Adaptive Metric Learning Vector Quantization for Ordinal Classification

Shereen Fouad and Peter Tino[1]

[1]School of Computer Science, The University of Birmingham, Birmingham B15 2TT, United Kingdom, (e-mail: saf942, P.Tino@cs.bham.ac.uk.)

Keywords: Generalized Matrix Learning Vector Quantization (GMLVQ), Matrix Learning Vector Quantization (MLVQ), Ordinal Classification.

Abstract

Many pattern analysis problems require classification of examples into naturally ordered classes. In such cases nominal classification schemes will ignore the class order relationships, which can have detrimental effect on classification accuracy. This paper introduces two novel ordinal Learning Vector Quantization (LVQ) schemes, with metric learning, specifically designed for classifying data items into ordered classes. Unlike in nominal LVQ, in ordinal LVQ the class order information is utilized during training in selection of the class prototypes to be adapted, as well as in determining the exact manner in which the prototypes get updated. Prototype based models are in general more amenable to interpretations and can often be constructed at a smaller computational cost than alternative non-linear classification models. Experiments demonstrate that the proposed ordinal LVQ formulations compare favourably with their nominal counterparts. Moreover, our methods achieve competitive performance against existing benchmark ordinal regression models.

1 Introduction

Most classification algorithms focus on predicting data labels from nominal (non-ordered) classes. However, many pattern recognition problems involve classifying data into classes which have a natural ordering. This type of problem, known as ordinal classification or ordinal regression, is commonly seen in several real life applications, as in information retrieval (Chu et al., 2007) and medical analysis (Cardoso et al., 2005). In such problems, although it is still possible to use the conventional (nominal) methods, the order relation among the classes will be ignored, which may affect the stability of learning and the overall prediction accuracy.

Lot of effort has already been devoted to the problem of ordinal classification in the machine learning literature. A simple approach involves converting ordinal regression to a set of nested binary classification problems that encode the ordering of the original ranks. Results of these nested binary classifications are combined to produce the overall label predictions. For example, (Frank et al., 2001) employs binary classification tree learners, while (Waegeman et al., 2006) uses binary Support Vector Machines (SVM).

Another stream of ordinal regression research assumes that ordinal labels originate from coarse measurements of a continuous variable. The labels are thus associated with intervals on the real line. A group of algorithms, known as threshold models, focuses on two main issues:

1) How to find the 'optimal' projection line, representing the assumed linear order of classes, onto which the input data will be projected;

2) How to optimally position thresholds defining the label intervals so that the margin of separation between neighbouring classes is maximized.

For example, in the SVM context, a class of models under the name of Support Vector Ordinal Regression (SVOR) have been developed. Shashua and Levin (Shashua et al., 2002) proposed two large-margin principles: **(i)** The fixed-margin principle, in which the margin of the closest pair of classes is being maximized leading to equal margins between two neighbouring classes (the assumption that is too strict in most cases); **(ii)** The sum of margins principle, which allows for different margins and only the sum of all $K - 1$ margins is maximized (assuming there are K ordered categories).

However, the order on the $K-1$ class thresholds was not imposed. Therefore this work was further extended in the SVOR with EXplicit ordering Constraints (SVOR-EXC) formulation (Chu et al., 2007), where the order of class thresholds is considered explicitly. Furthermore, Chu and Keerthi (Chu et al., 2007) also presented an alternative SVOR model, namely SVOR with IMplicit ordering Constraints (SVOR-IMC).

Based on the SVOR-EXC and SVOR-IMC methods, Li and Lin (Li et al., 2007; Lin et al., 2012) presented a reduction framework from ordinal regression to binary classification based on 'extended' examples. We refer to this model as REDuction-SVM (RED-SVM). This work was further extended into another reduction framework known as Weighted LogitBoost (Xia et al., 2007).

However, the SVM based algorithms all suffer from high computational complexity (in the number of training points) (Sun et al., 2010). Therefore, Sun et al. (Sun et al., 2010) introduced a (non-SVM)-based model with a lower computational complexity - Kernel Discriminant Learning for Ordinal Regression (KDMOR).

In this paper, we propose two novel Learning Vector Quantization based learning models specifically designed for classifying data into ordered classes. Learning Vector Quantization (LVQ), originally introduced by Kohonen in (Kohonen, 1986, 1998), constitutes a family of supervised learning multi-class classification algorithms. Classifiers are parameterized by a set of prototypical-vectors, representing classes in the input space, and a distance measure[1] on the input data. In the classification phase, an unknown sample is assigned to the class represented by the closest prototype. Compared to SVM type methods, prototype based models are in general more amenable to interpretations and can be constructed at a smaller computational cost. The function of such classifiers can be more directly understood because of the intuitive classification of data points to the class of their closet prototype (under a given metric).

In particular, we extend the recently proposed modifications of LVQ, termed Matrix LVQ (MLVQ) and Generalized MLVQ (GMLVQ) (Schneider et al., 2009; Schneider , 2010), to the case of ordinal classification. In MLVQ/GMLVQ the prototype positions, as well as the (global) metric in the data space can be modified. Unlike in nominal LVQ, in the proposed ordinal LVQ the class order information is utilized during training in selection of the class prototypes to be adapted, as well as in determining the exact

[1] Different distance metric measures can be used to define the closeness of prototypes.

manner in which the prototypes get updated. To the best of our knowledge this paper presents the first attempt at extending the LVQ model with metric learning to ordinal regression.

This paper is organized as follows: Section 2 gives a brief introduction to the LVQ based methods related to this study. In section 3 we introduce two novel ordinal LVQ approaches for classifying data with ordered labels. Experimental results are presented and discussed in section 4. Section 5 concludes the study by summarizing the key contributions.

2 Learning Vector Quantization (LVQ) and its Extensions

Learning Vector Quantization (LVQ) constitutes a family of supervised learning algorithms which uses Hebbian online learning to adapt prototypes to the training data (Kohonen, 1998). The original version, named LVQ1, was introduced by Kohonen in 1986 (Kohonen, 1986).

Assume training data $(x_i, y_i) \in \mathbb{R}^m \times \{1, ..., K\}$, $i = 1, 2, ..., n$ is given, m denoting the data dimensionality and K is number of different classes. A typical LVQ network consists of L prototypes $w_q \in \mathbb{R}^m$, $q = 1, 2, 3, ..., L$, characterized by their location in the input space and their class label $c(w_q) \in \{1, ..., K\}$. Obviously, at least one prototype per class needs to be included in the model. The overall number of prototypes is a model (hyper) parameter optimized e.g. in a data driven manner through a validation process.

Given a distance measure $d(x_i, w)$ in $\mathbb{R}^m$, i.e. the distances of the input sample x_i to the different prototypes w, classification is based on a winner-takes-all scheme: a data point $x_i \in \mathbb{R}^m$ is assigned to the label $c(w_j)$ of prototype w_j with $d(x, w_j) < d(x, w_q), \forall j \neq q$.

Each prototype w_j with class label $c(w_j)$ will represent a receptive field in the input space[2]. Points in the receptive field of prototype w_j will be assigned class $c(w_j)$ by

[2]The receptive field of prototype w is defined as the set of points in the input space which pick this prototype as their winner.

the LVQ model. The goal of learning is to adapt prototypes automatically such that the distances between data points of class $c \in \{1, ..., K\}$ and the corresponding prototypes with label c (to which the data belong) is minimized.

In the training phase for each data point x_i with class label $c(x_i)$, the closest prototype with the same label is rewarded by pushing it closer to the training input; the closest prototype with different label is penalized by moving it away of the pattern x_i.

Several modifications of this basic learning scheme have been proposed, aiming to achieve better approximation of decision boundaries, faster or more robust convergence (Sato et al., 1996; Hammer et al., 2002). Many LVQ variants use the squared Euclidean distance $d^2(x, w) = (x - w)^T(x - w)$ as a distance measure between prototypes and feature vectors. However, the use of Euclidean distance can be problematic in case of high-dimensional, heterogeneous data sets where different scalings and correlations of dimensions can be observed. Recently, special attention was paid to schemes for manipulating the input space metric used to quantify the similarity between prototypes and feature vectors (Schneider et al., 2009; Hammer et al., 2002). Generalized Relevance LVQ (GRLVQ), introduced in (Hammer et al., 2002), proposed an adaptive diagonal matrix acting as the metric tensor of a (dis)similarity distance measure. This was further extended in Matrix LVQ (MLVQ) and Generalized Matrix LVQ (GMLVQ) (Schneider et al., 2009; Schneider , 2010) that use a fully adaptive metric tensor. Metric learning in the LVQ context has been shown to have a positive impact on the stability of learning and the classification accuracy (Schneider et al., 2009; Schneider , 2010).

2.1 Matrix LVQ (MLVQ)

Matrix LVQ (MLVQ) (Schneider , 2010) is a new heuristic extension of the basic LVQ1 (Kohonen, 1986) with a full (e.g. not only diagonal elements) matrix tensor based distance measure. Given an $(m \times m)$ positive definite matrix $\mathbf{\Lambda}$, the algorithm uses a generalized form of the squared Euclidean distance

$$d^{\mathbf{\Lambda}}(x_i, w) = \sqrt{(x_i - w)^T \mathbf{\Lambda} (x_i - w)}. \tag{1}$$

Positive definiteness of $\mathbf{\Lambda}$ can be achieved by substituting $\mathbf{\Lambda} = \mathbf{\Omega}^T \mathbf{\Omega}$, where $\mathbf{\Omega} \in \mathbb{R}^{m \times m}$, $1 \leq l \leq m$ is a full-rank matrix. Furthermore, $\mathbf{\Lambda}$ needs to be normalized after

each learning step to prevent the algorithm from degeneration.

For each training pattern x_i, the algorithm implements Hebbian updates for the closest prototype w and for the metric parameter $\mathbf{\Omega}$. If $c(x_i) = c(w)$, then w is attracted towards x_i, otherwise w is repelled away (for more details please consult (Schneider , 2010)).

2.2 Generalized MLVQ (GMLVQ)

Generalized Matrix LVQ (GMLVQ, see (Schneider et al., 2009; Schneider , 2010)) is a recent extension of the Generalized LVQ (Sato et al., 1996) that uses the adaptive input metric (1). The model is trained in an on-line-learning manner, minimizing the cost function

$$f_{GMLVQ} = \sum_{i=1}^{n} \Phi \left(\frac{d_+^{\mathbf{\Lambda}}(x_i, w) - d_-^{\mathbf{\Lambda}}(x_i, w)}{d_+^{\mathbf{\Lambda}}(x_i, w) + d_-^{\mathbf{\Lambda}}(x_i, w)} \right) \tag{2}$$

based on the steepest descent method. Φ is a monotonic function, e.g. the logistic function or the identity $\Phi(\ell) = \ell$ (which we use throughout the paper), $d_+^{\mathbf{\Lambda}}(x_i, w)$ is the distance of data point x_i from the closest prototype with the same class label $y_i = c(x_i)$, and $d_-^{\mathbf{\Lambda}}(x_i, w)$ is the distance to x_i from the closest prototype with a different class label than y_i.

Note that the numerator is smaller than 0 if the classification of the data point is correct. The smaller the numerator, the greater the 'security' of classification, i.e. the difference of the distance from a correct and wrong prototype. Note that, the 'security' of classification characterizes the hypothesis margin of the classifier. The larger this margin, the more robust is the classification of a data pattern with respect to noise in the input or function parameters (Schneider et al., 2009; Hammer et al., 2002)[3]. A large margin generalization bound for the GMLVQ model was derived in (Schneider et al., 2009). The bound represents a particularly strong result since it is dominated by the margin size and the input space dimensionality does not explicitly occur in it.

The denominator scales the argument of Φ such that it falls in the interval $[-1, 1]$. The learning rules are derived from this cost function by taking the derivatives with respect to the prototype locations w and the distance metric parameters $\mathbf{\Omega}$.

[3] We are thankful to the anonymous reviewer for pointing this out.

Hebbian-like on-line updates are implemented for the closest correct prototype w^+ (i.e. $c(x_i) = c(w^+)$) and the closest incorrect prototype w^- (i.e. $c(x_i) \neq c(w^-)$), along with the metric parameters $\mathbf{\Omega}$. While w^+ is pushed towards the training instance x_i, w^- is pushed away from it (for more details please see (Schneider et al., 2009)).

All previous LVQ variants (with or without metric learning) were designed for nominal classification problems. However, the training examples may be labeled by classes with a natural order imposed on them (e.g classes can represent rank). Pattern recognition problems of classifying examples into ordered classes, namely ordinal classifications, have received a great attention in the recent literatures. They lend themselves to many practical applications as in (Chu et al., 2007; Cardoso et al., 2005). In this paper we would like to extent the LVQ framework to ordinal classification, since the existing LVQ models do not consider the ordinal label information explicitly during learning.

3 The Proposed Ordinal LVQ Classifiers

This section presents two novel methodologies based on LVQ for classifying data with ordinal classes.

We assume that we are given training data $(x_{\mathrm{i}}, y_i) \in \mathbb{R}^m \times \{1, ..., K\}$, where $i = 1, 2, .., , n$, and K is the number of different classes. In the ordinal classification problem, it is assumed that classes are ordered $y_K > y_{K-1} > ... > y_1$, where $>$ denotes the order relation on labels. As in LVQ models, the proposed classifier is parameterized with L prototype-label pairs:

$$W = \{(w_q, k) \mid w_q \in \mathbb{R}^m, q \in \{1, ..., L\}, k \in \{1, ..., K\}\}. \tag{3}$$

We assume that each class $k \in \{1, 2, ..., K\}$, may be represented by P prototypes[4] collected in the set $W(k)$,

$$W(k) = \{w \in W | \, c(w) = k\}, \tag{4}$$

leading to total number of $L = K \cdot P$ prototypes. The prototypes define a classifier by means of a winner-takes-all rule, where a pattern $x_i \in \mathbb{R}^m$ is classified with the label

[4] Of course, this imposition can be relaxed to a variable number of prototypes per class.

of the closest prototype, $c(x_i) = c(w_j)$, $j = \arg\min_l d^{\mathbf{\Lambda}}(x_i, w_l)$, where $d^{\mathbf{\Lambda}}$ denotes the metric (1).

Whereas nominal versions of LVQ aim to position the class prototypes in the input space so that the overall misclassification error is minimized, the proposed ordinal LVQ models adapt the class prototypes so that the average absolute error of class mislabeling is minimized. Loosely speaking, this implies that some class mislabeling (e.g. claiming class $c(w_j) = (k+1)$ instead of class $c(x_i) = k$) will be treated as 'less serious' than other ones (e.g. outputting $c(w_j) = K$ instead of $c(x_i) = 1$), where the 'seriousness' of misclassification will be related to[5] $|c(x_i) - c(w_j)|$. In the next section we describe identification of prototypes to be modified, given each training input x_i.

3.1 Identification of Class Prototypes to be Adapted

The initial step in each training instance x_i, $i = 1, 2, ..., n$, focuses on detecting the 'correct' and 'incorrect' prototype classes (with respect to $c(x_i)$) that will be modified. Subsequently, the correct prototypes will pushed towards x_i, whereas the incorrect ones will be pushed away from x_i.

Correct and Incorrect Prototype Classes

Due to the ordinal nature of labels, for each training instant x_i and prototype w_q, $q = 1, 2, ..., L$, the correctness of prototype's label $c(w_q)$ is measured through the absolute error loss function $H\left(c(x_i), c(w_q)\right)$ (e.g. (Dembczynski et al., 2008)):

$$H\left(c(x_i), c(w_q)\right) = \mid c(x_i) - c(w_q) \mid \tag{5}$$

Given a rank loss threshold L_{min}, defined on the range of the loss function[6], the class prototypes w_q with $H(c(x_i), c(w_q)) \leq L_{min}$ will be viewed as 'tolerably correct', while prototypes with $H(c(x_i), c(w_q)) > L_{min}$ will be classified as 'incorrect'. This is illustrated in Figure 1. The sets of correct and incorrect prototype classes for input x_i hence read:

[5] Of course, other order related costs could be used.

[6] in our case $[0, K-1]$

$$N\left(c(x_i)\right)^+ = \{c(w_q) \in \{1, 2, 3.., , K\} \mid |c(x_i) - c(w_q)| \leq L_{min}\} \tag{6}$$

and

$$N\left(c(x_i)\right)^- = \{c(w_q) \in \{1, 2, 3.., , K\} \mid |c(x_i) - c(w_q)| > L_{min}\}, \tag{7}$$

respectively.

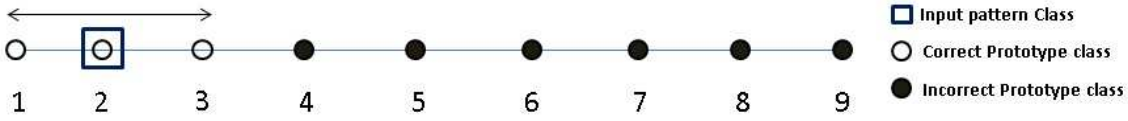


Figure 1: Correct and incorrect prototype classes estimation. Given training pattern $c(x_i) = 2$ indicated with square, and threshold $L_{min} = 1$. White circles are prototypes of correct classes with respect to $c(x_i)$, while black circles indicate prototypes of incorrect classes.

Prototypes to be Adapted

Given a training pattern x_i, the nominal LVQ techniques adapt either the closest prototype or the closest pair of correct/incorrect prototypes. In our case we need to deal with the class prototypes in a different way.

1) **Correct prototypes** with labels in $N(c(x_i))^+$: For correct prototypes it makes sense to push towards x_i only the closest prototype from each class in $N(c(x_i))^+$. The set of correct prototypes to be modified given input x_i reads:

$$W(x_i)^+ = \{w_{z(k)} |\ c(w_{z(k)}) = k \in N^+(c(x_i)),\ z(k) = \arg\min_{l \in W(k)} [d^{\mathbf{\Lambda}}(x_i, w_l)]\} \tag{8}$$

2) **Incorrect prototypes** with labels in $N(c(x_i))^-$: For incorrect prototypes it is desirable to push away from x_i all incorrect prototypes lying in the 'neighbourhood' of x_i. In our case the neighbourhood will be defined as a sphere of radius D under the metric $d^{\mathbf{\Lambda}}$.

$$W(x_i)^- = \{w_z |\ c(w_z) \in N^-(c(x_i)),\ d^{\mathbf{\Lambda}}(x_i, w_z) < D\}. \tag{9}$$

3.2 Prototype Weighting Scheme

Unlike in nominal LVQ, we will need to adapt multiple prototypes, albeit to a different degree. Given a training input x_i, the attractive and repulsive force applied to correct and incorrect prototypes w will decrease and increase, respectively, with growing $H(c(x_i), c(w))$. In addition, for incorrect prototypes w, the repulsive force with diminish with increasing distance form x_i. In the two following sections we describe the prototype adaptation schemes in greater detail.

Given a training pattern x_i, there are two distinct weighting schemes for the correct and incorrect prototypes w in $W(x_i)^+$ and $W(x_i)^-$, respectively.

1) **Weighting correct prototypes $w \in W(x_i)^+$:**
   We propose a Gaussian weighting scheme,
   $$\alpha^+ = \exp\left\{-\frac{(H(c(x_i), c(w)))^2}{2\sigma^2}\right\}, \tag{10}$$
   where, σ is the Gaussian kernel width.

2) **Weighting incorrect prototypes $w \in W(x_i)^-$:**
   Denote by ϵ_{max} the maximum rank loss error within the set $W(x_i)^-$,
   $$\epsilon_{max} = \max_{w \in W(x_i)^-} H(c(x_i), c(w)).$$
   The weight factor α^- for incorrect prototype $w \in W(x_i)^-$ is then calculated as follows:
   $$\alpha^- = \exp\left\{-\frac{(\epsilon_{max} - H(c(x_i), c(w)))^2}{2\sigma^2}\right\} \cdot \exp\left\{-\frac{(d^{\mathbf{\Lambda}}(x_i, w))^2}{2\sigma'^2}\right\}, \tag{11}$$
   where σ' is the Gaussian kernel width for the distance factor in α^-.

These weighting factors will be utilized in two prototype update schemes introduced in the next two sections.

3.3 Ordinal MLVQ (OMLVQ) Algorithm

In this section we generalize the MLVQ algorithm 2.1 to the case of linearly ordered classes. We will refer to this new learning scheme as Ordinal MLVQ (OMLVQ). In particular, there are two main differences between MLVQ and OMLVQ:

- In OMLVQ the order information on classes is utilized to select appropriate multiple prototypes (rather than just the closest one as in MLVQ) to be adapted.
- The ordinal version of MLVQ realizes Hebbian updates for all prototype parameters in $W(x_i)^+$ and $W(x_i)^-$, using the assigned weights $\alpha^{\pm}$. Similarly to MLVQ, each prototype update Δw will be followed by a corresponding metric parameter update $\Delta\mathbf{\Omega}$.

The OMLVQ training algorithm is outlined in greater detail below.

1) **Initialization:** Initialize the prototype positions[7] $w_q \in \mathbb{R}^m$, $q = 1, 2, ..., L$. Initialize the matrix tensor parameter $\mathbf{\Omega}$ by setting it equal to the identity matrix (Euclidean distance).

2) **While** a stopping criterion (in our case the maximum number of training epochs) is not reached do:

    1) Randomly select a training pattern x_i, $i \in \{1, 2, ..., n\}$, with class label $c(x_i)$.

    2) Determine the correct and incorrect classes for x_i, $N\left(c(x_i)\right)^+$ and $N\left(c(x_i)\right)^-$ based on (6) and (7), respectively.

    3) Find collections of prototypes $W(x_i)^+$ and $W(x_i)^-$ to be adapted using (8) and (9).

    4) Assign weight factors[8] $\alpha^{\pm}$ to the selected prototypes (Eq. (10) and (11)).

    5) Update the prototypes from $W(x_i)^+$, $W(x_i)^-$ and the distance metric $\mathbf{\Omega}$ as follows:

        1) $\forall\, w \in W(x)^+$ do:

        $w = w + \eta_w \cdot \alpha^+ \cdot \mathbf{\Lambda} \cdot (x_i - w)$ (w dragged towards x_i)

        $\mathbf{\Omega} = \mathbf{\Omega} - \eta_{\mathbf{\Omega}} \cdot \alpha^+ \cdot \mathbf{\Omega} \cdot (x_i - w)(x_i - w)^T$ ($d^{\mathbf{\Lambda}}(x_i, w)$ is shrinked)

[7] Following (Schneider et al., 2009; Schneider , 2010), the means of P random subsets of training samples selected from each class k, where $k \in \{1, 2, ..., K\}$, are chosen as initial states of the prototypes. Alternatively, one could run a vector quantization with P centers on each class.

[8] For ease of presentation we omit from the notation the classes of the prototypes and the training point.

2) $\forall\, w \in W(x)^-$ do:

$w = w - \eta_w \cdot \alpha^- \cdot \mathbf{\Lambda} \cdot (x_i - w)$ (w pushed away from x_i)

$\mathbf{\Omega} = \mathbf{\Omega} + \eta_{\mathbf{\Omega}} \cdot \alpha^- \cdot \mathbf{\Omega} \cdot (x_i - w)(x_i - w)^T$ ($d^{\mathbf{\Lambda}}(x_i, w)$ is increased).

Here η_w, $\eta_{\mathbf{\Omega}}$ are positive learning rates for prototypes and metric[9], respectively. They decrease monotonically with time as (Darken et al., 1992):

$$\eta_g \leftarrow \frac{\eta_g}{1 + \tau(t-1)} \tag{12}$$

where $g \in \{\mathbf{\Omega}, w\}$, $\tau > 0$ determines the speed of annealing[10], and t indexes the number of training epochs done.

Similarly to the original MLVQ (see section 2.1), to prevent the algorithm from degeneration, $\mathbf{\Omega}$ is normalized after each learning step so that $\sum_i \mathbf{\Lambda}_{ii} = 1$ (Schneider et al., 2009; Schneider , 2010).

3) **End While**

Note that, unlike in the original MLVQ, during the training, adaptation of the prototypes is controlled by the corresponding weight factors $\alpha\pm$ which reflect, **(i)** the class order (see (10), (11)), and **(ii)** the distance of incorrect prototypes from training inputs (see (11)).

3.4 Ordinal GMLVQ (OGMLVQ) Algorithm

This section extends the update rules of the GMLVQ Algorithm (section 2.2) to the case of ordinal classes. The algorithm, referred to as Ordinal GMLVQ (OGMLVQ), will inherit from GMLVQ its cost function (2). There are two main differences between OGMLVQ and GMLVQ:

[9] The initial learning rates are chosen individually for every application through cross-validation. We imposed $\eta_w > \eta_{\mathbf{\Omega}}$, implying slower rate of changes to the metric, when compared with prototype modification. This setting has proven better performance in other matrix relevance learning applications (e.g. (Schneider et al., 2009; Schneider , 2010))

[10] In our experiments τ was set to 0.0001

- For each training pattern x_i, GMLVQ scheme applies Hebbian update for the single closest prototype pair (with the same and different class labels with respect to the label $c(x_i)$ of x_i, see section 2.2). On the other hand, in OGMLVQ there will be updates of $r \geq 1$ prototype pairs from $W(x_i)^+ \times W(x_i)^-$ (see (8) and (9)). This is done in an iterative manner as follows:

  Set $W^{\pm} = W(x_i)^{\pm}$, r=0.
  While ($W^+ \neq \emptyset$ and $W^- \neq \emptyset$)

  1) $r \leftarrow r + 1$.

  2) Construct 'the closest' prototype pair $R_r = (w_a, w_b)$, where

$$a = \arg\min_{l \in W^+} d^{\mathbf{\Lambda}}(x_i, w_l),\ b = \arg\min_{l \in W^-} d^{\mathbf{\Lambda}}(x_i, w_l). \tag{13}$$

  3) Update w_a, w_b and $\mathbf{\Omega}$ (to be detailed later).

  4) $W^+ \leftarrow W^+ \setminus \{w_a\}$, $W^- \leftarrow W^- \setminus \{w_b\}$.

  End While

- In order to control prototype adaptation by their corresponding weight factors $\alpha^{\pm}$ (Eq. (10) and (11)), OGMLVQ scales the metric (1) (used in the original GMLVQ cost function (2)) as

$$\begin{aligned} d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) &= \alpha^+ \cdot d^{\mathbf{\Lambda}}(x_i, w_a) \\ d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b) &= \alpha^- \cdot d^{\mathbf{\Lambda}}(x_i, w_b) \end{aligned} \tag{14}$$

  The OGMLVQ cost function reads:

$$f_{OGMLVQ} = \sum_{i=1}^{n} \sum_{j=1}^{r} \Phi(\mu(x_i, R_j)), \tag{15}$$

  where

$$\mu(x_i, R_j) = \frac{d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) - d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)}{d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)}, \qquad (w_a, w_b) = R_j.$$

  The cost function f_{OGMLVQ} will be minimized with respect to prototypes and metric parameter $\mathbf{\Omega}$ using the steepest descent method. Recall that $d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)$ is

the distance of the data point x_i from the correct prototype w_a, and $d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)$ is the distance from the incorrect prototype w_b and, Φ is a monotonic function set (as in GMLVQ) to the identity mapping.

To obtain the new adaptation rules for the OGMLVQ algorithm, we present derivatives of $\mu(x_i, R_j)$ with respect to the prototype pair $(w_a, w_b) = R_j$ (13) and the metric parameter $\mathbf{\Omega}$.

Derivatives of $\mu(x_i, R_j)$ with respect to the correct prototype w_a,

$$\frac{\partial \mu(x_i, R_j)}{\partial w_a} = \frac{\partial \mu(x_i, R_j)}{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)} \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)}{\partial w_a} = \gamma^+ \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha+}(x_i, w_a)}{\partial w_a},$$

where

$$\begin{aligned}
\gamma^+ &= \frac{\partial \mu(x_i, R_j)}{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)} \\
&= \frac{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)) - (d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) - d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))^2} \\
&= \frac{2d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))^2}
\end{aligned} \tag{16}$$

and

$$\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)}{\partial w_a} = -2\alpha^+ \cdot [\mathbf{\Omega}^T \mathbf{\Omega}](x_i - w_a) = -2\alpha^+ \cdot \mathbf{\Lambda}(x_i - w_a) \tag{17}$$

Derivatives of $\mu(x_i, R_j)$ with respect to the incorrect prototype w_b,

$$\frac{\partial \mu(x_i, R_j)}{\partial w_b} = \frac{\partial \mu(x_i, R_j)}{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)} \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)}{\partial w_b} = \gamma^- \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha-}(x_i, w_b)}{\partial w_b},$$

where

$$\begin{aligned}
\gamma^- &= \frac{\partial \mu(x_i, R_j)}{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)} \\
&= \frac{-(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)) - (d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) - d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))^2} \\
&= \frac{-2d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i, w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b))^2},
\end{aligned} \tag{18}$$

and

$$\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i, w_b)}{\partial w_b} = -2\alpha^- \cdot [\mathbf{\Omega}^T \mathbf{\Omega}](x_i - w_b) = -2\alpha^- \cdot \mathbf{\Lambda}(x_i - w_b) \tag{19}$$

Furthermore, derivatives of $\mu(x_i, R_j)$ with respect to the metric parameter $\mathbf{\Omega}$,

$$\begin{aligned}\frac{\partial \mu(x_i, R_j)}{\partial \mathbf{\Omega}} &= \frac{\left(\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{\partial \mathbf{\Omega}} - \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{\partial \mathbf{\Omega}}\right)\left(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)\right)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b))^2} \\ &- \frac{\left(\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{\partial \mathbf{\Omega}} + \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{\partial \mathbf{\Omega}}\right)\left(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) - d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)\right)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b))^2} \end{aligned} \tag{20}$$

$$\begin{aligned}&= \frac{2d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b))^2} \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{\partial \mathbf{\Omega}} \\ &+ \frac{-2d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{(d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a) + d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b))^2} \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{\partial \mathbf{\Omega}} \end{aligned} \tag{21}$$

using (16) and (18) then,

$$\frac{\partial \mu(x_i, R_j)}{\partial \mathbf{\Omega}} = \gamma^+ \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{\partial \mathbf{\Omega}} + \gamma^- \cdot \frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{\partial \mathbf{\Omega}} \tag{22}$$

where

$$\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^+}(x_i,w_a)}{\partial \mathbf{\Omega}} = 2\alpha^+ \cdot [\mathbf{\Omega}\,(x_i - w_a)(x_i - w_a)^T] \tag{23}$$

and

$$\frac{\partial d^{\mathbf{\Lambda}}_{\alpha^-}(x_i,w_b)}{\partial \mathbf{\Omega}} = 2\alpha^- \cdot [\mathbf{\Omega}\,(x_i - w_b)(x_i - w_b)^T] \tag{24}$$

Note that the OGMLVQ cost function (15) is a sum of r "weighted versions" of the GMLVQ cost function (Schneider et al., 2009) (eq. (2)). The only difference is that the distances from data points to prototypes are linearly scaled by factors $\alpha^{\pm}$ (see eq. (14)). As such, the OGMLVQ cost function inherits all the discontinuity problems of the GMLVQ cost functional at receptive field boundaries of the prototypes. As argued in (Schneider et al., 2009), the GMLVQ prototype and metric updates resulting from gradient descent on the GMLVQ cost function are valid whenever the metric is differentiable (see also (Hammer et al., 2002, 2005)). Using delta function (as derivative of the Heaviside function) the argument can be made for cost functions rewritten with respect to full 'reasonable' distributions on the input space (with continuous support) (Schneider et al., 2009). Since weighting of distances in individual GMLVQ cost functions that make up the OGMLVQ cost function preserves differentiability of the metric

and because the OGMLVQ cost function is a sum of such individual weighted GMLVQ cost functions, the theoretical arguments made about updates from the GMLVQ cost function also fall through in the case of the OGMLVQ cost function.

We summarize the OGMLVQ algorithm below:

1) **Initialization:** Initialize the prototype positions $w_q \in \mathbb{R}^m$, $q = 1, 2, ..., L$.
   Initialize the matrix tensor parameter $\mathbf{\Omega}$ by setting it equal to the identity matrix (Euclidean distance).

2) **While** a stopping criterion (in our case the maximum number of training epochs) is not reached do:

   1) Randomly select a training pattern x_i, $i \in \{1, 2, ..., n\}$, with class label $c(x_i)$.

   2) Determine the correct and incorrect classes for x_i, $N\left(c(x_i)\right)^+$ and $N\left(c(x_i)\right)^-$ based on (6) and (7), respectively.

   3) Find collections of prototypes $W(x_i)^+$ and $W(x_i)^-$ to be adapted using (8) and (9).

   4) Assign weight factors $\alpha^\pm$ to the selected prototypes (Eq. (10) and (11)).

   5) Set $W^\pm = W(x_i)^\pm$, r=0.
      While ($W^+ \neq \emptyset$ and $W^- \neq \emptyset$)

      1) $r \leftarrow r + 1$.

      2) Construct 'the closest' prototype pair $R_r = (w_a, w_b)$ as in (13).

      3) Update the prototypes position:

$$\Delta w_a = 2\eta_w \cdot \gamma^+ \cdot \alpha^+ \mathbf{\Lambda}(x_i - w_a)$$

(w_a dragged towards x_i)

$$\Delta w_b = 2\eta_w \cdot \gamma^- \cdot \alpha^- \mathbf{\Lambda}(x_i - w_b)$$

(w_b pushed away[11] from x_i)

[11] Note that unlike γ^+, γ^- is negative.

4) update the metric parameter $\mathbf{\Omega}$,

$$\Delta\mathbf{\Omega} = -2\eta_{\mathbf{\Omega}} \cdot [\gamma^{+}\alpha^{+}\mathbf{\Omega}(x_i - w_a)(x_i - w_a)^T + \gamma^{-}\alpha^{-}\mathbf{\Omega}(x_i - w_b)(x_i - w_b)^T]$$

where γ^{+} and γ^{-} are given in (16) and (18), respectively.
η_w, $\eta_{\mathbf{\Omega}}$ are the learning rates for prototypes and metric respectively, and they normally decrease throughout the learning as given in (12).
Each $\mathbf{\Omega}$ update is followed by a normalization step as described in the OMLVQ algorithm (see section 3.3).

5) $W^{+} \leftarrow W^{+} \setminus \{w_a\}$, $W^{-} \leftarrow W^{-} \setminus \{w_b\}$.

End While

3) **End While**

During the adaptation, distances between the training point x_i and the correct prototypes in W^{+} are on average decreased, in line with the aim of minimizing the rank loss error. Conversely, the average distances between x_i and the incorrect prototypes in W^{-} are increased, so that the risk of higher ordinal classification error (due to the high rank loss error of incorrect prototypes) is diminished.

Note that while OMLVQ is a heuristic extension of MLVQ, updating each prototype independently of the others, the OGMLVQ is an extension of GMLVQ, with parameter updates following in a principled manner from a well-defined cost function. In OGMLVQ the prototypes are updated in pairs as explained above.

4 Experiments

We evaluated the performance of the proposed ordinal regression LVQ methods through a set of experiments conducted on two groups of data sets: eight benchmark ordinal regression data sets[12] (Sun et al., 2010; Chu et al., 2007; Li et al., 2007; Lin et al., 2012; Xia et al., 2007) and two real-world ordinal regression data sets (Lin et al., 2012).

[12] Regression data sets are available at `http://www.gatsby.ucl.ac.uk/~chuwei/ordinalregression.html`

The ordinal LVQ models, OMLVQ and OGMLVQ, were assessed against their nominal (non-ordinal) counterparts, MLVQ and GMLVQ, respectively. The ordinal LVQ models were also compared with benchmark ordinal regression approaches.

The experiments utilized three evaluation metrics to measure accuracy of predicted class $\hat{y}$ with respect to true class y on a test set:

1) **Mean Zero-one Error (MZE)** - (misclassification rate) the fraction of incorrect predictions,

$$MZE = \frac{\sum_{i=1}^{v} I(y_i \neq \hat{y}_i)}{v}.$$

where v is the number of test examples and $I(y_i \neq \hat{y}_i)$ denotes the indicator function returning 1 if the predicate holds and 0 otherwise.

2) **Mean Absolute Error (MAE)** - the average deviation of the prediction from the true rank,

$$MAE = \frac{\sum_{i=1}^{v} |y_i - \hat{y}_i|}{v}.$$

3) **Macroaveraged Mean Absolute Error (MMAE)** (Baccianella et al., 2009) - macroaveraged version of Mean Absolute Error - it is a weighted sum of the classification errors across classes,

$$MMAE = \frac{1}{K} \sum_{k=1}^{K} \frac{\sum_{y_i=k} |y_i - \hat{y}_i|}{v_k}.$$

where K is the number of classes and v_k is the number of test points whose true class is k. The macroaveraged MAE is typically used in imbalanced ordinal regression problems as it emphasizes errors equally in each class.

For comparison purposes and with respect to the eight benchmark ordinal regression data sets we conducted the same pre-processing as described in (Sun et al., 2010; Chu et al., 2007; Li et al., 2007; Lin et al., 2012; Xia et al., 2007). Data labels were discretized into ten ordinal quantities using the equal-frequency binning. Hence, the eight benchmark ordinal regression data sets are balanced with respect to their classes distribution. The input vectors were normalized to have zero mean and unit variance. Each data set was randomly partitioned into training/test splits as recorded in Table 1. The partitioning was repeated 20 times independently, yielding 20 re-sampled training/test

sets. For these class-balanced data sets, the experimental evaluations were done using the MZE and MAE measures.

The two real-world ordinal ranking problems were represented by two data sets: *cars* and the red wine subset *redwine* of the wine quality set from the UCI machine learning repository (Hettich et al., 1998). For fair comparison, we followed the same experimental settings as in (Lin et al., 2012). We randomly split 75% of the examples for training and 25% for testing, as recorded in Table 1, and conducted 20 runs of such a random splits. The cars problem intends to rank cars to four conditions (unacceptable, acceptable, good, very good), while the redwine problem ranks red wine samples to 11 different levels (between 0 and 10, however, the actual data only contains samples with ranks between 3 and 8). It is worth mentioning that the two data sets are highly imbalanced (with respect to their classes distribution). In the *cars* data set the class distribution (percentage of instances per class) is as follows: unacceptable - 70%, acceptable - 22%, good - 4% and very good - 4%. The *redwine* data set has the following class distribution: 3 - 1%, 4 - 3%, 5 - 43%, 6 - 40%, 7 - 12% and 8 - 1%. Real-world ordinal regression data sets are often severly imbalanced, i.e. are likely to have different class populations at their class order, and (unlike in many previous ordinal classification studies) ordinal classification algorithms should be examined in both balanced and imbalanced class distribution cases[13]. As shown in (Baccianella et al., 2009), testing a classifier on imbalanced data sets using standard evaluation measures (e.g. MAE) may be insufficient. Therefore, along with the MZE and MAE evaluation measures, we examined our prototype based models with the Macroaveraged Mean Absolute Error (MMAE)(Baccianella et al., 2009) that is specially designed for evaluating classifiers operating on imbalanced data sets.

On each data set, the algorithm parameters were chosen through 5-fold cross-validation on the training set. Test errors were obtained using the optimal parameters found for each data re-sampling, and were averaged over the 20 trials (runs). We also report standard deviations across the 20 trails.

[13] We are thankful to the anonymous reviewer for pointing this out.

Table 1: Ordinal regression data sets partitions

Data set	Dimension	Training	Testing
Pyrimidines	27	50	24
MachineCpu	6	150	59
Boston	13	300	206
Abalone	8	1000	3177
Bank	32	3000	5182
Computer	21	4000	4182
California	8	5000	15640
Census	16	6000	16784
Cars	8	1296	432
Redwine	11	1200	399

4.1 Comparison with MLVQ and GMLVQ

This section evaluates performance of the proposed OMLVQ and OGMLVQ algorithms against their standard nominal versions MLVQ and GMLVQ. For the eight benchmark ordinal regression data sets, the MZE and MAE results, along with standard deviations (represented by error bars), across 20 runs are shown in Figures 2 and 3, respectively. Furthermore, the MZE, MAE and MMAE results, along with standard deviations (represented by error bars) across 20 runs, for the two real-world ordinal regression data sets are presented in Figures 4.a, 4.b and 4.c, respectively.

The results in general confirm that the proposed ordinal LVQ models achieve better performance in terms of MZE, MAE and MMAE rates than their standard (nominal) LVQ counterparts. On average, across the eight benchmark ordinal regression data sets the OMLVQ algorithm outperforms the baseline MLVQ by relative improvement of 10% and 18% on MZE and MAE, repectively. Furthermore, OGMLVQ achieves relative improvements over the baseline GMLVQ of 5% and 15% on MZE and MAE, respectively. For the two real-world ordinal regression data sets, on average the OMLVQ algorithm outperforms the baseline MLVQ by relative improvement of 41%, 48% and 46% on MZE, MAE and MMAE, repectively. And the OGMLVQ achieves relative

improvements over the baseline GMLVQ of of 14%, 15% and 8% on MZE, MAE and MMAE, repectively.

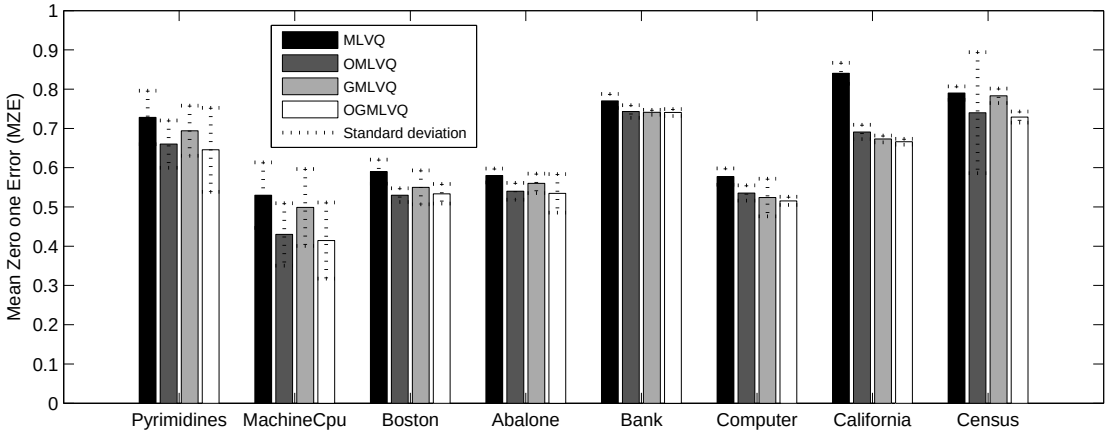


Figure 2: MZE results for the eight benchmark ordinal regression data sets.

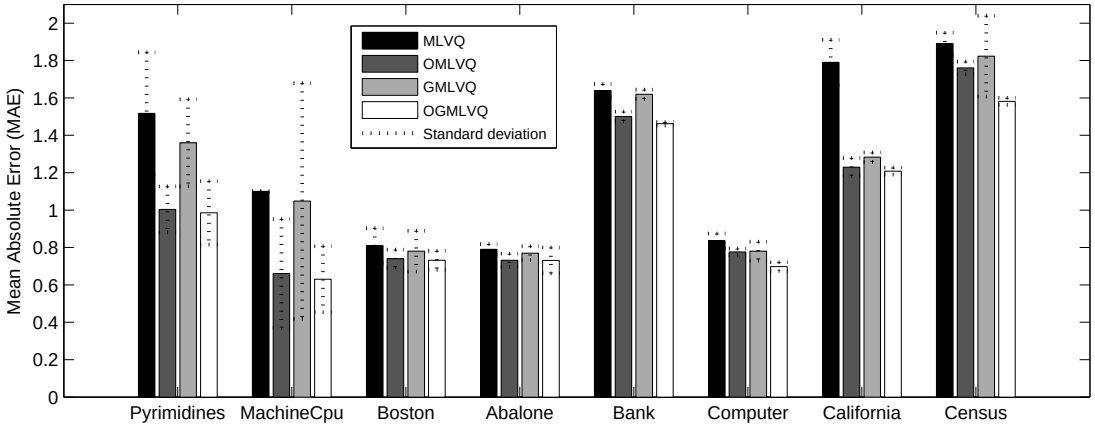
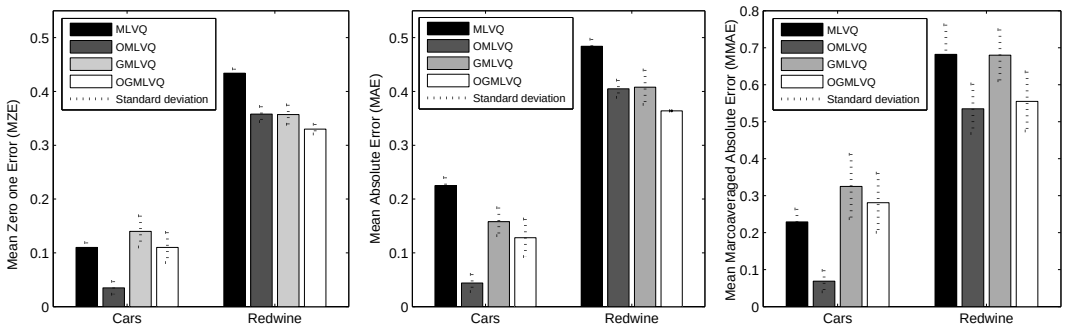


Figure 3: MAE results for the eight benchmark ordinal regression data sets.

4.2 Comparison with Benchmark Ordinal Regression Approaches

This section compares (in terms of MZE, MAE and MMAE) the proposed ordinal LVQ approaches (OMLVQ and OGMLVQ) against five benchmark ordinal regression methods: two threshold SVM based models (SVOR-IMC and SVOR-EXC (Chu et al., 2007) with the Gaussian kernel), two reduction frameworks (the SVM based model RED-SVM with perceptron kernel (Li et al., 2007; Lin et al., 2012) and the Weighted Log-



(a) MZE Results (b) MAE Results (c) MMAE Results

Figure 4: MZE, MAE and MMAE results for the the two real-world ordinal regression data sets shown in (a), (b) and (c), respectively.

itBoost (Xia et al., 2007)), and a non SVM based - Kernel Discriminant Learning for Ordinal Regression method (KDLOR (Sun et al., 2010)).

The first comparison was conducted on eight benchmark ordinal ranking data sets used in (Chu et al., 2007; Sun et al., 2010; Li et al., 2007; Lin et al., 2012; Xia et al., 2007). We used the same data set pre-processing and experimental settings as in (Chu et al., 2007; Sun et al., 2010; Li et al., 2007; Lin et al., 2012; Xia et al., 2007).

MZE and MAE test results[14], along with standard deviations over 20 training /test re-samplings, are listed in Tables 2 and 3, respectively[15]. We use bold face to indicate the lowest average error value among the results of all algorithms.

In comparison with other methods and with respect to the eight benchmark ordinal ranking data sets, OGMLVQ and OMLVQ algorithms achieve the lowest MZE results on four data sets, with OGMLVQ being lowest in *Pyrimidines*, *MachineCPU*, and *Abalone* data sets, and OMLVQ in *Boston* data set. Furthermore, OGMLVQ and OMLVQ attain the lowest MAE for three data sets *Pyrimidines*, *MachineCPU*, and *Abalone*, with OGMLVQ being slightly better than OMLVQ on all data sets. Note that on *Abalone* data set, both ordinal LVQ models beat the competitors out-of-sample by a

[14] The underlying eight benchmark data sets are considered as balanced (with respect to their class distribution). Thus, we did not examine their MMAE results.

[15] MZE results of the Weighted LogitBoost reduction model is not listed because only MAE of this algorithm was recorded in (Xia et al., 2007).

Table 2: Mean Zero-one Error (MZE) results along with standard deviations, ($\pm$) across 20 training/test re-sampling, for the ordinal LVQ models (OMLVQ and OGMLVQ) and the benchmark algorithms KDLOR reported in (Sun et al., 2010), SVOR-IMC (with Gaussian kernel), SVOR-EXC (with Gaussian kernel) reported in (Chu et al., 2007), RED-SVM (with Perceptron kernel) reported in (Lin et al., 2012). The best results are marked with bold font.

Data set	KDLOR	SVOR-IMC	SVOR-EXC	RED-SVM	OMLVQ	OGMLVQ
Pyrimidines	0.739± (0.050)	0.719± (0.066)	0.752± (0.063)	0.762± (0.021)	0.660 ± (0.060)	**0.645± (0.106)**
MachineCpu	0.480± (0.010)	0.655± (0.045)	0.661± (0.056)	0.572± (0.013)	0.431± (0.079)	**0.415± (0.096)**
Boston	0.560± (0.020)	0.561± (0.026)	0.569± (0.025)	0.541± (0.009)	**0.532± (0.017)**	0.534± (0.024)
Abalone	0.740± (0.020)	0.732± (0.007)	0.736± (0.011)	0.721± (0.002)	0.545± (0.021)	**0.532± (0.049)**
Bank	0.745± (0.0025)	0.751± (0.005)	**0.744± (0.005)**	0.751± (0.001)	0.756 ± (0.016)	0.750± (0.008)
Computer	0.472± (0.020)	0.473± (0.005)	0.462± (0.005)	**0.451± (0.002)**	0.535± (0.019)	0.510± (0.010)
California	0.643± (0.005)	0.639± (0.003)	0.640± (0.003)	**0.613± (0.001)**	0.710± (0.018)	0.680± (0.007)
Census	0.711± (0.020)	0.705± (0.002)	0.699± (0.002)	**0.688± (0.001)**	0.754± (0.154)	0.735± (0.014)

large margin. However, relative to the competitors, OMLVQ and OGMLVQ exhibit the worst performance on three data sets (*Computer*, *California* and *Census*), and comparable performances on the remaining data sets *Boston* and *Bank*. Note that on the three data sets where the ordinal LVQ methods were beaten by the competitors, the original LVQ methods performed poorly as well (see Figures 2 and 3). We hypothesize that the class distribution structure of those data sets may not be naturally captured by the prototype based methods.

We also examined the performance of our prototype based models, using the two real-world ordinal ranking problems, against two SVM-based ordinal regression approaches (SVOR-IMC (Chu et al., 2007) with the Gaussian kernel and RED-SVM with perceptron kernel (Li et al., 2007; Lin et al., 2012))[16].

The MZE and MAE test results of the *cars* and *redwine* data sets for the two compared algorithms were reported in (Lin et al., 2012). MZE, MAE and MMAE test results over 20 training/test random re-samplings are listed in Table 4[17]. We use bold face to indicate the lowest average error value among the results of all algorithms.

In comparison with SVOR-IMC (Chu et al., 2007) and RED-SVM (Li et al., 2007; Lin et al., 2012)), on the two real-world ordinal regression data sets (*cars* and *redwine*), the prototype based models for ordinal regression (OMLVQ and OGMLVQ) show a competitive performance in MZE and MAE. For the *cars* data set, among the compared algorithms the OMLVQ model is performing the best with respect to the MZE and MAE results. For the *redwine* data set, the RED-SVM yields the best MZE/MAE performance. The OMLVQ and OGMLVQ models are slightly worse than RED-SVM, but better than the SVM-IMC algorithm.

4.3 Sensitivity of the Ordinal LVQ Models to the Correct Region

As specified in section 3.1, the rank loss threshold L_{min} defines the sets of correct and incorrect prototype classes. Given classes $1, 2, ..., K$, the value of the L_{min} is defined on the range of the absolute error loss function, i.e. $[0, K-1]$.

The following experiment investigates the sensitivity of the presented models to the choice of the correct region[18], i.e. the value of L_{min}. The experiment was conducted on four data sets with different number of classes K (*Pyrimidines* and *Abalone* with $K = 10$; *cars* and *redwine* with $K = 4$ and $K = 6$, respectively). Using settings of the best-performing models from the previous experiments, we examined sensitivity of the

[16] Unfortunately we have not been able to obtain codes for the two other ordinal regression algorithms considered in this study (Weighted LogitBoost (Xia et al., 2007) and KDLOR (Sun et al., 2010)).

[17] MMAE results of the SVM based models are not listed because only MZE and MAE of these algorithms were recorded in (Lin et al., 2012). Furthermore, MZE of the SVOR-IMC with Gaussian kernel algorithm were not reported in (Lin et al., 2012).

[18] We are thankful to the anonymous reviewer for suggesting this experiment.

model performance with respect to varying L_{min} in the range $[L^*_{min}-1, L^*_{min}+1]$, where L^*_{min} denotes the 'optimal' value of L_{min} found using cross-validation as described above.

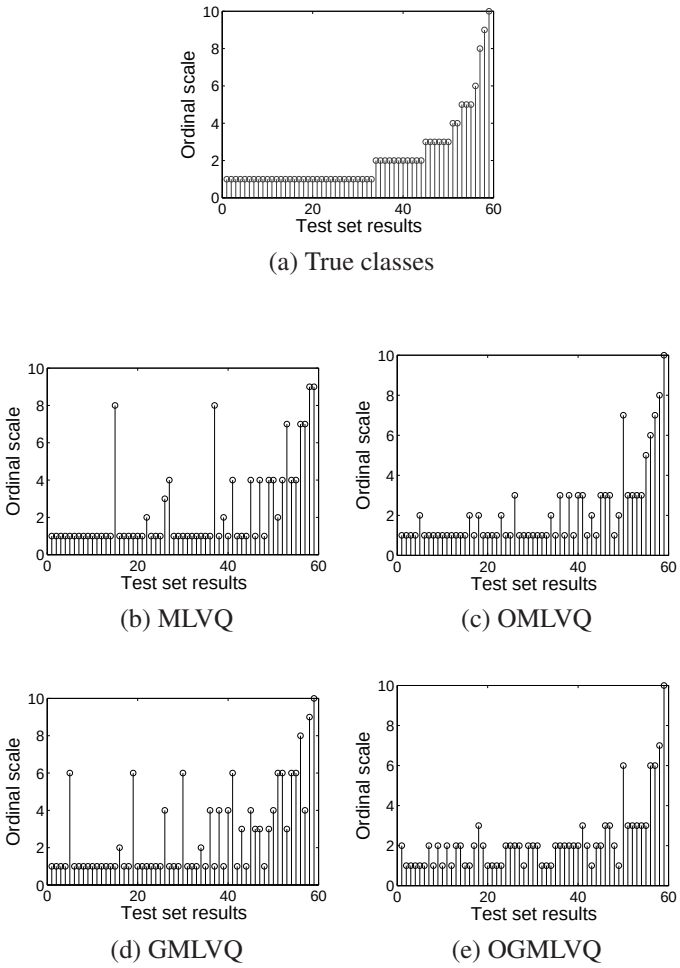
The MAE and MMAE[19] results are presented in Tables 5 and 6, respectively. As expected, sensitivity with respect to variations in L_{min} is much greater if the number of classes is small (e.g. *cars* and *redwine*). In such cases, setting the 'right' value' of L_{min} is crucial. Not surprisingly, for small number of classes the selected value of L_{min} was 0. Interestingly, OGMLVQ appears to be more robust to changes in L_{min} than OMLVQ. We speculate that this is so since OMLVQ in each training step updates all selected correct and incorrect prototypes independently of each other. On the other hand, OGMLVQ updates only the closest pair of correct and incorrect prototypes, affecting potentially a smaller number of prototypes.

4.4 Discussion

OGMLVQ slightly outperforms OMLVQ in almost all cases. This may be due to principled adaptation formulation through the novel cost function (15). Interestingly enough, this is also reflected in the nominal classification case, where GLVQ (later extended to GMLVQ) has been shown to be superior to LVQ1 (later extended to MLVQ) (Sato et al., 1996).

As expected, ordinal LVQ methods demonstrate stronger improvements over their nominal counterparts in terms of MAE, rather than MZE. As an example, this is illustrated in Figure 5 obtained on a *MachineCpu* test set. The figure compares the true class labels in the selected test set (a) against the predicted ones generated by MLVQ, OMLVQ, GMLVQ and OGMLVQ ((b), (c), (d) and (e), respectively). Although there are several misclassifications by our ordinal LVQ methods (OMLVQ and OGMLVQ), they incorporate less deviations (from their true ordinal label) when compared to the deviations occurring in the MLVQ and GMLVQ misclassifications. Clearly, the ordinal LVQ schemes efficiently utilize the class order information during learning, thus improving the MAE performance.

[19] The MMAE results of the *Pyrimidines* and *Abalone* data sets were not assessed as they are considered as balanced data sets, and hence their MAE and MMAE results coincide.



(a) True classes

(b) MLVQ

(c) OMLVQ

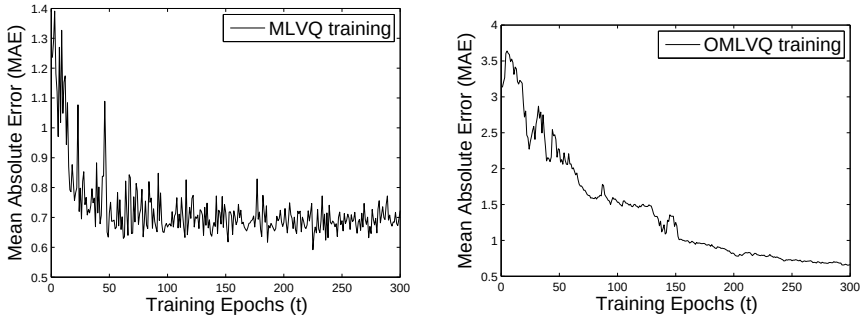
(d) GMLVQ

(e) OGMLVQ

Figure 5: Ordinal prediction results of a single example run in *MachineCpu* data set (true labels in (a)) obtained by MLVQ, OMLVQ, GMLVQ and OGMLVQ shown in (b),(c),(d) and (e), respectively.

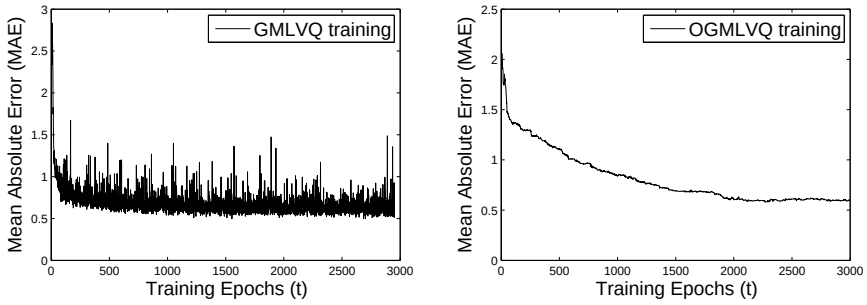
Interestingly enough, we observed that reshaping the class prototypes in the ordinal LVQ methods by explicit use of the class order information stabilizes the training substantially, when compared to the nominal LVQ methods. Provided the class distribution in the data space respects the class order, the class prototypes of ordinal LVQ will quickly reposition to reflect this order. Then most misclassifications that need to be acted on during training have low absolute error, i.e. most misclassifications happen on the border of receptive fields of ordered prototypes with small absolute differences between the classes of data points and those of their closest prototypes. This stabilizes the training in that only relatively small prototype updates are necessary. In nominal LVQ, where the order of classes is not taken into account during training, larger jumps

in absolute error can occur. For example in Figures 6 and 7 we show evolution of MAE error rates as the training progresses (measured in training epochs) for a single run of (O)MLVQ and (O)GMLVQ on the *Abalone* and *Boston* data sets, respectively. The same training sample and similar experimental settings for MLVQ and OMLVQ, as well as for GMLVQ and OGMLVQ were used.



(a) MLVQ (b) OMLVQ

Figure 6: Evolution of MAE in the course of training epochs (t) in the *Abalone* training set obtained by the MLVQ, OMLVQ algorithms, in (a) and (b), respectively.



(a) GMLVQ (b) OGMLVQ

Figure 7: Evolution of MAE in the course of training epochs (t) in the *Boston* training set obtained by the GMLVQ, OGMLVQ algorithms, in (a) and (b), respectively.

5 Conclusion

This paper introduced two novel prototype-based learning methodologies, especially tailored for classifying data with ordered classes. Based on the existing nominal LVQ

methods with metric learning, Matrix LVQ (MLVQ) and Generalized MLVQ (GMLVQ) (Schneider et al., 2009; Schneider , 2010), we proposed two new ordinal LVQ methodologies - Ordinal MLVQ (OMLVQ) and Ordinal GMLVQ (OGMLVQ).

Unlike in nominal LVQ, in ordinal LVQ the class order information is utilized during training in selection of the class prototypes to be adapted, as well as in determining the exact manner in which the prototypes get updated. In particular, the prototypes are adapted so that the ordinal relations amongst the prototype classes are preserved, reflected in reduction of the overall mean absolute error. Whereas in the OMLVQ approach the prototypes are adapted independently of each other, in the OGMLVQ approach the prototypes are updated in pairs based on minimization of a novel cost function.

Experimental results on eight benchmark data sets and two real-world imbalanced data sets empirically verify the effectiveness of our ordinal LVQ frameworks when compared with their standard nominal LVQ versions. The mean zero-one error (MZE), mean absolute error (MAE) and macroaveraged mean absolute error (MMAE) (in case of imbalanced data sets) rates of the proposed methods were considerably lower, with more pronounced improvements on the MAE (in case of balanced data sets) and MAE, MMAE rates (in case of imbalanced data sets) when compared to the MZE rate. In addition, our ordinal models exhibit more stable learning behavior when compared to their nominal counterparts. Finally, in comparison with existing benchmark ordinal regression methods, our ordinal LVQ frameworks attained a competitive performance in terms of MZE and MAE measurements.

References

Chu, W., & Keerthi, S. S. (2007). Support vector ordinal regression. *Neural Computation*, *19(3)*, 792–815.

Cardoso, J. S., Pinto da Costa, J. F., & Cardoso, M. J. (2005). Modelling ordinal relations with SVMs: An application to objective aesthetic evaluation of breast cancer conservative treatment. *Neural Networks*, *18(5-6)*, 808–817.

Frank, E., & Hall, M. (2001). A simple approach to ordinal classification. *Proceedings of the 12th European Conference on Machine Learning*, 145–156. Springer-Verlag.

Waegeman, W., & Boullart, L. (2006). An ensemble of weighted support vector machines for ordinal regression. *Transactions on Engineering, Computing and Technology*, *12*, 71–75.

Shashua, A., & Levin, A. (2002). Ranking with large margin principle: Two approaches. Becker, S., Thrun, S., & Obermayer, K. (Eds.) *Advances in Neural Information Processing Systems 15*, 937–944. Vancouver, British Columbia, Canada: MIT Press.

Li, L., & Lin, H. (2007). Ordinal regression by extended binary classification. Schölkopf, B., Platt, J. C., & Hofmann, T. (Eds.). *Advances in Neural Information Processing Systems 19*, 865–872. Vancouver, British Columbia, Canada: MIT Press.

Lin, H., & Li, L. (2012). Reduction from Cost-Sensitive Ordinal Ranking to Weighted Binary Classification. *Neural Computation*, *24(5)*, 1329-1367.

Xia, F., Zhou, L., Yang, Y., & Zhang, W. (2007). Ordinal regression as multiclass classification. *International Journal of Intelligent Control and Systems*, *12(3)*, 230–236.

Sun, B. Y., Li, J., Wu, D. D., Zhang, X. M., & Li, W. B. (2010). Kernel discriminant learning for ordinal regression. *IEEE Transactions on Knowledge and Data Engineering*, *22(6)*, 906–910.

Kohonen, T. (1986). Learning vector quantization for pattern recognition *Technical Report. TKKF-A601* Laboratory of Computer and Information Science, Department of Technical Physics, Helsinki University of Technology. Espoo, Finland.

Kohonen, T. (1998). Learning vector quantization. *The handbook of brain theory and neural networks*, 537–540. Cambridge, MA, USA: MIT Press.

Schneider, P., Biehl, M., & Hammer, B. (2009). Adaptive relevance matrices in learning vector quantization. *Neural Computation*, *21(12)*, 3532–3561.

Schneider, P. (2010). Advanced methods for prototype-based classification. (Doctoral Dissertation). Available from http://irs.ub.rug.nl/ppn/327245379, *University of Groningen*.

Sato, A., & Yamada, K. (1996). Generalized learning vector quantization. M. C. M. D. S. Touretzky and M. E. Hasselmo (Eds.) *Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press*, *8*, 423–429.

Hammer, B., & Villmann, T. (2002). Generalized relevance learning vector quantization. *Neural Networks. Oxford, UK, Elsevier Science Ltd*, *15(8-9)*, 1059–1068.

Dembczynski, K., Kotlowski, W., & Slowinski, R. (2008). Ordinal classification with decision rules. *Proceedings of the 3rd ECML/PKDD International Conference on Mining Complex Data*, 169–181. Warsaw, Poland: Springer-Verlag.

Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, 3–12. IEEE Press.

Hammer, B., Strickert, M., & Villmann, T. (2005). On the generalization ability of GRLVQ networks. *Neural Processing Letters. Hingham, MA, USA, Kluwer Academic Publishers*, *21(2)*, 109–120.

Baccianella, S., Esuli, A., & Sebastiani, F. (2009). Evaluation measures for ordinal regression. *Ninth International Conference on Intelligent Systems Design and Applications*, 283–287. IEEE Computer Society.

Hettich, S., Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases. Available online at http://archive.ics.uci.edu/ml/.

Table 3: Mean Absolute Error (MAE) results, along with standard deviations (±) across 20 training/test re-sampling, for the ordinal LVQ models (OMLVQ and OGMLVQ) and the benchmark algorithms KDLOR reported in (Sun et al., 2010), SVOR-IMC (with Gaussian kernel), SVOR-EXC (with Gaussian kernel) reported in (Chu et al., 2007), RED-SVM (with Perceptron kernel) reported in (Lin et al., 2012), Weighted Logit-Boost, reported in (Sun et al., 2010). The best results are marked with bold font.

Data set	KDLOR	SVOR-IMC	SVOR-EXC	RED-SVM	Weighted LogitBoost	OMLVQ	OGMLVQ
Pyrimidines	1.1± (0.100)	1.294± (0.204)	1.331± (0.193)	1.304± (0.040)	1.271± (0.205)	1.004± (0.123)	**0.985± (0.169)**
MachineCpu	0.690± (0.015)	0.990± (0.115)	0.986± (0.127)	0.842± (0.022)	0.800± (0.087)	0.660± (0.291)	**0.630± (0.176)**
Boston	**0.700± (0.035)**	0.747± (0.049)	0.773± (0.049)	0.732± (0.013)	0.816± (0.056)	0.742± (0.048)	0.731± (0.050)
Abalone	1.400± (0.050)	1.361± (0.013)	1.391± (0.021)	1.383± (0.004)	1.457± (0.014)	0.732± (0.035)	**0.731± (0.068)**
Bank	1.450± (0.020)	**1.393± (0.011)**	1.512± (0.017)	1.404± (0.002)	1.499± (0.016)	1.501± (0.025)	1.462± (0.009)
Computer	0.601± (0.025)	0.596± (0.008)	0.602± (0.009)	**0.565± (0.002)**	0.601± (0.007)	0.776± (0.018)	0.698± (0.023)
California	0.907± (0.004)	1.008± (0.005)	1.068± (0.005)	0.940± (0.001)	**0.882± (0.009)**	1.238± (0.048)	1.208± (0.018)
Census	1.213± (0.003)	1.205± (0.007)	1.270± (0.007)	1.143± (0.002)	**1.142± (0.005)**	1.761± (0.033)	1.582± (0.018)

Table 4: Mean Zero-one Error (MZE), Mean Absolute Error (MAE) and Macroaveraged Mean Absolute Error (MMAE) results on the real-world *cars* and *redwine* data sets, along with standard deviations, ($\pm$) across 20 training/test re-sampling, for the ordinal LVQ models (OMLVQ and OGMLVQ) and the benchmark algorithms (SVOR-IMC with Gaussian kernel and RED-SVM with Perceptron kernel) reported in (Lin et al., 2012). The best results are marked with bold font.

Data set	Algorithm	MZE	MAE	MMAE
Cars	SVOR-IMC	N/A	0.051±(0.002)	N/A
	RED-SVM	0.064±(0.003)	0.061±(0.003)	N/A
	OMLVQ	**0.035±(0.012)**	**0.044±(0.016)**	**0.069±(0.029)**
	OGMLVQ	0.111±(0.029)	0.128±(0.035)	0.281±(0.080)
Redwine	SVOR-IMC	N/A	0.429±(0.004)	N/A
	RED-SVM	**0.327±(0.005)**	**0.357±(0.005)**	N/A
	OMLVQ	0.358±(0.014)	0.405±(0.016)	**0.535±(0.067)**
	OGMLVQ	0.331±(0.009)	0.364±(0.014)	0.555±(0.083)

Table 5: Mean Absolute Error (MAE) results, along with standard deviations ($\pm$) across 20 training/test re-sampling, obtained using varying number of rank loss threshold ($(L_{min}-1)$, (L_{min}) and $(L_{min}+1)$), on four ordinal regression data sets. Note that, the value of L_{min} is determined using a cross validation procedure on each of the four examined data sets. The best results are marked with bold font.

Data set	K	L_{min}	Algorithm	MAE $(L_{min}-1)$	MAE (L_{min})	MAE $(L_{min}+1)$
Cars	4	0	OMLVQ	N/A	**0.044±(0.016)**	0.403±(0.027)
		0	OGMLVQ	N/A	**0.128±(0.035)**	0.324±(0.034)
Redwine	6	0	OMLVQ	N/A	**0.405±(0.016)**	0.800±(0.080)
		0	OGMLVQ	N/A	**0.364±(0.014)**	0.440±(0.019)
Pyrimidines	10	1	OMLVQ	1.274±(0.177)	**1.004± (0.123)**	1.300±(0.168)
		1	OGMLVQ	1.162±(0.199)	**0.985±(0.169)**	1.062±(0.130)
Abalone	10	1	OMLVQ	0.885±(0.082)	**0.732±(0.035)**	0.901±(0.104)
		1	OGMLVQ	0.740±(0.011)	**0.731±(0.068)**	0.886±(0.034)

Table 6: Macroaveraged Mean Absolute Error (MMAE) results, along with standard deviations ($\pm$) across 20 training/test re-sampling, obtained using varying number of rank loss threshold ($(L_{min}-1)$, (L_{min}) and $(L_{min}+1)$), on two ordinal regression data sets. Note that, the value of L_{min} is determined using a cross validation procedure on each of the four examined data sets. The best results are marked with bold font.

Data set	K	L_{min}	Algorithm	MMAE $(L_{min}-1)$	MMAE (L_{min})	MMAE $(L_{min}+1)$
Cars	4	0	OMLVQ	N/A	**0.069±(0.029)**	0.268±(0.036)
		0	OGMLVQ	N/A	**0.281±(0.080)**	0.390±(0.062)
Redwine	6	0	OMLVQ	N/A	**0.535±(0.067)**	0.781±(0.145)
		0	OGMLVQ	N/A	**0.555±(0.083)**	0.678±(0.071)