart using

$$\Delta\omega_j(t) = -\eta_\omega(t)\left[\xi - \omega_j(t)\right]$$

and the update of $\omega_j(t)$ to $\omega_j(t+1)$ proceeds as

$$\omega_j(t+1) = \omega_j(t) - \eta_\omega(t)\Big(\xi - \omega_j(t)\Big) \tag{3}$$

where $\eta_\omega(t)$ is the learning rate of the prototypes, which is an iteration dependent parameter used to control algorithm convergence and decreases with the number of iterations (epochs) of the training. The value of $\eta_\omega(t)$ may be constant or may vary throughout the learning process in order to ensure convergence of the algorithm. If weights ω_j(t) correspond to other input vectors (no winner), ω_j(t) remains unchanged.

There are modifications of LVQ1 based on the same concept of heuristic prototype updates. These are LVQ2.1 and LVQ3, see [1, 2, 3] and GLVQ introduced in [9]. In both LVQ2.1 and LVQ3, two winning code vectors, one having a correct label and another having a wrong label (the two prototypes ω_j and ω_k, which belong to the correct class and the wrong class of ξ respectively are the nearest neighbours to data point, ξ) are changed simultaneously at each update step unlike in LVQ1, where only one codebook vector is changed at each update step. The simultaneous update of $\omega_{j(k)}$ is given by

$$\omega_j(t+1) = \omega_j(t) + \eta(t)\Big(\xi - \omega_j(t)\Big) \tag{4}$$

$$\omega_k(t+1) = \omega_k(t) - \eta(t)\Big(\xi - \omega_k(t)\Big) \tag{5}$$

2.3 GLVQ

Like LVQ1, LVQ2.1 and LVQ3; the generalized LVQ (GLVQ) algorithm proposed by Sato and Yamada [9] is also based on the nearest prototype classification concept. It determines the closest correct prototype and closest incorrect prototype using the winner-take-all scheme. This method is based on the minimization of the cost function.

The prototypes, $\omega_{j(k)}$ are adapted such that for each class, the corresponding prototypes represent the class as accurately as possible. This requires a minimum relative distance difference between the points of the class and the corresponding prototypes, given by

$$\mu(\xi) = \left(\frac{d_j - d_k}{d_j + d_k}\right)$$

where $d_j = d(\omega_j, \xi_i)$ is the distance of the data point ξ_i from the closest correct prototype ω_j of the same class label and $d_k = d(\omega_k, \xi_i)$ is the distance of the data point ξ_i from the closest wrong prototype ω_k of a different class label than that of data point, ξ_i.

Note that $\mu(\xi)$ ranges between -1 and +1 and is negative when a data point, ξ_i is correctly classified otherwise, it is positive (when a data point ξ_i is wrongly classified). This approach involves minimizing the misclassification (error) measure, using a stochastic gradient descent approach. The error improves when $\mu(\xi)$ decreases for all inputs. Eq. (6) gives a very flexible approach introduced in [5], which involves minimising the cost function (and aims at maximizing the number of correctly classified data points). The learning rule is therefore formulated as the minimization of the cost function, f defined by Eq. (6).

$$f = \sum_{i=1}^{P} \phi\left(\mu\left(\xi_i\right)\right) = \sum_{i=1}^{P} \phi\left(\frac{d_j - d_k}{d_j + d_k}\right) \tag{6}$$

From Eq. (6), f is minimized by updating the prototypes ω_j and ω_k based on steepest descent approach. In Eq. (6), the quantities

$$d_j = d\left(\omega_j, \xi_i\right) \text{ with } c\left(\omega_j\right) = c\left(\xi\right),$$

$$d_k = d\left(\omega_k, \xi_i\right) \text{ with } c\left(\omega_k\right) \neq c\left(\xi\right),$$

where P is the number of input vectors for training, and ϕ is a monotonically increasing function such as the logistic function or the identity $\phi\left(x\right) = x$, which we use throughout the following. Also, note that the numerator of Eq. (6) can only be smaller than 0 if and only if the classification of the data point is correct, which provides greater classification security.

The learning rule is derived from the formulation of cost function, given in Eq. (6) by taking derivatives with respect to the prototypes ω, which yields an adaptation rule based on gradient. If we assume that the similarity measure $d\left(\omega, \xi\right)$ can be differentiated with respect to ω; to minimize $\boldsymbol{f}$, prototypes ω_j and ω_k have to be updated based on the steepest descent method to give

$$\Delta\omega_j = +\eta\phi^{'}\left(\mu\left(\xi\right)\right)\mu^{+}\left(\xi\right)\nabla_{\omega_j} d_j\left(\xi\right) \tag{7}$$

$$\Delta\omega_k = -\eta\phi^{'}\left(\mu\left(\xi\right)\right)\mu^{-}\left(\xi\right)\nabla_{\omega_k} d_k\left(\xi\right) \tag{8}$$

where η is the learning rate, $\phi^{'}$ is the derivative of function, ϕ taken at position $\mu\left(\xi\right)$, $\mu^{+}\left(\xi\right) = \frac{2.d_k}{\left(d_j + d_k\right)^2}$, $\mu^{-}\left(\xi\right) = \frac{2.d_j}{\left(d_j + d_k\right)^2}$, and $\mu\left(\xi\right) = \left(\frac{d_j - d_k}{d_j + d_k}\right)$.

This choice of employing a standard Euclidean metric as the similarity measure yields the Generalized Learning Vector Quantization (GLVQ) algorithm.

2.4 Relevance Learning in LVQ Algorithms

The idea of relevance learning is now widely applied in newer variants of LVQ algorithms. Generalised Relevance Learning Vector Quantization (GRLVQ) method proposed in [7], is a variant of LVQ algorithms that aims at producing better convergence results by employing relevance factors in the similarity measure, $d^{\lambda}\left(\omega, \xi\right)$.

If we consider a set of training data points, $\xi_k \in \mathbb{R}^N \times \{1, 2, ..., C\}$, for k = 1,2, ..., P; where classification of the data into C, classes is required, the squared Euclidean distance measure is formulated as

$$d^{\lambda}\left(\omega, \xi\right) = \sum_{i}^{N} \lambda_i \left(\xi_i - \omega_i\right)^2 \tag{9}$$

with relevance terms $\lambda_i \geq 0$ for every dimension, i, and $\sum_i \lambda_i = 1$. GRLVQ is a powerful approach that supports prototype learning (based on the nearest prototype classification concept) in the presence of of high data dimensionality features of different, yet a priori unknown, relevance [5].

According to [7, 5], the use of relevance factors λ_i in the similarity measure enhances easy interpretation. Dimensions with large λ_i are considered to be more important for classification

while those with very small (or zero) relevances indicate that the corresponding feature could be omitted.

The choice of the metric with relevance factors does not have to be global but can be attached locally to single prototypes. If that happens, individual updates take place for relevance factors λ^j for each prototype ω_j, and the distance of a data point ξ from prototype ω_j, $d^{\lambda^j}(\omega, \xi)$ is computed based on λ^j, which allows local relevance adaptation. Localised GRLVQ (LGRLVQ) is a variant of GRLVQ method with localised relevance factors attached to a single prototype, see [5] for more details.

Generalised Matrix Learning Vector Quantization (GMLVQ) introduced and analysed in [7, 14] is another variant of LVQ algorithms based on relevance learning. It gives an important extension of the concept of using relevance factors in the similarity measure. It has two variants, GMLVQ(NxN) and GMLVQ(MxN) according to [15, 16]. The GMLVQ algorithm is based on the use of full matrices of relevances in the distance similarity measure of the form

$$d^{\wedge}(\omega, \xi) = (\xi - \omega)^T \wedge (\xi - \omega) \tag{10}$$

where $\wedge$ is an NxN full matrix ($\wedge \in \mathbb{R}^{N \times N}$). An Euclidean metric is derived from Eq. (9) by deciding on the suitable parameters to use [10]. Accordingly, the similarity measure becomes a squared Euclidean distance metric if matrix, $\wedge$ is positive (semi-) definite, so that,

$$d^{\wedge}(\omega, \xi) = \Big(\Omega\,(\xi - \omega)\Big)^2 \geq 0 \tag{11}$$

$$\text{where, } \wedge = \Omega^T.\Omega \text{ with } \Omega \in \mathbb{R}^{N \times N} \text{ or } \Omega \in \mathbb{R}^{M \times N}, \text{ where } M{<}N.$$

The original GMLVQ employs a symmetric squared (quadratic) matrix, Ω in the implementation of Eq. (10), which is extended in [15, 16] with the use of a rectangular matrix, Ω ($\Omega \in \mathbb{R}^{M \times N}$, where $M{<}N$) of limited rank corresponding to low but varying dimensionality representation of the data. This provides reduction of the number of adaptive parameters where 2- or 5- or 9- or 13-dimensional representations are deemed to provide sufficient and efficient visualization.

The dimension of matrix, Ω (even $\wedge$) plays a key role because it influences how the prototypes and the data are transformed and/or projected in the feature space. Two forms of matrix, Ω are used in the formulation of the similarity measure, which results into the two variants of GMLVQ. The choice of which form to use depends on the shape and the dimension required to address, [15, 16] proposes the following; (i) Quadratic and symmetric matrix, Ω (i.e. $\Omega \in \mathbb{R}^{N \times N}$, $\Omega_{ij} = \Omega_{ji}$), (ii) Quadratic and non-symmetric matrix, Ω (i.e. $\Omega \in \mathbb{R}^{N \times N}$, $\Omega_{ij} \neq \Omega_{ji}$), and (iii) Rectangular matrix, Ω (i.e. $\Omega \in \mathbb{R}^{M \times N}, M{<}N$).

Note that, a rectangular matrix, $\Omega \in \mathbb{R}^{M \times N}$, where M = N is a special case of the rectangular matrix that is equivalent to a non-symmetric quadratic (square) matrix, Ω.

To effectively reduce the dimensionality of data, the GMLVQ extension does the training of prototypes and identifying of suitable transformations simultaneously unlike other algorithms where dimensionality reduction is a pre-processing step [15, 16]. The extended method provides a possibility to incorporate prior knowledge about the intrinsic dimension of the data efficiently, and significantly reduces the number of free parameters in the learning problem.

The computation of the derivatives with respect to matrix, Ω, therefore depends on the shape and the dimensionality of matrix, Ω. With two different forms of the transformation matrix, Ω selected and used in GMLVQ(NxN) and GMLVQ(MxN), gives rise to two different alternatives for expressing $d^{\wedge}(\omega, \xi)$ in terms of Ω. Hence, $d^{\wedge}(\omega, \xi)$ is expressed in terms of a symmetric quadratic matrix, $\Omega \in \mathbb{R}^{N \times N}$ as

$$d_1^{\wedge}(\omega,\xi)=\sum_{i,j,k}^{N}(\xi_i-\omega_i)\,\Omega_{ik}\Omega_{kj}\,(\xi_j-\omega_j) \tag{12}$$

The expression of $d^{\wedge}(\omega,\xi)$ in terms of either a rectangular matrix ($\Omega\in\mathbb{R}^{M\times N}$, where $M\neq N$), or a quadratic non-symmetric matrix ($\Omega\in\mathbb{R}^{M\times N}$, where $M=N$), gives

$$d_2^{\wedge}(\omega,\xi)=\sum_{i,j}^{N}\sum_{k}^{M}(\xi_i-\omega_i)\,\Omega_{ki}\Omega_{kj}\,(\xi_j-\omega_j) \tag{13}$$

Training in LVQ involves minimizing the cost function and the learning rule is derived by taking the derivatives of the cost function with respect to the prototypes and the involved metric parameters. Thus, the adaptation formulae for GMLVQ variants whose formulation is given in Eq. (10) are derived to attain a stochastic gradient descent.

To attain the learning criterion and improve on the error rates, the cost function, f of the form given in Eq. (6), (where $f=\sum_i\phi\left(\frac{d_J^{\wedge}-d_K^{\wedge}}{d_J^{\wedge}+d_K^{\wedge}}\right)$ with $\phi(x)=x$ being a function that increases monotonically) has to be minimized by updating the prototypes and metric parameters with their respective derivatives.

If we consider a data point, ξ with the closest correct prototype, ω_J and the closest wrong prototype, ω_K; the update equations are obtained based on the strategies given in Eq. (14) and Eq. (15) for the prototypes and matrix, Ω respectively.

$$\omega_{J(K)}=\omega_{J(K)}-\eta_{\omega}\frac{\partial f}{\partial\omega_{J(K)}} \tag{14}$$

$$\Omega^{J(K)}=\Omega^{J(K)}-\eta_{\Omega}\frac{\partial f}{\partial\Omega} \tag{15}$$

where η_{ω} and η_{Ω} are the respective learning rates of the prototypes and matrix, Ω, and ∂f is the derivative of f, see the flexible learning approach proposed in [14] derived as a minimization of the cost function, of the form given in Eq. (6). Taking the derivative of f with respect to ω, we get

$$\tfrac{\partial f}{\partial\omega_J}=-\phi^{'}\left(\mu(\xi)\right)\mu^{+}(\xi)\,\nabla_{\omega_J}d_J^{\wedge}$$

and

$$\tfrac{\partial f}{\partial\omega_K}=+\phi^{'}\left(\mu(\xi)\right)\mu^{-}(\xi)\,\nabla_{\omega_K}d_K^{\wedge}$$

But the derivative of $d^{\wedge}(\omega,\xi)$ with respect to ω is as given by

$$\frac{d^{\wedge}(\omega,\xi)}{\partial\omega_{J,K}}=-2\wedge(\xi-\omega_{J,K}) \tag{16}$$

By substitution, we get

$$\frac{\partial f}{\partial\omega_J}=+\phi^{'}\left(\mu(\xi)\right)\mu^{+}(\xi).2\wedge(\xi-\omega_J) \tag{17}$$

$$\frac{\partial f}{\partial\omega_K}=-\phi^{'}\left(\mu(\xi)\right)\mu^{-}(\xi).2\wedge(\xi-\omega_J) \tag{18}$$

From Eq. (14), the updates of the closest correct prototype, ω_J and closest wrong prototype, ω_K are given by

$$\Delta\omega_J = +\eta_\omega.\phi^{'}\left(\mu\left(\xi\right)\right).\mu^{+}\left(\xi\right).2 \wedge \left(\xi - \omega_J\right) \tag{19}$$

$$\Delta\omega_K = -\eta_\omega.\phi^{'}\left(\mu\left(\xi\right)\right).\mu^{-}\left(\xi\right).2 \wedge \left(\xi - \omega_K\right) \tag{20}$$

where $\mu^{+}\left(\xi\right) = \frac{2.d_K^{\wedge}}{(d_J^{\wedge}+d_K^{\wedge})^2}$, $\mu^{-}\left(\xi\right) = \frac{2.d_J^{\wedge}}{(d_J^{\wedge}+d_K^{\wedge})^2}$, $\mu\left(\xi\right) = \left(\frac{d_J^{\wedge}-d_K^{\wedge}}{d_J^{\wedge}+d_K^{\wedge}}\right)$, and $\phi^{'}$ is the derivative of the cost function ϕ; index $_{J(K)}$ refers to the closest correct (wrong) prototype $\omega_{J(K)}$ and η_ω is the learning rate for the prototypes.

For the update of matrix elements, Ω_{lm}, we get

$$\Delta\Omega_{lm} = -2\eta_\Omega.\phi^{'}\left(\mu\left(\xi\right)\right). \tag{21}$$

$$\left(\mu^{+}\left(\xi\right)\left(\left(\xi_m - \omega_{J,m}\right)\left[\Omega\left(\xi - \omega_J\right)\right]_l\right) - \mu^{-}\left(\xi\right)\left(\left(\xi_m - \omega_{J,m}\right)\left[\Omega\left(\xi - \omega_K\right)\right]_l\right)\right)$$

where the parameters $\mu^{+}\left(\xi\right)$, $\mu^{-}\left(\xi\right)$, $\mu\left(\xi\right)$ and $\phi^{'}$ are as defined for Eq. (19) and Eq. (20), and η_Ω, is the learning rate of the parameters.

The two learning rules are given special terms in [15, 16] based on the choice of the transformation matrix, Ω employed in the formulation of the similarity measure. The original GMLVQ is termed as GMLVQ(NxN) due to the matrix, Ω having a square shape ($\Omega \in \mathbb{R}^{N\times N}$), and its extension as GMLVQ(MxN) because matrix, Ω has a rectangular shape ($\Omega \in \mathbb{R}^{M\times N}$, where $M < N$).

The learning rates η_ω and η_Ω (assuming that $\eta_\omega \gg \eta_\Omega$ for a slower time scale than that of the prototypes) are chosen independent of one another.

3 GMLVQ(MxN) modification

In our proposed novel approach, we investigate and adapt GMLVQ classification algorithm. We modify the use of the full matrix of relevance vectors in GMLVQ's formulation of the similarity measure to a new method that uses matrices of relevance vectors that only transform the data set instead of transforming both the data and the prototypes like in GMLVQ. We investigate the use of rectangular matrices used in GMLVQ(MxN) for the implementation our proposed new approach. We train both both the GMLVQ(MxN) and its modification on the same set of data to determine their classification performances, which we compare to analyse how they both perform.

We use more or less the same implementation approach as that used to implement GLMVQ [7] and GMLVQ(MxN) [15, 16]. We extend the use of a matrix relevance vectors, such that for a data point, ξ from prototype, ω, we modify and formulate the distance similarity measure of the form used in the formulation of the GMLVQ, Eq. (10) to the algorithm of the form

$$d^{\Omega}\left(\omega, \xi\right) = \left(\omega - \Omega\xi\right)^{T}\left(\omega - \Omega\xi\right) \tag{22}$$

Accordingly, this ensures that the matrix transforms only the data sets according to Eq. (22). This is the formulation of the algorithm of our approach, from which two varying implementations due to the different matrix shapes (symmetric quadratic and rectangular) can be obtained.

We use of a rectangular matrix, Ω in our approach, which GMLVQ(MxN) algorithm employs. Our approach is based on the same principle as that of GMLVQ algorithm and many other LVQ variants, where the updates move the closest prototype, ω_J towards a data point, ξ that belongs

to the same category and move away the closest prototype, ω_K that belongs to a different class label than that of the data point, ξ.

Training in LVQ involves minimizing the cost function, and the learning rule is derived from this cost function by taking its derivatives with respect to the prototypes and the involved metric parameters. If we consider the similarity measure given in Eq. (22), its adaptation formulae are derived using $f = \sum_i \phi(\frac{d_J^\Omega - d_K^\Omega}{d_J^\Omega + d_K^\Omega})$, from Eq. (6) to attain a stochastic gradient descent by computing the derivatives of f with respect to both ω and Ω. If we take derivatives of f with respect to ω, we have

$$\frac{\partial f}{\partial \omega_J} = -\phi^{'}\left(\mu\left(\xi\right)\right).\mu^{+}\left(\xi\right).\nabla_{\omega_J} d_J^\Omega$$

$$\frac{\partial f}{\partial \omega_K} = +\phi^{'}\left(\mu\left(\xi\right)\right).\mu^{-}\left(\xi\right).\nabla_{\omega_K} d_K^\Omega$$

But the derivatives of $d^\Omega\left(\omega, \xi\right)$ with respect to ω, is

$$\frac{\partial d^\Omega}{\partial \omega} = 2\left(\omega - \Omega\xi\right) \tag{23}$$

Hence, by substitution,

$$\frac{\partial f}{\partial \omega_J} = -\phi^{'}\left(\mu\left(\xi\right)\right)\mu^{+}\left(\xi\right).2\left(\omega_J - \Omega\xi\right) \tag{24}$$

$$\frac{\partial f}{\partial \omega_K} = +\phi^{'}\left(\mu\left(\xi\right)\right)\mu^{-}\left(\xi\right).2\left(\omega_K - \Omega\xi\right) \tag{25}$$

where $\phi^{'}$ is the derivative of the function ϕ, with $\mu^{+}\left(\xi\right) = \frac{2.d_K}{(d_J+d_K)^2}$, $\mu^{-}(\xi) = \frac{2.d_J}{(d_J+d_K)^2}$ and $\mu\left(\xi\right) = \left(\frac{d_J^\Omega - d_K^\Omega}{d_J^\Omega + d_K^\Omega}\right)$.

Using the strategy given in Eq. (14) for updating both $\omega_{J(K)}$, we obtain the following update equations

$$\Delta\omega_J = -2\eta_\omega \phi^{'}\left(\mu\left(\xi\right)\right)\mu^{+}\left(\xi\right).\left(\omega_J - \Omega\xi\right) \tag{26}$$

$$\Delta\omega_K = +2\eta_\omega \phi^{'}\left(\mu\left(\xi\right)\right)\mu^{-}\left(\xi\right).\left(\omega_K - \Omega\xi\right) \tag{27}$$

Note that, from Eq. (24),

$$\frac{\partial f}{\partial \omega_J} = -\frac{4.d_K}{(d_K+d_J)^2}\left(\omega_J - \Omega\xi\right),$$

and from Eq. (25),

$$\frac{\partial f}{\partial \omega_K} = +\frac{4.d_J}{(d_K+d_J)^2}\left(\omega_K - \Omega\xi\right).$$

Substituting the values of $\phi^{'}\left(\mu\left(\xi\right)\right)$, $\mu^{+}\left(\xi\right)$ and $\mu^{-}\left(\xi\right)$, we obtain the following simplified equations, which indicate the changes in prototypes.

$$\Delta\omega_J = -\eta_\omega \frac{4.d_K}{(d_K + d_J)^2}\left(\omega_J - \Omega\xi\right) \tag{28}$$

$$\Delta\omega_K = +\eta_\omega \frac{4.d_J}{(d_K + d_J)^2}\left(\omega_K - \Omega\xi\right) \tag{29}$$

We now consider the update of a single matrix element, $\Omega_{l,m}$. Because of the rectangular shape of the matrix, Ω, the expression of $d^{\Omega}(\omega, \xi)$ in terms of Ω is

$$\frac{\partial d^{\Omega}(\omega, \xi)}{\partial \Omega_{lm}} = \sum_{i,j}^{N} \sum_{k}^{M} (\omega_i - \Omega_{ki}\xi_i)(\omega_j - \Omega_{kj}\xi_j) \tag{30}$$

The derivative of $d^{\Omega}(\omega, \xi)$ with respect to a single rectangular matrix element, Ω_{lm}, is

$$\frac{\partial d^{\Omega}(\omega, \xi)}{\partial \Omega_{lm}} = 2 \sum_{i} -2\xi_m (\omega_i - \Omega_{li}\xi_i) \tag{31}$$

By substitution, we have

$$\frac{\partial d^{\Omega}(\omega, \xi)}{\partial \Omega_{lm}} = -4 \sum_{i} \xi_m . (\omega_i - \Omega_{li}\xi_i) \tag{32}$$

For the update of matrix elements, Ω_{lm}, we get

$$\Delta\Omega_{lm} = +4\eta_{\Omega} . \phi^{'}(\mu(\xi)) . \Big(\mu^{+}(\xi) \Big(\xi_m [\omega_J - \Omega\xi]_l \Big) - \mu^{-}(\xi) \Big(\xi_m [\omega_K - \Omega\xi]_l \Big) \Big) \tag{33}$$

Now that we have discussed the formulation of our modified approach of GMLVQ(MxN), next, we discuss the various experiments and associated results.

4 Experiments

We train our approach on the same data sets as GMLVQ(MxN) to compare their respective performances.

Various data sets can be used for training the algorithms, which include; artificial data, image segmentation data (that can be obtained from UCI Repository [8]), iris data and bio-informatics data. But for purposes of simplifying the experiments, we train with with only one data set, whose results from the experiments will be discussed. We chose segmentation data.

Experiments are carried-out to test the performances of our approach with available data sets so as to compare and analyse the classification performances attained against GMLVQ(MxN) performance. We reduce the number of feature dimensionality of the data used to train the algorithms to ease the analysis of the classification performances. Because of the rectangular shape of matrix, Ω that is used, we use M-dimensional prototypes, which results in fewer prototype components. The unused dimensions are neglected.

In all the experiments, prototype training is done for at least 1000 time steps (epochs) and the adaptation of the parameters starts after some number of time steps, which varies with different dimensions. The learning rates are re-set continuously (increased) during the entire duration of training with the initial values set to $\eta_{\omega} = 0.001$ and $\eta_{\Omega} = 0.0001$ (each value of η_{Ω} being 10 times smaller than that of the corresponding η_{ω}). The learning goes on, as the training error decreases until it remains constant, at which point the optimal values of the learning rates are determined. The optimal values of the learning rates are found and set to 0.1 and 0.01 respectively. We use the same schedule of the learning rates as that used in GMLVQ of the form given by Eq. (33) and Eq. (34) for η_{ω} and η_{Ω} respectively;

$$\eta_{\omega}(t) = \frac{\eta_{\omega}(0)}{1 + c(t-1)} \tag{34}$$

$$\eta_{\Omega}(t) = \frac{\eta_{\Omega}(0)}{1 + c(t-1)} \tag{35}$$

where t counts the number of training epochs, factor c determines the speed and is chosen independently for every application. $\eta_{\omega}(t)$ and $\eta_{\omega}(t)$ are learning rates at epoch, t and $\eta_{\omega}(0)$ and $\eta_{\omega}(0)$ are the initial learning rates at epoch, 0. Note that, these learning rates η_{ω}, for the metric and η_{Ω}, for the prototypes are chosen independent of one another.

There are various ways that can be used for the initialization of the prototypes in LVQ algorithms. Different prototype initializations approaches are used for GMLVQ(MxN) and the modified algorithm. In GMLVQ(NxN), the prototypes are initialized by randomly choosing 10% of all training samples of each class and computing their corresponding mean values. They are N-dimensional prototypes. In our approach, we randomly choose 10% of the transformation of the training data by matrix, Ω for each class and compute their corresponding mean values which we then use to initialize the prototypes. There are M-dimensional prototypes to correspond to the MxN matrices, Ω that project to an N-dimensional space.

In the experiments, after every update; matrix, Ω is normalised. It is updated and normalized repeatedly a number of time steps (epochs) to be able to achieve convergence, which is attained after 1000 epochs. To normalize matrix,Ω, the $\sum_{ij} \Omega_{ij}^2 = 1$ is used by dividing all elements of matrix, Ω by $\sum_i \Omega_{ii}^2$ after every update.

Since both GMLVQ(MxN) and the modified algorithm have to be trained with the appropriate data sets to facilitate in analysing their respective performances; next, we discuss the nature of the data used in the various experiments.

4.1 Data

During the training and testing of our approach, image segmentation data is used. It contains 19-dimensional feature vectors with different attributes of 3x3 pixel regions extracted out of seven outdoor images. each sub-region is assigned to one of the seven classes; brickface, sky, foliage, cement, window, path and grass, in that order, see [8]. The training data set consists of 210 data points with 30 samples for each of the seven (7) classes and the test data set contains 300 data points per class.

Features 3, 4 and 5 for both test and training data sets are deemed (almost) constant and are excluded (eliminated) from being used in the experiments. The remaining 16 are pre-processed. The features are normalised to zero mean and unit variance.

In the next section, we discuss the results obtained from the various experiments carried out.

5 Results

Various experiments were carried-out. Different observations and results were obtained with varying dimensions, M. Some of which we will discuss in detail. The ones we discuss give generalized performances for other dimensions. The overall classification performances of our approach in comparison to those of GMLVQ(MxN) for different dimensions of transformation matrix, Ω are displayed in Fig.1 (left and right panels for GMLVQ(M×N) and its variant modification respectively), with mean percentage accuracies plotted as a function of dimension, M. Note that, the mean performance accuracy results are averaged over a number of trials (10 runs to be exact).

The results obtained after training the modified version of GMLVQ(M×N) on segmentation data are compared with the classification performance results of GMLVQ(M×N) in order to analyse if the modified version gives similar, better or worse performance (and/or convergence).

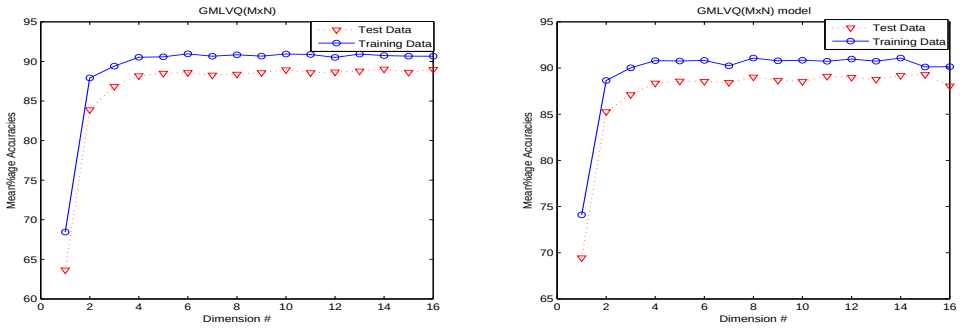

Figure 1: Visualization of the classification performances after training GMVLQ(MxN) and its modified algorithm (left and right panels respectively), with one prototype per class for segmentation data set as a function of dimensions, M. The accuracies in both cases for training- and test data sets are displayed on an average of 10 randomized initializations. Both algorithms exhibit low performance for dimension, $\mathbf{M = 1}$, though the modification's performance is better than that of GMLVQ(MxN) by approximately 0.6% for both test- and training data sets. With dimension, M = 2; both algorithms perform close to the optimal performance but still the modified algorithm performs better with a 0.74% and 1.4% on training- and test data respectively. Both algorithms converge with optimal performances starting at dimension, M = 4. GMLVQ(MxN) performance stabilizes compared to the modified algorithm's performance, which has slight differences and fluctuations for higher dimensions.

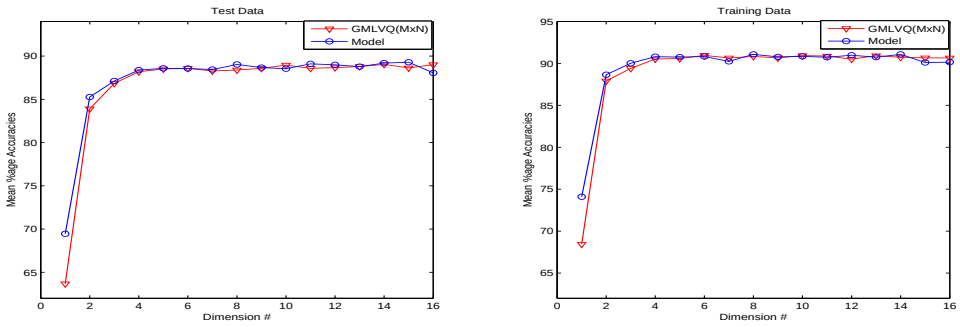

Figure 2: Visualization of the classification performances after training both GMVLQ(MxN) and the modified algorithm with one prototype per class as a function of dimensions, M for segmentation data set. The fluctuations of performances accuracies for both algorithms after training them with test- and training data sets averaged over 10 randomized initializations, are displayed in the left and right panels respectively. The modified algorithm performs better than GMLVQ(MxN) for lower dimensions but does not give stable results for higher dimensions.

The mean classification accuracies during the entire course of training both the GMLVQ(MxN) algorithm and its modified variant on the training- and test data sets with the varying dimensions, M re displayed in Fig. 1 for the both algorithms. Fig. 2 shows the performance variations for the two algorithms on test data (left panel) and training data (right panel).

Table 1 and Table 2 summarize the mean classification accuracies on both the training- and test data sets during the entire process of training with the GMLVQ(MxN) and modified algorithm respectively for a few cases of dimension, M. The modified algorithm shows better performance on the test- and training data sets than GMLVQ(MxN) with dimension M = 1, with a higher standard deviation, see Table 3, than the rest of the dimensions. Therefore, both algorithms provide performances which are similar.

Table 1: Mean classification accuracies of GMLVQ(MxN) are averaged over 10 runs for each dimension. Performances of a few dimension cases for GMLVQ(MxN) algorithm are summarised in the table. Lower dimensions, $M \in 1, 2$ give low performances but rest give high performances.

Algorithm	Accuracies	
	Test Data	Training Data
GMLVQ(1xN)	63.67	68.45
GMLVQ(2xN)	83.90	87.91
GMLVQ(5xN)	88.51	90.58
GMLVQ(9xN)	88.58	90.66
GMLVQ(12xN)	88.67	90.51
GMLVQ(16xN)	89.00	90.65

Table 2: Mean classification accuracies of the GMLVQ(MxN) modification are averaged over 10 runs for each dimension. Performances of a few cases of dimension for the modified algorithm are summarised in the table. Lower dimensions, $M \in 1, 2$ give low performances but higher dimensions produce high performances.

Algorithm	Accuracies	
	Test Data	Training Data
GMLVQ(1xN) modified	69.45	74.10
GMLVQ(2xN) modified	85.27	88.65
GMLVQ(5xN) modified	88.58	90.75
GMLVQ(9xN) modifiied	88.67	90.78
GMLVQ(12xN) modified	88.99	90.97
GMLVQ(16xN) modified	88.05	90.15

Table 3: Standard deviations of the modified algorithm against GMLVQ(MxN) on test and training data for a few dimensions, M=1 exhibits the highest standard deviation. Standard deviations realised with the rest of the dimensions suggest that both algorithm perform equally.

Algorithm	Standard Deviations			
	1	2	9	16
Test data	0.0409	0.0097	0.0006	0.0067
Training data	0.0400	0.0052	0.0008	0.0035

Both algorithms exhibit low performances with dimension, $M \in 1, 2$. The performance increases with increase in dimension, **M** (see Fig. 1 and Fig. 2). Both GMLVQ(MxN) algorithm and its modification perform close to the optimal performance with dimension, $\mathbf{M = 3}$, see left panel and and right panels of Fig. 1 for their respective performance behaviours. Generally, GMLVQ(MxN) modification algorithm performs equally as good as the GMLVQ(MxN) algorithm on both data sets with every dimensions, **M**. After attaining the optimal classification performance, both algorithms exhibit very minimal fluctuations in performances as the dimensions increase. Though, the performance of GMLVQ(MxN) stabilises, while the one of modified algorithm fluctuates slightly.

There are more performance behaviours and convergence results as a result of training the two algorithms to be considered. They vary from dimension to dimension. For that matter, we will not discuss the results for every dimension, M. We will instead consider and discuss results separately for a few cases (3 cases) for dimension, M (i.e. M = 2, 9 and 16).

Case 1: Dimension, M = 9

We investigate and analyse the performance behaviour of training both algorithms with dimension, $M = 9$ on segmentation data. Below, we give a full account and analysis of the results and observations from the various experiments.

The plot of the cost function against epoch for both cases, see Fig. 3, shows that there is similar behaviour of the curve for the two algorithms which emphasizes that gradient descent steps are correctly and effectively implemented. A flexible approach given in Eq. (6) enables the minimization of the cost function.

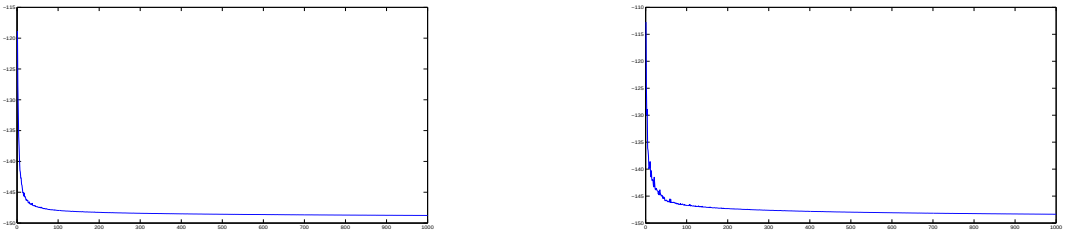

Figure 3: Visualization of cost function curves of the GMLVQ(9xN) algorithm and the modified algorithm after training both algorithms on segmentation data. Both algorithms display similar curve behaviours because of the proper implementation of gradient descent using Eq. (6).

From Table 1 and Table 2 (even Fig.1 and Fig. 2), it be observed that the modified algorithm has almost similar classification performance as GMLVQ(9xN) algorithm on both the test- and training data sets averaging 88.67% and 90.78%, with 0.12% and 0.09% difference on the GMLVQ(9xN) algorithm respectively. With more training time steps, we were able to attain slightly higher performance (minimum error) accuracies.

Fig. 4 represents eigenvalues of the global matrix, $\wedge$ $(=\Omega^T\Omega)$ for GMLVQ(9xN) and its modification in the left and right panels respectively. Note that in our modification, the global matrix, $\wedge$ is obtained using the same notation as that used for GMLVQ(9xN) for comparison purposes even though it is not required in our implementation. Both figures show that there is correlation in the way both algorithms behave. There is significant change in the first eigenvalue, which is always increasing for both cases.

Figure 4: Visualization of the evolution of eigenvalues of the global matrix, $\wedge$ ($=\Omega^T\Omega$) plotted as a function of epochs during training both the GMLVQ(9xN) algorithm and its modification on segmentation data, as shown in the left and right panels of the figure respectively. The first eigenvalue increases drastically in both cases whereas the rest, which are non-zero, decrease and stabilize after a few epochs.

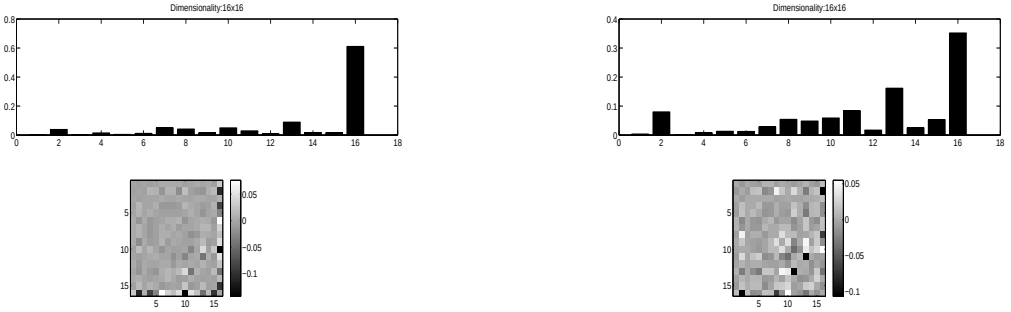

Figure 5: Visualization of the diagonal and off-diagonal elements of global matrix $\wedge$ after 1000 epochs training of the GMLVQ(9xN), left panel and the modified algorithm, right panel. The diagonal elements are set to zero for the plot. The feature indexed 16 is ranked highest in both algorithms.

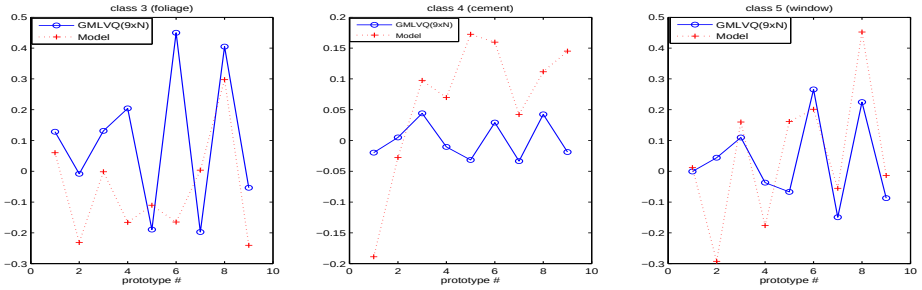

Figure 6: Visualization of prototype positions of class 3 (foliage), class 4 (cement) and class 5 (window) as identified by both GMLVQ(9xN) and the modified algorithm, left and right panels of the figure respectively. We use Ω to project prototypes for the GMLVQ(9xN) algorithm, which has a big influence on the final locations in the feature space. Prototypes in the modified algorithm are considered to be implicitly projected.

The analysis of the diagonal elements of the full matrix in Fig. 5 shows that the last feature (indexed 16) is ranked highest followed by feature indexed 13 in both algorithms; the rest have low values. In our modified approach, there are no off-diagonal elements with zero values but in GMLVQ(9xN), there are some off-diagonal elements with value zero.

In our modified approach, prototypes are are defined in the feature space projected by matrix Ω, which is not the case in GMLVQ(9xN). We achieve the same influence, by projecting the prototypes of GMLVQ(9xN) with matrix, Ω after training for purposes of comparison. The position projections of the prototypes of 3 of the 7 classes of segmentation data for both algorithms are as depicted in Fig. 6 (for classes 3, 4, and 5). The metrics used in these algorithms have a great influence on these positions. The behaviours of the two algorithms are quiet different.

Case 2: Dimension, M = 16 (Special Case)

We discuss the results obtained and observations analysed after training both algorithms with dimension, M = 16. A rectangular matrix, $\Omega \in \mathbb{R}^{M \times N}$, where $M = N$, is a special case that is equivalent to a quadratic (square) matrix, $\Omega \in \mathbb{R}^{N \times N}$. Hence, for M = 16 and the number of features, **N** = 16, we have a **16x16** square matrix, Ω.

From Table 1 and Table 2, the classification performances for both algorithms are similar. The modified algorithm averages 88.05% and 90.15% compared to 89.00 and 90.65% for GMLVQ(16xN) on the test- and training data sets respectively. These results are more or less the same as the optimal classification performance accuracies, see Fig. 1 and Fig. 2.

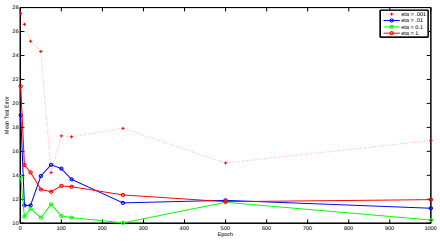

Figure 7: Evolution of mean test error during the process of training GMLVQ(16xN) with segmentation test data.

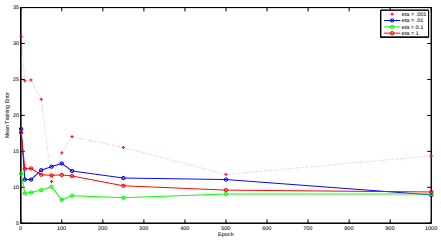

Figure 8: Evolution of mean training error during training of GMLVQ(16xN)with segmentation training data.

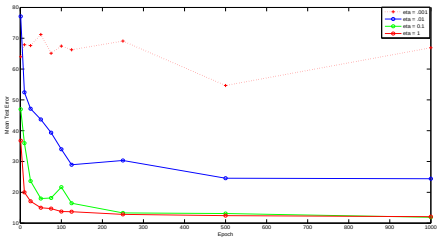

Figure 9: Evolution of mean test error during the process of training GMLVQ(16xN) modification with segmentation test data.

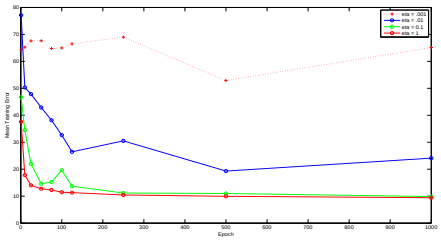

Figure 10: Evolution of mean training error during training of GMLVQ(16xN) modification with segmentation training data.

The mean classification errors during the entire course of training with GMLVQ(16xN) are shown in Fig. 7 and Fig. 8 respectively, and in Fig. 9 and Fig. 10 for the GMLVQ(16xN) modification on the test- and training data sets respectively. Mean percentages of error are plotted as function of training time for different variations of prototypes (and the metric parameters) learning rates. For both algorithms, the training errors drastically reduce and stabilise afterwards.

The cost function is repeatedly computed through the entire training process. Fig. 11 shows a plot of cost function against epoch, representing cost functions of GMLVQ(16xN) and its modification (left and right panels of the figure respectively). Both curves have a decreasing slope (negative gradient) due to the proper implementation of gradient descent. The curves have the same behaviours.

Figure 11: Visualization of cost function against epoch for GMLVQ(16xN) and the modified variant. Left and right panels respectively.

The projections of prototypes of 3 of the 7 classes of segmentation data for both algorithms are as shown in Fig. 12 (for classes 3, 4, and 5). The metrics definitely influenced these positions. The behaviours of the two algorithms are quiet different.

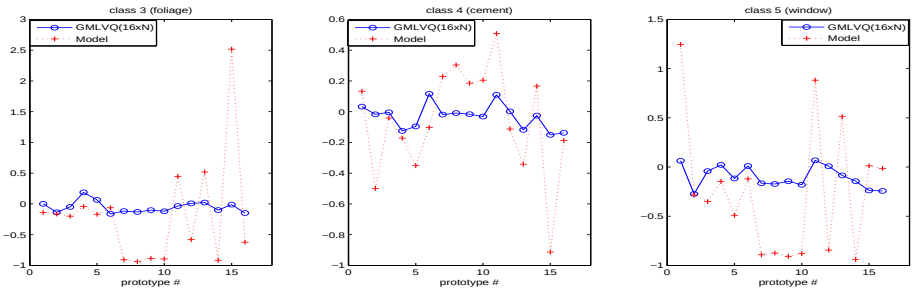

Figure 12: Visualization of prototype positions of class 3 (foliage), class 4 (cement) and class 5 (window) as identified by both GMLVQ(16xN) and the modified algorithm, as displayed by the left and right panels of the figure respectively.

The eigenvalues, off-diagonal elements and diagonal elements of the global relevance matrices obtained as a result of training both GMLVQ(16xN) and the modified algorithm on the segmentation data are shown in Fig. 13 and Fig. 14. An analysis of the diagonal elements of the global full matrix show that the last feature (indexed 16) is ranked highest for both algorithms followed by the feature indexed 13. For GMLVQ(16xN), most of the features have very low values compared to the modified algorithm close to zero.

The global matrix, $\wedge$ ($=\Omega^T\Omega$), after training both algorithms with segmentation data sets has the eigenvalues shown in Fig. 14 (left and right panels for GMLVQ(16xN) and the modified

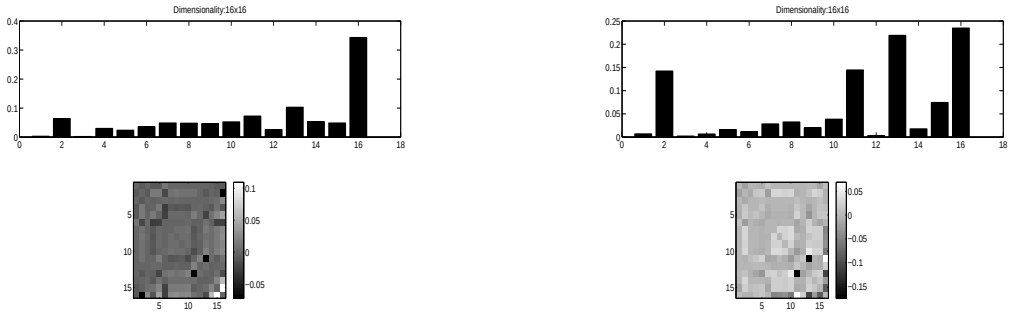

Figure 13: Visualization of the diagonal and off-diagonal elements of global matrix $\wedge$ after 1000 epochs training of the GMLVQ(16xN), left panel and the modified algorithm (right panel). The diagonal elements are set to zero for the plot.

algorithm respectively), which are obtained in every run with different values of Ω.

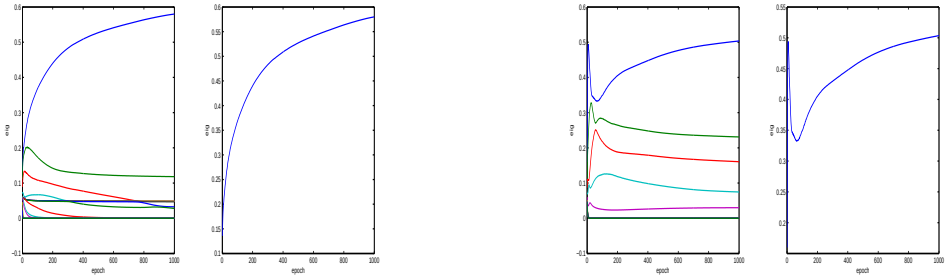

Figure 14: Visualization evolution of eigenvalues of matrix, $\wedge$ ($=\Omega\Omega^T$) plotted as a function of epochs during the process of training the GMLVQ(16xN) algorithm and its modified variant, as shown in the left and right panels of the figure respectively. After 1000 epochs, all eigenvalues are non-zero. The first one increases drastically during metric adaptation while others decrease after a few epochs and stabilize.

The features were observed throughout the training process. After approximately 1000 epochs, for both algorithms, only one eigenvalue remains increasing. The left part of the left panel of Fig 14 displays all the eigenvalues of matrices, $\wedge$ for GMLVQ(16xN) as a function of training time (epochs), so is the left part of the right panel but for the GMLVQ(16xN) modification. For GMLVQ(16xN), all eigenvalues apart from the first one start decreasing to zero almost immediately at the start of metric adaptation. After 1000 epochs, only one eigenvalue remains. For the modified algorithm, it can be observed in Fig. 14 (right panel) that some eigenvalues, at the start of metric adaptation increase and then start diminishing shortly after about 50 epochs, then increase slightly for like another 50 epochs and start diminishing, apart from the first eigenvalue, which increases drastically and start stabilising after 100 epochs.

Case 3: Dimension, M = 2

We consider the case where the two algorithms are trained with rectangular transformation matrices, Ω of dimension, M = 2. From Table 1, Table 2 and Figure 1, both algorithms exhibit

relatively low classification performances compared to higher dimensions. But the performance is better in both cases than for dimension, M = 1.

Fig. 15 shows a plot of cost function against epoch, left and right panels of the figure representing the behaviours GMLVQ(2xN) and the modified algorithm respectively. Both display a decreasing slope (negative gradient) due to gradient descent, implemented using Eq. (6).

Figure 15: Visualization of the cost function against epoch for GMLVQ(2xN) and the modified version, left and right panels respectively.

The final locations of the prototypes of 3 classes out of 7 in the feature space are as depicted in Fig. 16 (for classes 3, 4 and 5). There is a difference in the behaviours of the two algorithms.

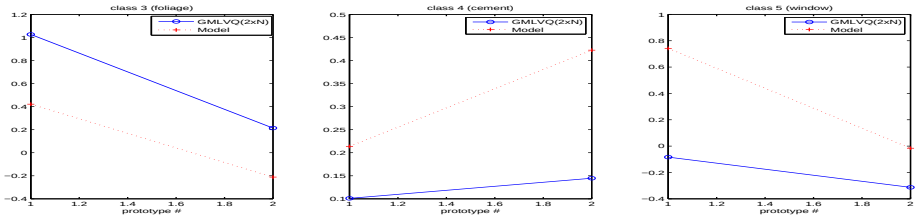

Figure 16: Visualization of prototype positions of class 3 (foliage), class 4 (cement) and class 5 (window) as identified by both GMLVQ(2xN) algorithm and its modified variant, displayed in the left and right panels of the figure respectively.

An analysis of the global relevance matrix elements shows that diagonal elements of the full matrix rank the last feature (indexed 16) highest for both algorithms, see Fig. 17.

Figure 17: Visualization of the diagonal and off-diagonal elements of global matrix $\wedge$ after 1000 epochs training of the GMLVQ(2xN), left panel and the modified algorithm (right panel). The diagonal elements are set to zero for the plot.

The global matrices, $\wedge$ ($=\Omega^T\Omega$), of both the GMLVQ(2xN) and modified algorithm have the eigenvalues shown in Fig. 17, which are obtained in every run with different values of Ω.

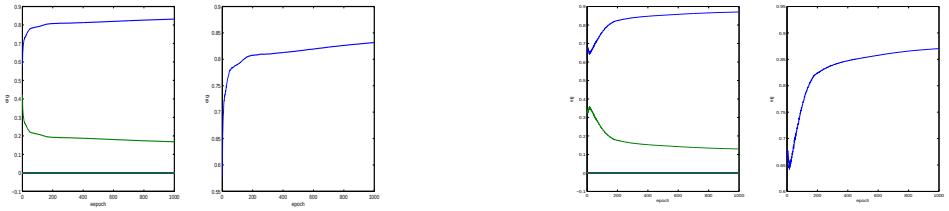

Figure 18: Visualization of the eigenvalues of global matrix, $\wedge$ ($=\Omega^T\Omega$) as a function of epochs during the process of training the GMLVQ(2xN) algorithm and its modification, as depicted in the left and right panels of the figure respectively. In both cases, two non-zero eigenvalues are used to determine convergence. The right hand plots in both the left and right panels of the figure display the evolution of the first eigenvalue.

After approximately 1000 epochs, for both variants, there is only one eigenvalue remaining. Fig. 18 displays the eigenvalues of the global matrices, $\wedge$ for GMLVQ(2xN) and its modified version (on the left and right panels of the figure respectively) as a function of training time. In both cases, the first eigenvalue increasing almost immediately at the start of metric adaptation, and the second diminishes to zero. After 1000 epochs, only the first eigenvalue remains increasing, the second stabilises but is greater than zero.

The global matrices, $\wedge$ ($=\Omega^T\Omega$), of both the GMLVQ(2xN) and modified algorithm have the eigenvalues shown in Fig. 18, which are obtained in every run with different values of Ω.

Dimension, M = 2, provides a good visualisation tool of the data sets. Fig. 19 displays a visualization for labelled segmentation data after the transformations with matrix, Ω. Both algorithms give good classification performance in the two dimensions (M = 2). In the training, 10 independent runs are performed to obtain classification with a single prototype per class.

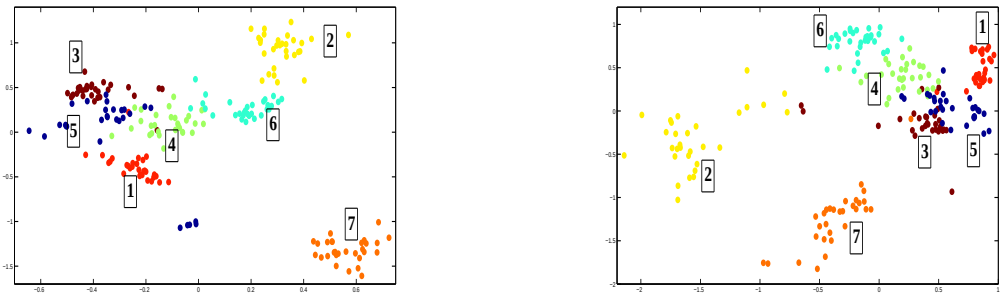

Figure 19: Visualization of the image segmentation data set as transformed in the GMLVQ(2xN) algorithm and its modified version displayed in the left and right panels of the figure respectively. The two axes correspond to the components of the projection, $\Omega\xi$. All data samples for each of the labelled 7 classes are displayed. Only one prototype per class is used during training. The two projections are quite different but not so different. Each of the classes 2 and 7 are projected far away from the rest for each algorithm.

Most of the results obtained in the case of dimension, M = 2 are totally different from other cases discussed possibly because there is very low degree of freedom.

6 Conclusion

In this research project, we investigated GMLVQ algorithm (with two variants GMLVQ(NxN) and GMLVQ(MxN)), a prototype based classification but put much emphasis on GMLVQ(MxN), which we adapted and modified it by replacing the full matrix of relevances that transforms both the data and the prototypes with a matrix that transforms only the data as given by Eq. (22).

Our modified algorithm provides almost similar results when trained with image segmentation data as those of the GMLVQ(MxN) variant. Having a matrix transform only the data rather than transforming both the data and the prototypes as is the case in the GMLVQ(MxN), does not decrease or increase the classification convergence significantly. We obtained good performances in the predictions observed for both training and test data sets that were considered for experiments, which is almost the same performance as that achieved by GMLVQ(MxN) algorithm. Since the results are similar with the established results of GMLVQ [15, 16], the modified algorithm,s metric is sufficient to be used as a similarity distance measure to achieve classification performance.

We would like to train both algorithms on artificial data, iris data and possibly bio-informatics data and see how they perform.

We would like to investigate further how the re-formulation of Eq. (22) to the one given below, with the introduction of function, *phi*, which may be a square matrix of dimension 2x2 or 16x16 would affect the classification performances. Consider the following equation

$$d^{\Omega}(\omega, \xi) = (\omega - \Omega\xi)^T \Phi (\omega - \Omega\xi) \tag{36}$$

where $\Omega \in R^N$ matrix, Φ is a 2x2 matrix, ω is 2-dimensional so that the learning rule is achieved by finding the derivatives with respect to Ω, Φ and ω.

7 Acknowledgements

We would like to thank the NUFFIC co-ordinators of the NPT programme at the University of Groningen for giving this research a life-line given the fact that it ought have been completed much earlier than now. Special appreciation go to the supervisors, Prof. dr. Michael Biehl and Prof. dr. ir. Paris Avgeriou but more especially Prof. dr. Michael Biehl for their constant guidance, endurance and tolerance towards the project/research completion; and also, to Drs. Petra Schneider for her positive role and contributions. We thank the Uganda Parliamentary Commission and Kyambogo University for their support.

References

[1] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, Jorma Laaksonen, and Kari Torkkola; LVQ Programming Team of Helsinki University of Technology, Labaratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND *LVQ-PAK: The Learning Vector Quantization Program Package*, 1996.

[2] T. Kohonen, *Self-organising maps. Springer, Berlin*, 1995.

[3] T. Kohonen, *Self-organisation and Associative Memory, Second Edition. Berlin, Springer-Verlag*, 1987.

[4] B. Hammer, M. Strickert and T. Villmann, *Supervised neural gas with general similarity measure, Neural Processing Letters 21(1): 21 - 44*, 2005.

[5] P. Schneider, M. Biehl and B. Hammer, *Adaptive Relevance Matrices in Learning Vector Quantization, Neural Computation 21:3532-3561*, 2009.

[6] A. Ghosh, M. Biehl and B. Hammer. *Performance analysis of LVQ algorithms: a statistical physics approach. Neural Networks 19:817 - 829*, 2007.

[7] B. Hammer and T. Villmann, *Generalised Relevance Learning Vector Quantization, Neural Networks 15: 1059 - 1068*, 2002.

[8] D. J. Newman, S. Hettich, C. L. Blake, C. J, Meez; *UCI Repository of machine learning database - http://www.ics.uci.edu/ mlearn/MLRepository.html; Irvine, CA: University of California, Department of Information and Computer Science.*, 1998.

[9] Atsushi Sato and Keiji Yamada, *Generalized Learning Vector Quantization, In Advances in Neural Information Processing Systems, volume 8, pages 423-429*, 1996.

[10] P. Schneider, M. Biehl and B. Hammer, *Relevance Matrices in Learning Vector Quantization, Proc. European Symposium on Artificial Neural Networks ESANN, M. Verleysen (ed.), d-side (2007) 37-43*, 2007.

[11] R. Duda, P. Hart and D. Stock, *Pattern Classification, Whiley*, 2001.

[12] M. Biehl, A. Ghosh and B. Hammer, *Learning Vector Quantization: the Dynamics of Winner-Takes-All algorithms, Neuralcomputing 69: 660 - 670*, 2006.

[13] M. Biehl, B. Hammer, and P. Schneider, *Matrix Learning in Learning Vector Quantization, Technical Report ifl-06-14, Institute of Informatics, Clausthal, University of Technology*, 2006.

[14] P. Schneider, F.-M. Schleif, T. Villman and M. Biehl, *Generalized Matrix Learning in Learning Vector Quantization for the Analysis of Spectra Data, Proc. Europ. Symp. on Artificial Neural Networks ESANN, M. Verletsen (ed.), d-side publishing 451-456*, 2008.

[15] P. Schneider, K. Bunte, H. Stiekema, B. Hammer, T. Villman and M. Biehl, *Regularization in Matrix Relevance Learning, in press, IEEE Trans. Neural Networks*, 2010.

[16] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villman and M. Biehl, *Discriminative Visualization by Limited Rank Matrix Learning, Machine Learning Reports 02/2008, Univ. Leipzig*, 2008.