Ordinal Regression based on Learning Vector Quantization

Fengzhen Tang[a,b,*], Peter Tiňo[b,]

[a]*Shenyang Institute of Automation, Chinese Academy of Sciences, No.114, Nanta Street, Shenyang, Liaoning Province, 110016, China*
[b]*School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK*

Abstract

Recently, ordinal regression, which predicts categories of ordinal scale, has received considerable attention. In this paper, we propose a new approach to solve ordinal regression problems within the learning vector quantization framework. It extends the previous approach termed ordinal generalized matrix learning vector quantization with a more suitable and natural cost function, leading to more intuitive parameter update rules. Moreover, in our approach the bandwidth of the prototype weights is automatically adapted. Empirical investigation on a number of datasets reveals that overall the proposed approach tends to have superior out-of-sample performance, when compared to alternative ordinal regression methods.

Keywords: Ordinal regression, Learning vector quantization, Generalized matrix learning vector quantization

1. Introduction

Recently a new learning setting that predicts categories of ordinal scale, referred to as ordinal regression or ranking learning, has received considerable attention [1, 2, 3, 4, 5]. In this setting, the training examples are labeled by categories (ranks) exhibiting a natural order. Consequently, ordinal regression bears resemblance to both regression and classification. However, in contrast to regression, the ranks are of discrete and finite

*Corresponding author at: Shenyang Institute of Automation, Chinese Academy of Sciences, No.114, Nanta Street, Shenyang, Liaoning Province, 110016, China.
Email addresses: tangfengzhen87@hotmail.com (Fengzhen Tang), P.Tino@cs.bham.ac.uk (Peter Tiňo)

type, and the difference between the consecutive ranks is not quantified. The existence of order information in the class labels also makes ordinal regression different from classification. Ordinal regression problems arise commonly in real world applications, for example, financial movements grading [6], credit rating [7], life quality assessments [8], facial beauty assessment [9], etc.

A variety of algorithms have been developed for ordinal regression problems in the literatures. The simplest idea, presented in [10], is to map the ordinal scales into numeric values and then solve the problem as a regular regression problem (e.g. by a regression tree). However, this may be suboptimal since the true distances between ordinal scales are unknown in most cases. Another idea is to transform the ordinal regression problem into a set of binary classification problems. For instance, [11] transforms an C-class ordinal problem into $C-1$ binary classification problems and then combines the $C-1$ model predictions to estimate the probabilities of the C original ordinal classes for test instances. This method requires no modification of the standard binary classification algorithm. However, it cannot capture the overall structure among the ordinal classes. It is also possible to formulate the ordinal regression problem as a large augmented binary classification problem by adding additional constraints. Herbrich et al. [12] applied the principle of structural risk minimization to ordinal regression, leading to a new distribution-independent learning algorithm based on a loss function between pairs of ranks. Crammer and Spinger [13] generalized the online perception algorithm with multiple thresholds to seek the direction and thresholds for ordinal regression. Shashua and Levin [14] generalized the support vector machine formulation for ordinal regression by maximizing the margin of $C-1$ parallel hyperplanes separating adjacent classes in the feature space. The parallel separation hyperplanes share the same normal vector and are defined by a set of $C-1$ thresholds $b_1, ..., b_{C-1}$. The margin can either be the same for all separating hyperplanes, or can be different for each adjacent class separation, in which case the sum of the margins is maximized. However, in this approach it cannot be guaranteed that the optimized thresholds will preserve the category order. To address this problem, Chu and Keerthi [15, 16] proposed explicit and implicit constraints enforcing the inequalities on the thresholds, i.e $b_1 < b_2 < ... < b_{C-1}$. The explicit constraints directly enforce order on the ad-

jacent thresholds, while the implicit constraints ensure the threshold order implicitly through the data by stipulating that the j-th hyperplane (corresponding to threshold b_j) separates all points from classes $\leq j$ from all points of classes $> j$. Li and Lin[17] introduced a unified reduction framework from ordinal regression to binary classification. The explicit and implicit approaches of [16] can be regarded as special instances of this framework. Chu and Ghahramani [18] proposed a simple non-parametric Bayesian approach to ordinal regression based on a generalization of the probit likelihood function for Gaussian process, achieving competitive (often better) generalization performance to the support vector approaches. Sun et al. [19] augmented kernel discriminate learning for ordinal regression to take class distribution into consideration.

Recently, Fouad and Tiňo [3] extended generalized matrix learning vector quantization approach [20] to ordinal regression, where the order information among different categories is utilized in the selection of the prototypes to be adapted, as well as in updating of the selected prototypes. Given an *(input, target class)* training example, generalized matrix learning vector quantization identifies two prototypes - the closest prototype of the target class and the closest prototype among the prototypes with a different label. The positions of this prototype pair are updated according to a well-defined cost function. In the extended generalized matrix learning vector quantization approach to ordinal regression, spatially close prototypes to the training input from neighbouring classes (in terms of the class order) of the target class are selected as "correct" prototypes, while prototypes from classes far away in the class order from the target class, but lying in the neighbourhood of the training input are selected as "incorrect" prototypes. Correct and incorrect prototypes are then paired into a set of *(correct prototype, incorrect prototype)* couples. Updating rules for each such prototype couple are formulated in a principled manner through a weighted cost function. The weights for correct and incorrect prototypes will decrease and increase, respectively, with growing class difference from the target class. In addition, for incorrect prototypes, the weights will diminish with increasing distance from the input data.

However, the cost function of [3] based on the pairing strategy does not wholistically take into account the global order on classes. While prototype pairing, given an input vector, of correct and incorrect prototypes naturally extends the generalized

matrix learning vector quantization approach of [20], the price to be paid is the need to brake the global linear class order into ordered pairs that are treated independently of each other. This is unnatural and the updating rules of the prototypes derived from this cost function cannot consistently guarantee the ordering relation among the prototypes.

In this paper, we propose a new cost function that is not only more natural and intuitive, but crucially yields prototype updates that explicitly express the global prototype ordering. This is in contrast to the approach of [3] that employs partial pair-wise comparisons reflecting the global prototype order only implicitly. We aslo show that the deravatives of our cost function vanish at boundaries of the prototype receptive fields and thus gradient based optimization will constitute a valid gradient descent. Furthermore, we propose to automatically adapt parameters in the weighting functions employed in the cost functional, eliminating the need for costly grid search of [3].

Using both artificial and real ordinal regression datasets, we demonstrate that the generalization performance of the proposed approach is competitive and often better than the approach presented in [3]. We will refer to the original approach of [3] and the proposed approach as pair ordinal generalized matrix learning vector quantization (p-OGMLVQ) and accumulative ordinal generalized matrix learning vector quantization (a-OGMLVQ), respectively.

The rest of the paper is organized as follows. In section 2, we briefly introduce learning vector quantization and its extension to ordinal regression. In section 3, we present the proposed approach along with automatic hyperparameter updates in the weighting function. Experimental results are given in section 4. Main findings and conclusions are presented in section 5.

2. Background

Our approach is developed within the framework of learning vector quantization (LVQ) [21, 22]. LVQ is a prototype-based supervised classification algorithm where the classifier is parametrized by a set of labeled prototypes which live in the same space as the input data. The approach has enjoyed popularity because of its simplicity, intuitive nature, and natural accommodation of multi-class classification problems. In

this section, we will briefly introduce LVQ and its extension to ordinal regression.

Consider a training dataset $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^m \times \{1, ..., C\}$, $i = 1, .., n$, where m is the dimension of the inputs, C is the number of different classes and n is the number of training examples. A typical LVQ classifier consists M ($M \geq C$) prototypes $\boldsymbol{w}_i \in \mathbb{R}^m$, which are labeled by $c(\boldsymbol{w}_i) \in \{1, ..., C\}$. The classification is implemented as a winner-takes-all scheme. For a data point $\boldsymbol{x} \in \mathbb{R}^m$, the output class is determined by the class label of its closest prototype: i.e. $\hat{y}(\boldsymbol{x}) := c(\boldsymbol{w}_i)$ such that $i = \arg\min_j d(\boldsymbol{x}, \boldsymbol{w}_j)$, where $d(\cdot, \cdot)$ is a distance measure in $\mathbb{R}^m$. Each labeled prototype $\boldsymbol{w}_i$ with label $c(\boldsymbol{w}_i)$ defines a receptive filed in the input space – a set of points which pick this prototype as their winner. The goal of learning the LVQ classifier is to adapt prototypes automatically such that the class labels of data points within the receptive field coincide with the label of the respective prototype as much as possible.

A generalization of LVQ, termed generalized learning vector quantization (GLVQ), was introduced in [23]. In GLVQ the prototypes are updated based on the steepest descent method on a well defined cost function. The cost function is determined so that the obtained learning rule satisfies the convergence condition. In the training phase of GLVQ, for each labeled input $\boldsymbol{x}_i$, a pair of prototypes will be updated. The closest prototype to $\boldsymbol{x}_i$ with the same label y_i (correct prototype) is rewarded by dragging it closer to $\boldsymbol{x}_i$, while the closest prototype with a different label (incorrect prototype) is penalized by pushing it away from $\boldsymbol{x}_i$.

GLVQ has been extended towards metric learning, where a general adaptive full metric tensor of the distance measure is learned. In this way important dependencies between different input featuresas well as their individual importance for the classification task can be revealed and utilized [20]. In this approach, termed generalized matrix learning vector quantization (GMLVQ), again at each training step a *single* pair of prototypes is updated (the closest correct and the closest incorrect prototype to the current training input). Unlike in GLVQ, the distance function is also updated so that the distance between the example $\boldsymbol{x}_i$ and its closest correct prototype is shortened, while the distance between the example $\boldsymbol{x}_i$ and its closest incorrect prototype is enlarged.

The GMLVQ approach has been further extended to ordinal regression (ordinal GMLVQ - OGMLVQ, which is referred to as pair-OGMLVQ (p-OGMLVQ) in this

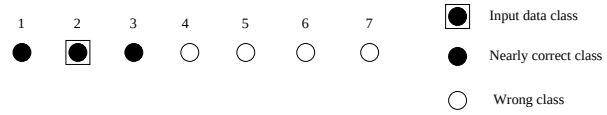

Figure 1: Illustration of ordinal learning vector quantization [3], with threshold $\tau = 1$. The class y_i of the training input $\boldsymbol{x}_i$ is 2, indicated with a square. Prototype classes indicated by black circles (i.e. 1, 2, and 3) will be regarded as correct ones, while other prototype classes denoted by white circles (i.e. 4 – 7) will be seen as incorrect ones.

paper in order to ease the presentation) in [3]. In the p-OGMLVQ approach, at each training step *several* pairs of "correct" and "incorrect" prototypes are updated. First, correct and incorrect prototype classes are identified according to the label order. If the prototype class is "close" to the class y_i of the training input $\boldsymbol{x}_i$ (label distance smaller or equal to a threshold τ), the prototype class will be viewed as "tolerably correct", otherwise it will be considered as incorrect (see Figure 1). The correct prototypes are the spatially closest prototypes to $\boldsymbol{x}_i$ (one for each class) from the correct classes. The incorrect prototypes are the prototypes with incorrect class lying in the neighbourhood of $\boldsymbol{x}_i$.

As in GMLVQ, the correct prototypes will be dragged towards $\boldsymbol{x}_i$, while the incorrect prototypes will be pushed away from $\boldsymbol{x}_i$. However, they will be not all dragged or pushed away to the same extent - a weighting scheme reflecting how close the class of the prototype is to the class of $\boldsymbol{x}_i$ is introduced. The weights for the correct prototypes are given as follows:

$$\alpha_j^+ = \exp\left\{-\frac{\kappa(y_i, c(\boldsymbol{w}_j^+))^2}{2\sigma^2}\right\}, \tag{1}$$

where $\boldsymbol{w}_j^+$ denotes the j-th correct prototype and $\kappa(\cdot,\cdot)$ is the loss function on the class labels. Commonly absolute error loss is adopted, i.e. $\kappa\left(y_i, c(\boldsymbol{w}_j)\right) = |y_i - c(\boldsymbol{w}_j)|$. σ is the bandwidth parameter of the Gaussian weighting function. The weight for the j-the incorrect prototype $\boldsymbol{w}_j^-$ is defined as follows:

$$\alpha_j^- \quad = \quad \exp\left\{-\frac{\left(T - \kappa\left(y_i, c(\boldsymbol{w}_j^-)\right)\right)^2}{2\sigma^2}\right\} \cdot \exp\left\{-\frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{2\sigma'^2}\right\}, \tag{2}$$

where $T = \max_{\boldsymbol{u}\in W^-} \kappa(y_i, c(\boldsymbol{u}))$ and W^- is the set of incorrect prototypes of data point $\boldsymbol{x}_i$. In other words, T is the maximum class difference between the data point and

the identified incorrect prototypes. $d^{\Lambda}(\cdot,\cdot)$ is the squared distance between data point and prototype [1], defined as $d^{\Lambda}(\boldsymbol{x},\boldsymbol{w}) = (\boldsymbol{x}-\boldsymbol{w})^T \cdot \Lambda \cdot (\boldsymbol{x}-\boldsymbol{w})$, where Λ is an $m \times m$ symmetric positive semi-definite matrix (metric tensor). The positive semi-definiteness of Λ can be enforced by substituting $\Lambda = \Omega^T\Omega$, where Ω is an unconstrained $m \times m$ matrix. Λ needs to be normalized since the (global) scale is irrelevant in GMLVQ. σ' is the bandwidth parameter of the Gaussian weighting function corresponding to distance between data point and prototype. These hyper-parameters can be tuned via cross-validation.

Motivated by GMLVQ, the cost function of p-OGMLVQ is defined through pairing correct and incorrect prototypes into the corresponding couples. Suppose there are r couples, for each couple j, a cost function is defined as follows:

$$f_j(\boldsymbol{x}_i) = \frac{\alpha_j^+ \cdot d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^+) - \alpha_j^- \cdot d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\alpha_j^+ \cdot d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \alpha_j^- \cdot d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}. \tag{3}$$

Consequently, the overall cost function is given by

$$C_{p-OGMLVQ} = \sum_{i=1}^{n}\sum_{j=1}^{r} \Phi(f_j(\boldsymbol{x}_i)), \tag{4}$$

where $\Phi(\cdot)$ is a monotonically increasing function. The core of the formulation in eq. (3) has been well studied both theoretically and empirically in the past as a large margin inducing cost function for prototype based classifiers (e.g. [23]). Moreover, dimensionality independent generalization bound can be derived by the LVQ cost function. Monotonically increasing function Φ plays a modulating role for the margin quantification and can be simply the identity $\Phi(x) = x$, or the logistic function $\Phi(x) = 1/(1+e^{-x})$ (if the large values need to be squashed - e.g. to dampen the influence of class-conditional outliers).

The updating rules of the prototypes and the metric can be derived by minimizing

[1] In this weight definition, the distance $d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)$ is involved in order to take the following aspect into consideration: if incorrect prototype is already far away from the input point, then we do not need a large weight to push it away that hard.

the cost function:

$$\begin{aligned}
\boldsymbol{w}_j^+ &:= \boldsymbol{w}_j^+ + \epsilon \cdot 2 \cdot \mu_j^+ \cdot \Lambda \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^+), && (5)\\
\boldsymbol{w}_j^- &:= \boldsymbol{w}_j^+ - \epsilon \cdot 2 \cdot \mu_j^- \cdot \Lambda \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^-), && (6)\\
\Omega &:= \Omega - \eta \cdot \left\{ 2 \cdot \mu_j^+ \cdot \Omega \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^+)(\boldsymbol{x}_i - \boldsymbol{w}_j^+)^T \right. \\
&\quad \left. -2 \cdot \mu_j^- \cdot \Omega \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^-)(\boldsymbol{x}_i - \boldsymbol{w}_j^-)^T \right\}, && (7)
\end{aligned}$$

where

$$\begin{aligned}
\mu_j^+ &= \frac{2\alpha_j^+ \cdot \alpha_j^- \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\left(\alpha_j^+ \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \alpha_j^- \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)\right)^2}, && (8)\\
\mu_j^- &= \frac{2\alpha_j^+ \cdot \alpha_j^- \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}{\left(\alpha_j^+ \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \alpha_j^- \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)\right)^2}, && (9)
\end{aligned}$$

$0 < \epsilon < 1$ and $0 < \eta < 1$ are learning rates for prototypes and metric, respectively. The learning rates are monotonically decreasing with time [20, 3] :

$$\begin{aligned}
\epsilon(t) &= \frac{\epsilon(0)}{1 + \nu(t-1)}, && (10)\\
\eta(t) &= \frac{\eta(0)}{1 + \nu(t-1)}, && (11)
\end{aligned}$$

where $\nu > 0$ is a parameter determining the decay speed, t denotes the number of training epochs. Commonly the magnitude of η is setted smaller than that of ϵ, in order to achieve a slower time-scale of metric learning compared to the weight updates [20].

3. Proposed Algorithm

In the p-OGMLVQ approach, μ_j^+ and μ_j^- control the magnitude of "forces" when updating the identified prototypes. Naturally μ_j^+ should be monotonically decreasing as a function of label distance between the prototype $\boldsymbol{w}_j^+$ and training input $\boldsymbol{x}_i$. In other words, if $\boldsymbol{w}_j^+$ is "less correct" than $\boldsymbol{w}_k^+$, i.e. $\kappa\left(y_i, c(\boldsymbol{w}_j^+)\right) > \kappa\left(y_i, c(\boldsymbol{w}_k^+)\right)$, then $\boldsymbol{w}_j^+$ should be dragged towards $\boldsymbol{x}_i$ to a lesser extent than $\boldsymbol{w}_k^+$, i.e. $\mu_j^+ < \mu_k^+$. However, this property might be violated as illustrated in Figure 2. The input pattern ($\boldsymbol{x}$,2) represents a training input $\boldsymbol{x}$ with class label 2. There are several labeled prototypes $(\boldsymbol{w}_j^\pm, c(\boldsymbol{w}_j^\pm))$ in the neighbourhood of $\boldsymbol{x}$. Assume $\sigma = \sigma' = 0.5$, Based on the prototype pairing and

prototype distances $d(\boldsymbol{x}, \boldsymbol{w}_j^{\pm})$ from $\boldsymbol{x}$ shown in the figure, we get $\mu_1^+ = 0.0514$ and $\mu_2^+ = 0.5381$. Obviously, $\boldsymbol{w}_2^+$ with label 1 is "less correct" than $\boldsymbol{w}_1^+$ with label 2 matching the label of $\boldsymbol{x}$. However, counterintuitively, we have $\mu_2^+ > \mu_1^+$ and the less correct prototype $\boldsymbol{w}_2^+$ will be updated more than the correct one $\boldsymbol{w}_1^+$. Similar problems may occur for μ^-.

To solve this problem mentioned above, we propose a new cost function such that the update rules derived from it will explicitly guarantee that the attractive force decreases as the class difference between the correct prototype and the input increases. It will also be the case that the repulsive force increases as the class difference between the incorrect prototype and the input increases. Consequently, the ordering of prototype classes is ensured. Since incorrect prototypes far from the input are of lesser importance, the repulsive force will decrease with increasing prototype distance from $\boldsymbol{x}_i$.

Equally importantly, our approach is derived in a principled manner from a single unified cost function, in contrast to somewhat ad-hoc unnatural decoupling of the p-OGMLVQ parameter updates into separate updates of binary classifiers.

As in section 2, we assume a training dataset $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^m \times \{1, ..., C\}, i = 1, ..., n$, is given, where m is the dimension of the inputs, C is the number of classes and n is the number of training examples. As we are dealing with ordinal regression problems, we add an additional assumption that classes are ordered as $1 \prec 2, ..., \prec C$, where $\prec$ denotes the order relation on labels. As in LVQ network, the proposed classifier is defined by M labeled prototypes $(\boldsymbol{w}_i, c(\boldsymbol{w}_i))$, where $\boldsymbol{w}_i \in \mathbb{R}^m$ and $c(\boldsymbol{w}_i) \in \{1, ..., C\}$. The classification scheme is implemented by means of the winner-takes-all rule, where the output class for a given input $\boldsymbol{x} \in \mathbb{R}^m$ is determined by its closest prototype. Formally, the output class is determined by $\hat{y}(\boldsymbol{x}) := c(\boldsymbol{w}_j)$ such that $j = \arg\min_i d^{\Lambda}(\boldsymbol{x}, \boldsymbol{w}_i)$, where $d^{\Lambda}(\boldsymbol{x}, \boldsymbol{w}_i)$ is the squared distance between input $\boldsymbol{x}$ and the prototype $\boldsymbol{w}_i$, as defined in section 2.

As in [3], our algorithm aims to minimize the average absolute error of class mislabeling. This is reflected by the utilization of absolute label distance $\kappa(y_i, c(\boldsymbol{w})) = |y_i - c(\boldsymbol{w})|$. Following [3], the ordering information in class label is utilized to identify the sets of correct and incorrect prototypes. The classes of prototypes within a neigh-

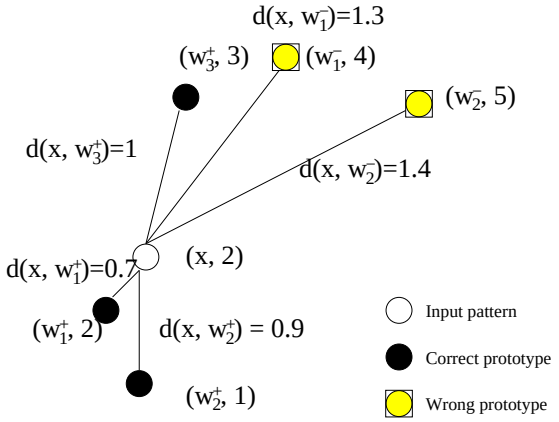

Figure 2: Illustration of the problem of breaking the global class linear order into a set of prototype pairs with correct and incorrect labels, as done in [3]. The input pattern ($\boldsymbol{x}$,2) represents a training input $\boldsymbol{x}$ with class label $y(\boldsymbol{x}) = 2$. There are several labeled prototypes $(\boldsymbol{w}_j^{\pm}, c(\boldsymbol{w}_j^{\pm}))$ in the neighbourhood of $\boldsymbol{x}$. Assume $\sigma = \sigma' = 0.5$, Based on the prototype pairing and prototype distances $d(\boldsymbol{x}, \boldsymbol{w}_j^{\pm})$ from $\boldsymbol{x}$ shown in the figure, we get $\mu_1^+ = 0.0514$ and $\mu_2^+ = 0.5381$. Obviously, $\boldsymbol{w}_2^+$ with label 1 is "less correct" than $\boldsymbol{w}_1^+$ with label 2 matching the label of $\boldsymbol{x}$. However, counterintuitively, we have $\mu_2^+ > \mu_1^+$ and the less correct prototype $\boldsymbol{w}_2^+$ will be updated more that the correct one $\boldsymbol{w}_1^+$.

bourhood (in the label space) of the given input class will be regarded as "correct". Formally, the set of correct classes is defined as follows:

$$C^+ = \{c(\boldsymbol{w}_j) \,|\kappa(y_i, c(\boldsymbol{w}_j)) \leq \tau\}, \tag{12}$$

where $\tau > 0$ is a rank loss threshold. The classes of prototypes which are outside the neighbourhood of the given input class will be treated as "incorrect" ones. The set of incorrect classes is given by:

$$C^- = \{c(\boldsymbol{w}_j) \,|\kappa(y_i, c(\boldsymbol{w}_j)) > \tau\}. \tag{13}$$

The spatially closest prototype in each correct class to the input $\boldsymbol{x}_i$ will be treated as a correct prototype. The set of correct prototypes is then

$$W^+ = \{\boldsymbol{w}_j \,|j = \arg \min_{c(\boldsymbol{w}_l)=k} d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_l), k \in C^+\}. \tag{14}$$

Prototypes from the identified incorrect classes which lie in the neighbourhood of $\boldsymbol{x}_i$ will be treated as incorrect ones that need to be pushed away from $\boldsymbol{x}_i$. The set of

incorrect prototypes is given as follows:

$$W^- = \{\boldsymbol{w}_j \,|d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j) < D, c(\boldsymbol{w}_j) \in C^-\}, \tag{15}$$

where $D > 0$. We chose D to be the median distance of the example $\boldsymbol{x}$ to all prototypes from the incorrect classes [2].

We rearrange the set W^+ and W^- in increasing order in terms of distance to the input $\boldsymbol{x}_i$. Let $r = \min(|W^+|, |W^-|)$, where $|W|$ is the cardinality of the set W. The first r prototypes of each prototype set will be selected for updating. However, instead of updating the prototypes in pairs as in [3], we propose to update all the selected prototypes collectively with a weighting scheme defined by the ordering information of the labels.

Weights for correct prototypes with respect to $c(\boldsymbol{x}_i)$ are given by a Gaussian kernel:

$$\alpha_j^+ = \exp\left\{-\frac{\kappa\left(y_i, c(\boldsymbol{w}_j^+)\right)^2}{2\sigma_1^2}\right\}. \tag{16}$$

Label of a correct prototype $\boldsymbol{w}_j^+$ further away from the label of the input pattern will result in a smaller weight α_j^+. Weights for incorrect prototypes with respect to y_i) are determined as:

$$\alpha_j^- \quad = \quad \exp\left\{-\frac{T - \kappa\left(y_i, c(\boldsymbol{w}_j^-)\right)^2}{2\sigma_2^2}\right\} \cdot \exp\left\{-\frac{(d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}))^2}{2\sigma_3^2}\right\}, \tag{17}$$

where T is the maximum label difference between input pattern and wrong prototypes, i.e. $T = \max_{\boldsymbol{w}_j \in W^-} \kappa\left(y_i, c(\boldsymbol{w}_j)\right)$. Wrong prototypes spatially closer to the input pattern and with labels further away from that of the input pattern will have larger weight α_j^-. We share the formulation of the weight functions with [3]. However, in contrast to [3], the scale parameters σ_1, σ_2 and σ_3 are learned as inherent part of the parameter updates, as explained in section 3.2[3]. Moreover, instead of unnatural pair-wise cost functions of p-OGMLVQ, we formulate a single unified cost function:

$$C_{a-OGMLVQ} = \sum_{i=1}^{n} \Phi(f(\boldsymbol{x}_i)), \tag{18}$$

[2] D can be set as mean distance as well.

[3] In p-OGMLVQ, $\sigma_1 = \sigma_2$.

where

$$f(\boldsymbol{x}_i) = \frac{\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) - \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}. \tag{19}$$

Note that instead of having f_j for each j-th pair of correct and incorrect prototypes, we only have one f for all the identified prototypes. The p-OGMLVQ cost function can be seen as sum of r weighted versions of the GMLVQ cost function [20]. The correct/incorrect prototypes are not treated equally in the p-OGMLVQ cost function. In contrast, our cost function treats the correct and incorrect prototypes equally by accumulating costs of all correct and all incorrect prototypes in a single cost $f(x_i)$. Our approach do not take advantages of binary classifier as [24] does, but directly incorporate the class ordering information in a natural multi-class classification algorithm– GMLVQ.

Following [3], we choose Φ as identity function. The learning rule can be derived from the cost function by differentiating with respect prototypes:

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{w}_j^+} = \frac{\partial f(\boldsymbol{x}_i)}{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)} \cdot \frac{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}{\partial \boldsymbol{w}_j^+}, \tag{20}$$

where

$$\begin{aligned} &\frac{\partial f(\boldsymbol{x}_i)}{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)} \\ &= \frac{\alpha_j^+ \left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)}{\left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\ &- \frac{\alpha_j^+ \left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) - \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)}{\left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\ &= \frac{2\alpha_j^+ \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2}, \end{aligned} \tag{21}$$

and

$$\frac{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}{\partial \boldsymbol{w}_j^+} = -2\Omega^T\Omega(\boldsymbol{x}_i - \boldsymbol{w}_j^+) = -2\Lambda(\boldsymbol{x}_i - \boldsymbol{w}_j^+). \tag{22}$$

Partial derivative of the cost function with respect to $\boldsymbol{w}_j^-$ can be writen as:

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{w}_j^-} = \frac{\partial f(\boldsymbol{x}_i)}{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)} \cdot \frac{\partial d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\partial \boldsymbol{w}_j^-}, \tag{23}$$

where

$$
\begin{aligned}
&\frac{\partial f(\boldsymbol{x}_i)}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)} \\
&= -\frac{\frac{\partial \alpha_j^-}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)} d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \alpha_j^-}{\left(\sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\cdot \left(\sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&- \frac{\frac{\partial \alpha_j^-}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)} d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \alpha_j^-}{\left(\sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\cdot \left(\sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+) - \sum_{k=1}^r \alpha_k^- d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&= -\frac{2\left(1 - d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)/(2\sigma_3^2)\right) \alpha_j^- \sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\left(\sum_{k=1}^r \alpha_k^+ d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2},
\end{aligned}
\tag{24}
$$

and

$$
\frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\partial \boldsymbol{w}_j^-} = -2\Omega^T \Omega(\boldsymbol{x}_i - \boldsymbol{w}_j^-) = -2\Lambda(\boldsymbol{x}_i - \boldsymbol{w}_j^-). \tag{25}
$$

Denoting $\mu_j^+ = \frac{\partial f(\boldsymbol{x}_i)}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}$ and $\mu_j^- = -\frac{\partial f(\boldsymbol{x}_i)}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}$, derivative of the cost function with respect to Ω can be computed as follows:

$$
\begin{aligned}
\frac{\partial f(\boldsymbol{x}_i)}{\partial \Omega} &= \sum_{k=1}^r \frac{\partial f(\boldsymbol{x}_i)}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+)} \cdot \frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\partial \Omega} \\
&\quad + \sum_{k=1}^r \frac{\partial f(\boldsymbol{x}_i)}{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)} \cdot \frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\partial \Omega} \\
&= \sum_{k=1}^r \mu_k^+ \cdot \frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\partial \Omega} - \sum_{k=1}^r \mu_k^- \cdot \frac{\partial d^{\Lambda}(x_i, w_k^-)}{\partial \Omega},
\end{aligned}
\tag{26}
$$

where

$$
\frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\partial \Omega} = 2\Omega(\boldsymbol{x}_i - \boldsymbol{w}_k^+)(\boldsymbol{x}_i - \boldsymbol{w}_k^+)^T, \tag{27}
$$

and

$$
\frac{\partial d^{\Lambda}(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\partial \Omega} = 2\Omega(\boldsymbol{x}_i - \boldsymbol{w}_k^-)(\boldsymbol{x}_i - \boldsymbol{w}_k^-)^T. \tag{28}
$$

Consequently, updating rules can be expressed as:

$$\boldsymbol{w}_j^+ := \boldsymbol{w}_j^+ + \epsilon \cdot 2 \cdot \mu_j^+ \cdot \Lambda \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^+), j = 1, ..., r \tag{29}$$

$$\boldsymbol{w}_j^- := \boldsymbol{w}_j^+ - \epsilon \cdot 2 \cdot \mu_j^- \cdot \Lambda \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j^-), j = 1, ..., r \tag{30}$$

$$\begin{aligned} \Omega := \Omega - \eta \cdot \Bigg(& 2 \cdot \sum_{k=1}^{r} \mu_k^+ \cdot \Omega \cdot (\boldsymbol{x}_i - \boldsymbol{w}_k^+)(\boldsymbol{x}_i - \boldsymbol{w}_k^+)^T \\ & -2 \cdot \sum_{k=1}^{r} \mu_k^- \cdot \Omega \cdot (\boldsymbol{x}_i - \boldsymbol{w}_k^-)(\boldsymbol{x}_i - \boldsymbol{w}_k^-)^T \Bigg). \end{aligned} \tag{31}$$

Comparing the prototype updating rules in the original p-OGMLVQ approach ((5)–(6)), with those of our approach ((29)–(30)), we observe that the prototype updates differ mainly in the definition of μ_j^+ and μ_j^-. In p-OGMLVQ, the updating coefficient $\mu_j^+, j = 1, ..., r$, corresponding to different correct prototypes are not directly comparable (the same holds for μ_j^-). However, in our approach μ_j^+ can be rewritten as

$$\mu_j^+ = \alpha_j^+ \cdot \gamma^+, \tag{32}$$

where

$$\gamma^+ = \frac{2 \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-) \right)^2}. \tag{33}$$

The differences among $\mu_j^+, j = 1, ..., r$ are solely due to the weights α_j^+. Recall that α_j^+ will be greater if the class of the corresponding prototype is closer to the class of the input data $\boldsymbol{x}_i$ in the label space. Consequently, greater μ_j^+ will be assigned to the prototype whose class is closer to the input data.

Similarly, the weight μ_j^- can be rewritten as:

$$\mu_j^- = \left(1 - d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)/(2\sigma_3^2)\right) \alpha_j^- \cdot \gamma^-, \tag{34}$$

where

$$\gamma^- = \frac{2 \sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\left(\sum_{k=1}^{r} \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^{r} \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-) \right)^2}.$$

We can see that μ_j^- increases with α_j^- but decreases with the distance $d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)$, which means that if the incorrect prototype is closer to the input, with large class difference, the updating push-away force μ_j^- will be greater.

3.1. Gradient structure

Following [25], we will now show that the derivatives of our cost function vanish at boundaries of the prototype receptive fields and hence gradient based optimization will constitute a valid gradient descent.

For convenience, we denote the squared distance between input $\boldsymbol{x}$ and prototype $\boldsymbol{w}_i$, $(\boldsymbol{x}-\boldsymbol{w}_i)^T \cdot \Lambda \cdot (\boldsymbol{x}-\boldsymbol{w}_i)$, by $d_i(\boldsymbol{x})$. Let $J(\boldsymbol{x}) = \{J_1, J_2, ..., J_{r(\boldsymbol{x})}\}$ and $K(\boldsymbol{x}) = \{K_1, K_2, ..., K_{r(\boldsymbol{x})}\}$ be the sets of $r(\boldsymbol{x})$ indices of correct and incorrect prototypes, respectively, that are selected for updating, given the input $\boldsymbol{x}$. Note that $r(\boldsymbol{x})$ can vary, depending on the input location $\boldsymbol{x}$. Our scaled general cost function (18)–(19) has the following form (taking Φ as identity function):

$$\hat{S} = \frac{1}{n}\sum_{i=1}^{n} \frac{\sum_{k=1}^{r(\boldsymbol{x_i})} \alpha^+_{J_k}(\boldsymbol{x}_i)\cdot d_{J_k}(\boldsymbol{x}_i) - \sum_{k=1}^{r(\boldsymbol{x_i})} \alpha^-_{K_k}(\boldsymbol{x}_i)\cdot d_{K_k}(\boldsymbol{x}_i)}{\sum_{k=1}^{r(\boldsymbol{x_i})} \alpha^+_{J_k}(\boldsymbol{x}_i)\cdot d_{J_k}(\boldsymbol{x}_i) + \sum_{k=1}^{r(\boldsymbol{x_i})} \alpha^-_{K_k}(\boldsymbol{x}_i)\cdot d_{K_k}(\boldsymbol{x}_i)}. \tag{35}$$

As in [25], suppose the input data comes from a distribution p on the input space $\mathbb{R}^m$ and there is a labeling function $y : \mathbb{R}^m \to \{1, ..., C\}$ assigning each input $\boldsymbol{x}$ its class label $y(\boldsymbol{x})$. The continuous version of the error function $\hat{S}$ reads

$$S = \int_{\mathbb{R}^m} \frac{\sum_{k=1}^{r(\boldsymbol{x})} \alpha^+_{J_k}(\boldsymbol{x})\cdot d_{J_k}(\boldsymbol{x}) - \sum_{k=1}^{r(\boldsymbol{x})} \alpha^-_{K_k}(\boldsymbol{x})\cdot d_{K_k}(\boldsymbol{x})}{\sum_{k=1}^{r(\boldsymbol{x})} \alpha^+_{J_k}(\boldsymbol{x})\cdot d_{J_k}(\boldsymbol{x}) + \sum_{k=1}^{r(\boldsymbol{x})} \alpha^-_{K_k}(\boldsymbol{x})\cdot d_{K_k}(\boldsymbol{x})}\, dp(\boldsymbol{x}). \tag{36}$$

To simplify the presentation, in what follows we do not explicitly denote dependence on $\boldsymbol{x}$, with the understanding that quantities such as r, $\alpha_i^{\pm}$, d_i etc. are all functions of $\boldsymbol{x}$. We partition the input space and the set of prototypes into class sets $X_\ell = \{\boldsymbol{x} | y(\boldsymbol{x}) = \ell\}$ and $W_\ell = \{\boldsymbol{w}_j | c(\boldsymbol{w}_j) = \ell\}$, respectively, $\ell = 1, 2, ..., C$. Assuming X_ℓ are measurable with respect to p, we decompose S as follows:

$$\sum_{\ell=1}^{C} \int_{X_\ell} \frac{\sum_{|l-\ell|\le\tau} \sum_{\boldsymbol{w}_j \in W_l} \alpha_j^+ d_j h(j,l,\ell) - \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \alpha_k^- d_k \tilde{h}(k,\ell)}{\sum_{|l-\ell|\le\tau} \sum_{\boldsymbol{w}_j \in W_l} \alpha_j^+ d_j h(j,l,\ell) + \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \alpha_k^- d_k \tilde{h}(k,\ell)} dp(\boldsymbol{x}) \tag{37}$$

where

- $h(j,l,\ell)$ is an indicator function attaining value 1 if and only if the prototype $\boldsymbol{w}_j \in W_l$ is the closest to $\boldsymbol{x}$ among the prototypes in W_l and the class l is among

the $r(\boldsymbol{x})$ correct classes w.r.t. class ℓ that are selected for prototype update, given input $\boldsymbol{x}$; otherwise $h(j, l, \ell)$ is zero.

- $\tilde{h}(k, \ell)$ is an indicator function indicating that $\boldsymbol{w}_k$ is an incorrect prototype w.r.t. class ℓ and is among the $r(\boldsymbol{x})$ incorrect prototypes closest to $\boldsymbol{x}$.

To find a closed-form expressions for the indicator functions $h(j, l, \ell)$ and $\tilde{h}(k, \ell)$ we first introduce two further indicator functions:

- $\mathcal{I}(j, l)$ indicates that the prototype $\boldsymbol{w}_j$ is the closest prototype to $\boldsymbol{x}$ among those labeled with class l,
- $\tilde{\mathcal{I}}(k, D)$ indicates that the prototype $\boldsymbol{w}_k$ is spatially within the D-neighbourhood of $\boldsymbol{x}$. Recall that D is the median distance of the example $\boldsymbol{x}$ of class $\ell = y(\boldsymbol{x})$ to the incorrect prototypes, i.e. prototypes labeled with l, such that $|l - \ell| > \tau$.

Denoting by H the Heaviside function, we have

$$\mathcal{I}(j, l) = H\left(\sum_{\boldsymbol{w}_q \in W_l} H(d_q - d_j) - |W_l|\right), \tag{38}$$

where $|W_l|$ is the size of W_l (and hence the number of prototypes labeled with class l) and

$$\tilde{\mathcal{I}}(k, D) = H(D - d_k). \tag{39}$$

Denote by $N(\ell)$ and $\tilde{N}(\ell)$ the number of correct and incorrect prototypes, respectively, given the input $\boldsymbol{x}$ and its class $\ell = y(\boldsymbol{x})$. We have,

$$N(\ell) = \sum_{|l-\ell| \leq \tau} 1, \quad \tilde{N}(\ell) = \sum_{|l-\ell| > \tau} \sum_{\boldsymbol{w}_k \in W_l} \tilde{\mathcal{I}}(k, D).$$

Hence,

$$\begin{aligned} r(\boldsymbol{x}) &= \min\left\{N(\ell), \tilde{N}(\ell)\right\} \\ &= N(\ell) - \left(N(\ell) - \tilde{N}(\ell)\right) \cdot H\left(N(\ell) - \tilde{N}(\ell)\right) \\ &= \tilde{N}(\ell) - \left(\tilde{N}(\ell) - N(\ell)\right) \cdot H\left(\tilde{N}(\ell) - N(\ell)\right). \end{aligned}$$

We are now ready to formally express $h(j, l, \ell)$ and $\tilde{h}(k, \ell)$:

$$h(j,l,\ell) = \mathcal{I}(j,l)\cdot H\left(g(j,\ell)+f(\ell)\right), \tag{40}$$

$$\tilde{h}(k,\ell) = \tilde{\mathcal{I}}(k,D)\cdot H(\tilde{g}(j,\ell)+\tilde{f}(\ell)), \tag{41}$$

where

$$g(j,\ell) = \sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_q\in W_l}\mathcal{I}(q,l)\cdot H(d_q-d_j), \tag{42}$$

$$\begin{aligned} f(\ell) &= -(N(\ell)-r(\boldsymbol{x})+1) \\ &= -(N(\ell)-\tilde{N}(\ell))\cdot H(N(\ell)-\tilde{N}(\ell))-1, \end{aligned} \tag{43}$$

$$\tilde{g}(k,\ell) = \sum_{|l-\ell|>\tau}\sum_{\boldsymbol{w}_q\in W_l}\tilde{\mathcal{I}}(q,D)\cdot H(d_q-d_k), \tag{44}$$

$$\begin{aligned} \tilde{f}(\ell) &= -(\tilde{N}(\ell)-r(\boldsymbol{x})+1) \\ &= -\left(\tilde{N}(\ell)-N(\ell)\right)\cdot H(\tilde{N}(\ell)-N(\ell))-1. \end{aligned} \tag{45}$$

The first factor of $h(j,l,\ell)$ verifies whether the prototype $\boldsymbol{w}_j$ is one of the "correct" prototypes, while the second factor further checks whether this prototype is one of the first r closest ones among all the "correct" prototypes. This is realized by ordering the "correct" prototypes through $g(j,\ell)$ and selecting the r first closest ones by $f(\ell)$. The same trick is used in $\tilde{h}(k,\ell)$ to pick the first r closest prototypes among those "incorrect" ones.

Let $c_i = c(\boldsymbol{w}_i)$ be the label of prototype $\boldsymbol{w}_i$. Denoting

$$D_i = \Lambda\cdot(\boldsymbol{x}-\boldsymbol{w}_i),$$

$$E(\ell) = \sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l}\alpha_j^+ d_j h(j,l,\ell),$$

and

$$\tilde{E}(\ell) = \sum_{|l-\ell|>\tau}\sum_{\boldsymbol{w}_k\in W_l}\alpha_k^- d_k \tilde{h}(k,\ell),$$

the derivative of (37) with respect to $\boldsymbol{w}_i$ reads:

$$\sum_{|\ell-c_i|\leq\tau}\int_{X_\ell}\frac{4\alpha_i\tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2}D_i h(i,c_i,\ell)dp(\boldsymbol{x}) \tag{46}$$

$$+\quad\sum_{|\ell-c_i|>\tau}\int_{X_\ell}\frac{-4(1-d_i/(2\sigma_3^2))\alpha_i E(\ell)}{(E(\ell)+\tilde{E}(\ell))^2}D_i\tilde{h}(i,\ell)dp(\boldsymbol{x}) \tag{47}$$

$$+\quad\sum_{\ell=1}^{C}\int_{X_\ell}\sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l}\frac{2\alpha_j d_j\tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2}\frac{\partial h(j,l,\ell)}{\partial\boldsymbol{w}_i}dp(\boldsymbol{x}) \tag{48}$$

$$+\quad\sum_{|\ell-c_i|>\tau}\int_{X_\ell}\sum_{|l-\ell|>\tau}\sum_{\boldsymbol{w}_k\in W_l}\frac{2\alpha_k d_k E(\ell)}{(E(\ell)+\tilde{E}(\ell))^2}\frac{\partial\tilde{h}(k,\ell)}{\partial\boldsymbol{w}_i}dp(\boldsymbol{x}). \tag{49}$$

Note that (46) and (47) correspond to updates (29) and (30), respectively. Terms (48) and (49) will vanish as shown in the appendix.

Similarly, we examine the derivative of (37) with respective to the metric term Ω. Denoting $\Omega(\boldsymbol{x}-\boldsymbol{w}_i)(\boldsymbol{x}-\boldsymbol{w}_i)^T$ by F_i, then the derivative reads:

$$\sum_{\ell=1}^{C}\int_{X_\ell}\frac{4}{(E(\ell)+\tilde{E}(\ell))^2}\left(\tilde{E}(\ell)\sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l}\alpha_j F_j h(j,l,\ell)\right. \tag{50}$$

$$\left.-E(\ell)\sum_{|l-\ell|>\tau}\sum_{\boldsymbol{w}_k\in W_l}(1-d_k/(2\sigma_3^2))\alpha_k F_k\tilde{h}(k,\ell)\right)dp(\boldsymbol{x}) \tag{51}$$

$$+\quad\sum_{\ell=1}^{C}\int_{X_\ell}\frac{2}{(E(\ell)+\tilde{E}(\ell))^2}\left(\tilde{E}(\ell)\sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l}\alpha_j d_j\frac{\partial h(j,l,\ell)}{\partial\Omega}\right. \tag{52}$$

$$\left.-\tilde{E}(\ell)\sum_{|l-\ell|>\tau}\sum_{\boldsymbol{k}\in W_l}\alpha_k d_k\frac{\partial\tilde{h}(k,\ell)}{\partial\Omega}\right)dp(\boldsymbol{x}). \tag{53}$$

Terms (50) and (51) correspond to the update of Ω in (31). As shown in the appendix, (52) and (53) will vanish. Consequently, with an appropriate choice of learning rate, the update of a-OGMLVQ constitutes a stochastic gradient descent method.

3.2. Hyperparameter Learning in the Proposed Algorithm

In this section, we propose to automatically adapt the bandwidth parameters σ_1, σ_2, and σ_3 in the Gaussian weighting functions during training process through minimizing the cost function with respect to the parameters σ_1, σ_2, and σ_3.

Partial derivative of $f(\boldsymbol{x}_i)$ with respect to σ_1 is computed as follows:

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \sigma_1} = \sum_{j=1}^{r} \frac{\partial f(\boldsymbol{x}_i)}{\partial \alpha_j^+} \cdot \frac{\partial \alpha_j^+}{\partial \sigma_1}, \tag{54}$$

where

$$\begin{aligned}
\frac{\partial f(\boldsymbol{x}_i)}{\partial \alpha_j^+} &= \frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}{\left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\quad \cdot \left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&\quad - \frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+)}{\left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\quad \cdot \left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) - \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&= \frac{2 d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)}{\left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&= d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) \cdot \gamma^+,
\end{aligned} \tag{55}$$

and

$$\frac{\partial \alpha_j^+}{\partial \sigma_1} = \alpha_j^+ \frac{h\left(c(\boldsymbol{x}_i), c(\boldsymbol{w}_j^+)\right)^2}{2\sigma_1^3}. \tag{56}$$

Partial derivatives of $f(\boldsymbol{x}_i)$ with respect to σ_2 and σ_3 read:

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \sigma_2} = \sum_{j=1}^{r} \frac{\partial f(\boldsymbol{x}_i)}{\partial \alpha_j^-} \cdot \frac{\partial \alpha_j^-}{\partial \sigma_2}, \tag{57}$$

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \sigma_3} = \sum_{j=1}^{r} \frac{\partial f(\boldsymbol{x}_i)}{\partial \alpha_j^-} \cdot \frac{\partial \alpha_j^-}{\partial \sigma_3}, \tag{58}$$

where

$$\begin{aligned}
&\frac{\partial f(\boldsymbol{x}_i)}{\partial \alpha_j^-} = \\
&-\frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\left(\sum_{j=1}^r \alpha_j^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\cdot \left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&-\frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)}{\left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&\cdot \left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) - \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right) \\
&= -\frac{2 d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-) \sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+)}{\left(\sum_{k=1}^r \alpha_k^+ d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^+) + \sum_{k=1}^r \alpha_k^- d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_k^-)\right)^2} \\
&= -d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-) \cdot \gamma^-, \qquad (59)
\end{aligned}$$

and

$$\frac{\partial \alpha_j^-}{\partial \sigma_2} = \alpha_j^- \frac{\left(T - h(c(\boldsymbol{x}_i) - c(\boldsymbol{w}_j^-)\right)^2}{2\sigma_2^3}, \qquad (60)$$

$$\frac{\partial \alpha}{\partial \sigma_3} = \alpha_j^- \frac{d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j -)}{2\sigma_3^3}. \qquad (61)$$

Hence, updating of σ_1, σ_2 and σ_3 proceeds as follows:

$$\sigma_1 := \sigma_1 - \epsilon \cdot \frac{\gamma^+}{2\sigma_1^3} \cdot \sum_{j=1}^r \alpha_j^+ \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^+) \cdot \left(h(c(\boldsymbol{x}_i), c(\boldsymbol{w}_j^+)\right)^2, \qquad (62)$$

$$\sigma_2 := \sigma_2 + \epsilon \cdot \frac{\gamma^-}{2\sigma_2^3} \cdot \sum_{j=1}^r \alpha_j^- \cdot d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-) \cdot \left(T - h(c(\boldsymbol{x}_i), c(\boldsymbol{w}_j^-)\right)^2, \qquad (63)$$

$$\sigma_3 := \sigma_3 + \epsilon \cdot \frac{\gamma^-}{2\sigma_3^3} \cdot \sum_{j=1}^r \alpha_j^- \cdot \left(d^\Lambda(\boldsymbol{x}_i, \boldsymbol{w}_j^-)\right)^2. \qquad (64)$$

The hyperparameter updaptation (62)–(64) is similar to that in Robust Soft LVQ algorithm presented in [26]. However the hyperparameters in our approach serve a difference purpose, introducing scales to the ordered label and input spaces.

The overall method is summarized in algorithm 1.

Algorithm 1 The proposed ordinal regression algorithm based on LVQ

Initialization : Initialize the prototypes $\boldsymbol{w}_j \in \mathbb{R}^d$. Initialize the metric tensor parameter Ω by setting it as identity matrix. Randomly initialize the bandwidth parameters σ_1, σ_2, and σ_3.
while a stopping criterion is not reached **do**
  Randomly select a training example $(\boldsymbol{x}_i, y_i)$.
  Determine the correct classes C^+ and incorrect classes C^- for $\boldsymbol{x}_i$ based on (12) and (13), respectively.
  Identify the correct prototype set W^+ and incorrect prototype set W^- using (14) and (15), respectively.
  Order W^+ and W^- in increasing order in terms of distance to the input $\boldsymbol{x}_i$, only keep the first r protptypes in the set W^+ and W^-, where $r = \min(|W^+|, |W^-|)$.
  Compute the weights for the correct and incorrect prototypes found in the previous step according to (16) and (17), respectively.
  for $\forall \boldsymbol{w}_j \in W^+$ **do**
    Update $\boldsymbol{w}_j$ according to (29)
  end for
  for $\forall \boldsymbol{w}_j \in W^-$ **do**
    Update $\boldsymbol{w}_j$ according to (30)
  end for
  Update Ω according to (31). Similarly, Ω is normalized so that $\sum_i \Lambda_{ii} = 1$ to prevent the algorithm from degeneration.
  Update σ_1, σ_2, and σ_3 according to (62), (63), and (64), respectively.
end while

4. Experiments

In this section, we study the performance of the proposed ordinal approach. Three performance measures were used:

1. *MZE*: The Mean Zero-one Error (MZE), also known as misclassification rate, is the fraction of incorrect predictions:

$$MZE = \frac{\sum_{i=1}^{n} I(y_i \neq \hat{y}(\boldsymbol{x}_i))}{n}, \tag{65}$$

   where n is the number of the test examples. $I(\cdot)$ is the indicator function whose function value is 1, if the argument $*$ is true, 0 otherwise. MZE values range from 0 to 1 and reflect a global performance for the classification task (without considering class order). However, this performance measurement alone is not suitable for ordinal regression problems.

2. *MAE*: The Mean Absolute Error (MAE) is the average absolute deviation of the predicted ranks from the true ranks:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}(\boldsymbol{x}_i)|}{n}. \tag{66}$$

   The MAE values range from 0 to $C-1$. Since MZE does not reflect the category order, MAE is typically used in the ordinal regression literature together with MZE [27]. However, this performance measurement is not suitable for imbalanced datasets.

3. *MMAE*: Macroaveraged Mean Absolute Error (MMAE) is specially designed for evaluating ordinal regression problems with imbalanced classes:

$$MMAE = \frac{1}{C}\sum_{j=1}^{C} \frac{\sum_{y_i=j} |y_i - \hat{y}(\boldsymbol{x}_i)|}{n_j}, \tag{67}$$

   where C is the number of classes and n_j is the number of examples of class j. The MMAE values also range from 0 to $C-1$.

We first compared our approach with p-OGMLVQ on the ordinal regression benchmark datasets provided by Chu et. al [15]. As a baseline, the performance of GMLVQ

Table 1: Brief description of the ordinal regression benchmark datasets

Dataset	Dimension	#Training	#Test
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	8	6000	16784

was also reported. A brief description of the datasets is given by Table 1. These datasets are not real ordinal datasets but regression ones. These datasets will be turned into ordinal regression through discretization. Following [3], the continuous targets were discretized into 10 ordinal categories (equal-frequency binning). The input vectors were normalized to zero mean and unit variance. Each data set was randomly partitioned into training/test splits according to Table 1. The partition was repeated 20 times independently, yielding 20 re-sampled training/test sets.

Prototype initialization and learning rates were set in the same way for all three approaches. Number of prototypes per class was tuned on the training sets, with candidates ranging in $\{1, 2, 3, 4, 5\}$, via 5-fold cross validation. For p-OGMLVQ, the ranking loss threshold τ together with the bandwidth parameters σ and σ_1 were also tuned through 5-fold cross validation. Candidate values for the hyperparameters are listed as follows: $\tau \in \{0, 1, 2\}$, $\sigma, \sigma_1 \in \{0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 1, 1.2, 2, 3, 5, 50\}$. For our approach, we also tuned the ranking loss threshold τ via 5-fold cross validation over the same candidate values as in p-OGMLVQ (The bandwidth parameters in our approach are automatically adapted during training process).

The MZE and MAE results [4], along with standard deviation across 20 runs, are

[4]MMAE results were not given in this case, since these datasets are balanced.

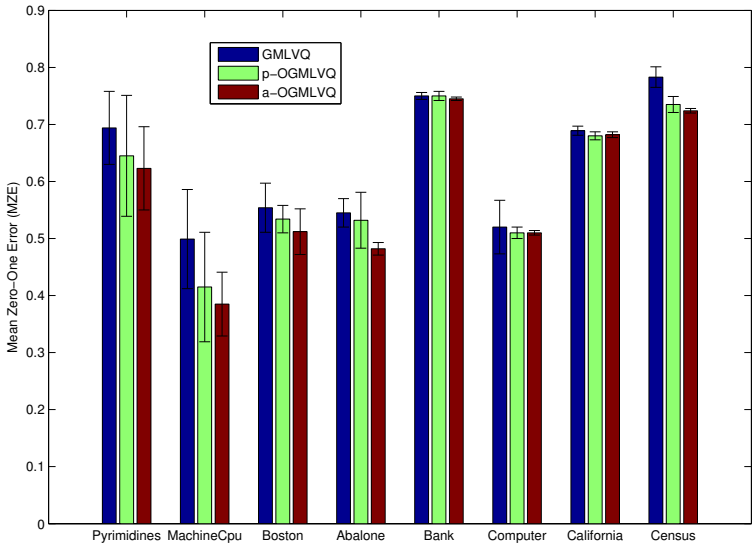

Figure 3: MZE results for the eight ordinal regression benchmark datasets.

given by Figure 3 and 4, respectively. The results show that our a-OGMLVQ ordinal approach was better than p-OGMLVQ in terms of MZE and MAE performance measures in most of the datasets, only occasionally our a-OGMLVQ obtained comparable performance. One possible reason might be that the problem mentioned in Figure 2 do not happen in that case. Still, our a-OGMLVQ ordinal approach is consistently better than the baseline GMLVQ approach.

We further compared our ordinal approach to other ordinal approaches: two support vector ordinal approaches proposed in [15] – support vector ordinal regression with implicit constraints (SVORIM) and support vector ordinal regression with explicit constraints (SVOREX), SVM based reduction framework of ordinal regression to binary classification (RED-SVM) proposed in [17], and an ordinal approach extended from kernel discriminate analysis proposed in [19]. The MZE and MAE results are reported in Tables 2 and 3, respectively. Our a-OGMLVQ approach beats the other approaches four times in terms of MZE and twice in terms of MAE. In order to have a general summary of the results, we computed the mean ranking values when com-

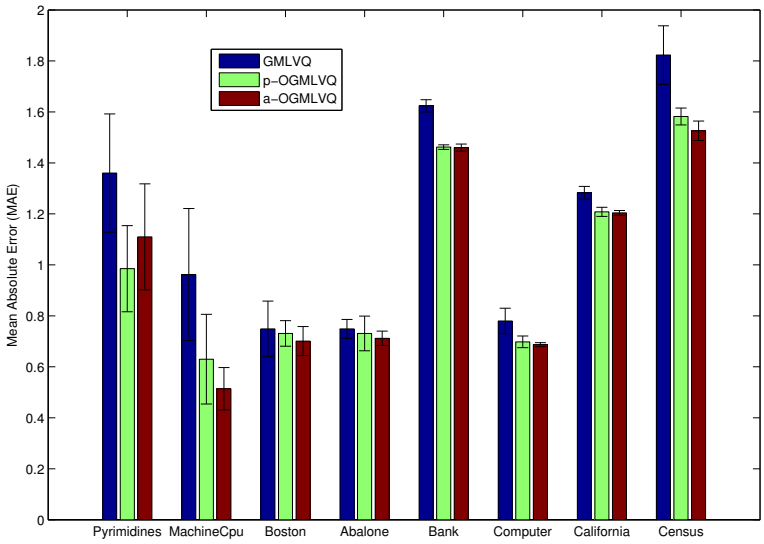

Figure 4: MAE results for the eight ordinal regression benchmark datasets.

paring the different methods, where rank $R = 1$ corresponds to the best method, while $R = 6$ represents for the worse one. The mean rankings $\bar{R}$ were computed based on the performance on the test sets. The results are given in Table 4, indicating that our approach is the best in terms of MZE and third in terms of MAE. For both performance measures the mean rank of a-OGMLVQ is clearly superior to that of p-OGMLVQ.

The datasets described in Table 1 are not real ordinal regression tasks but adapted from regression tasks. Thus, we further evaluated our approach on 10 real ordinal regression datasets with imballanced classes suggested in [27], which are briefly described in Table 5. Following the experimental setup in [27], 30 different random splits of the datasets were considered, with 75% of the instances for training, the remaining 25% for testing. The data partitions were shared by all the compared methods. The results reported in Tables 6, 7, and 8 are the average values over 30 runs, together with standard derivations. From the results, we can see that our approach beats the other algorithms once in terms of MZE, twice in terms of MAE, and three times in terms of MMAE. Again, the mean rankings were computed and reported in Table 9.

Table 2: Mean zero-one error (MZE) results along with standard deviations ($\pm$) across 20 runs of the ordinal regression benchmark datasets given in Table 1. Our a-OGMLVQ approach is compared with KDLOR reported in [19], SVORIM and SVOREX reported in [15], RED-SVM reported in [17], and p-OGMLVQ reported in [3]. The best performance for each dataset has been boldfaced.

Dataset/Method	KDLOR	SVORIM	SVOREX	RED-SVM	p-OGMLVQ	a-OGMLVQ
Pyrimidines	0.739 ± 0.055	0.719 ± 0.066	0.752 ± 0.063	0.762 ± 0.021	0.645 ± 0.106	$\mathbf{0.623 \pm 0.073}$
MachineCPU	0.480 ± 0.010	0.655 ± 0.045	0.611 ± 0.056	0.572 ± 0.013	0.415 ± 0.096	$\mathbf{0.385 \pm 0.056}$
Boston	0.560 ± 0.020	0.561 ± 0.026	0.569 ± 0.025	0.541 ± 0.009	0.534 ± 0.024	$\mathbf{0.512 \pm 0.040}$
Abalone	0.740 ± 0.020	0.732 ± 0.007	0.736 ± 0.011	0.721 ± 0.002	0.532 ± 0.049	$\mathbf{0.482 \pm 0.011}$
Bank	0.745 ± 0.003	0.751 ± 0.005	$\mathbf{0.744 \pm 0.005}$	0.751 ± 0.001	0.750 ± 0.008	0.745 ± 0.003
Computer	0.472 ± 0.020	0.473 ± 0.005	0.462 ± 0.005	$\mathbf{0.451 \pm 0.002}$	0.510 ± 0.010	0.510 ± 0.004
California	0.643 ± 0.005	0.639 ± 0.003	0.640 ± 0.003	$\mathbf{0.613 \pm 0.001}$	0.680 ± 0.007	0.682 ± 0.005
Census	0.711 ± 0.020	0.705 ± 0.002	0.699 ± 0.002	$\mathbf{0.688 \pm 0.001}$	0.735 ± 0.014	0.724 ± 0.004

Table 3: Mean absolute error (MAE) results along with standard deviations ($\pm$) across 20 runs of the ordinal regression benchmark datasets given in Table 1. Our a-OGMLVQ approach is compared with KDLOR reported in [19], SVORIM and SVOREX reported in [15], RED-SVM reported in [17], and p-OGMLVQ reported in [3]. The best performance for each dataset has been boldfaced.

Dataset	KDLOR	SVOR-IMC	SVOR-EXC	RED-SVM	p-OGMLVQ	a-OGMLVQ
Pyrimidines	1.100 ± 0.100	1.294 ± 0.204	1.331 ± 0.193	1.304 ± 0.040	$\mathbf{0.985 \pm 0.169}$	1.114 ± 0.208
MachineCPU	0.690 ± 0.015	0.990 ± 0.115	0.986 ± 0.127	0.842 ± 0.022	0.630 ± 0.176	$\mathbf{0.514 \pm 0.083}$
Boston	$\mathbf{0.700 \pm 0.035}$	0.747 ± 0.049	0.773 ± 0.049	0.732 ± 0.013	0.731 ± 0.050	0.702 ± 0.057
Abalone	1.400 ± 0.050	1.361 ± 0.013	1.391 ± 0.021	1.383 ± 0.004	0.731 ± 0.068	$\mathbf{0.712 \pm 0.028}$
Bank	1.450 ± 0.020	$\mathbf{1.393 \pm 0.011}$	1.512 ± 0.017	1.404 ± 0.002	1.462 ± 0.009	1.460 ± 0.014
Computer	0.601 ± 0.025	0.596 ± 0.008	0.602 ± 0.009	$\mathbf{0.565 \pm 0.002}$	0.698 ± 0.023	0.687 ± 0.008
California	$\mathbf{0.907 \pm 0.004}$	1.008 ± 0.005	1.068 ± 0.005	0.940 ± 0.001	1.208 ± 0.018	1.204 ± 0.009
Census	1.213 ± 0.003	$\mathbf{1.205 \pm 0.007}$	1.270 ± 0.007	1.143 ± 0.002	1.582 ± 0.018	1.526 ± 0.038

Table 4: Mean ranking for the generalization sets of the ordinal benchmark datasets.

Method	$\bar{R}_{MZE}$	$\bar{R}_{MAE}$
KDLOR	3.94	2.75
SVORIM	4.19	3.25
SVORIEX	3.75	5.00
RED-SVM	3.19	2.88
p-OGMLVQ	3.69	3.88
a-OGLLVQ	2.25	3.25

Table 5: Real ordinal regression datasets used for the experiments (n is the number of patterns, d is the number of attributes and K is the number of classes).

Dataset	n	d	K	Ordered Class Distribution
automobile	205	71	6	(3,22,67,54,32,27)
bondrate	57	37	5	(6,33,12,5,1)
contact-lenses	24	6	3	(15,5,4)
eucalyptus	736	91	5	(180,107,130,214,105)
newthyroid	215	5	3	(30,150,35)
pasture	36	25	3	(12,12,12)
squash-stored	52	51	3	(23,21,8)
squash-unstored	52	52	3	(24,24,4)
tae	151	54	3	(49,50,52)
winequality-red	1599	11	6	(10,53,681,638,199,18)

Most importantly, our approach has the best mean rank in terms of the most relevant measure MMAE (imballanced classes). Again, our approach is consistently better than p-OGMLVQ in all the three performance measures.

Overall, our method retains a competitive edge when compared with the alternative methods considered in this study. Of course, there will be situations where one method is better than the other, but the overall picture emerging from the experiments is that our methods clearly have competitive performance with other methods, while retaining the advantages of prototype based models - e.g. interpretability, natural extension to relevance and metric learning etc. Besides generalization performance, typically there are other criteria to be considered in real world machine learning applications - for example training complexity or model interpretability. In terms of generalization performance, there are many application scenarios where LVQ performs on par with e.g. SVM. Those two approaches are derived in different frameworks and based on different principles. SVM focuses on boundaries of receptive fields and hence can construct more complex decision boundaries naturally, but at the cost of much more complex, less interpretable models and sufficient data. Often, it is preferable to have simple, in-

Table 6: Mean zero-one error (MZE) results along with standard deviations (±) across 30 runs of the real ordinal datasets. The best performance for each dataset has been boldfaced.

Dataset/Method	KDLOR	SVORIM	SVOREX	RED-SVM	p-OGMLVQ	a-OGMLVQ
automobile	**0.228 ± 0.058**	0.361 ± 0.076	0.335 ± 0.068	0.316 ± 0.055	0.325 ± 0.099	0.275 ±0.055
bondrate	0.458 ± 0.087	0.453 ± 0.092	**0.447 ±0.096**	**0.447 ± 0.073**	0.491 ± 0.090	0.477 ± 0.112
contact-lenses	0.411 ± 0.174	0.367 ± 0.127	0.350 ± 0.127	0.300 ± 0.111	0.233 ± 0.112	**0.222 ± 0.126**
eucalyptus	0.389 ± 0.028	0.361 ± 0.028	0.353 ± 0.029 ±	**0.349 ± 0.024**	0.368 ± 0.025	0.367 ± 0.026
newthyroid	**0.028 ± 0.019**	0.031 ± 0.021	0.033 ± 0.022	0.031 ± 0.022	0.055 ± 0.025	0.043 ± 0.021
pasture	0.322 ± 0.125	0.333 ± 0.120	0.370 ± 0.1	0.352 ± 0.134	**0.304 ± 0.109**	0.314 ± 0.105
squash-stored	**0.297 ± 0.112**	0.361 ± 0.118	0.372 ±0.113	0.336 ± 0.104	0.394 ± 0.124	0.389 ± 0.133
squash-unstored	**0.172 ± 0.104**	0.236 ± 0.103	0.282 ± 0.128	0.251 ± 0.086	0.366 ± 0.136	0.271 ± 0.106
tae	0.445 ± 0.052	**0.410 ± 0.066**	0.419 ± 0.060	0.478 ± 0.074	0.440 ± 0.063	0.428 ± 0.069
winequality-red	0.397 ± 0.017	0.370 ± 0.022	**0.361 ± 0.022**	0.392 ± 0.022	0.415 ± 0.016	0.415 ± 0.024

Table 7: Mean absolute error (MAE) results along with standard deviations (±) across 30 runs of the real ordinal datasets. The best performance for each dataset has been boldfaced.

Dataset/Method	KDLOR	SVOR-IMC	SVOR-EXC	RED-SVM	p-OGMLVQ	a-OGMLVQ
automobile	**0.334 ± 0.076**	0.424 ± 0.090	0.408 ± 0.089	0.393 ± 0.073	0.448 ± 0.122	0.403 ± 0.104
bondrate	0.587 ± 0.107	0.591 ± 0.102	0.573 ± 0.121	0.598 ± 0.008	0.564 ± 0.121	**0.536 ± 0.137**
contact-lenses	0.539 ± 0.208	0.506 ± 0.167	0.489 ± 0.185	0.378 ± 0.169	0.338 ± 0.172	**0.327 ± 0.188**
eucalyptus	0.424 ± 0.032	0.395 ± 0.035	0.392 ± 0.031	**0.380 ± 0.027**	0.422 ± 0.030	0.419 ± 0.032
newthyroid	**0.028 ± 0.019**	0.031 ± 0.021	0.033 ± 0.022	0.032 ± 0.022	0.055 ± 0.024	0.043 ± 0.021
pasture	0.322 ± 0.125	0.333 ± 0.120	0.370 ± 0.125	0.359 ± 0.142	**0.307 ± 0.116**	0.318 ± 0.112
squash-stored	**0.308 ± 0.128**	0.372 ± 0.126	0.382 ± 0.139	0.346 ± 0.110	0.415 ± 0.141	0.410 ±0.151
squash-unstored	**0.172 ± 0.104**	0.239 ± 0.109	0.282 ± 0.128	0.251 ± 0.086	0.394 ± 0.158	0.227 ± 0.096
tae	0.473 ± 0.069	**0.461 ± 0.081**	0.485 ± 0.078	0.515 ± 0.087	0.551 ± 0.090	0.536 ± 0.096
winequality-red	0.443 ± 0.019	**0.406 ± 0.022**	0.408 ± 0.023	0.419 ± 0.021	0.458 ± 0.020	0.458 ± 0.022

Table 8: Mean macro-averaged absolute error (MMAE) results along with standard deviations ($\pm$) across 30 runs of the real ordinal datasets. The best performance for each dataset has been boldfaced.

Dataset/Method	KDLOR	SVOR-IMC	SVOR-EXC	RED-SVM	p-OGMLVQ	a-OGMLVQ
automobile	**0.345 ± 0.104**	0.518 ± 0.096	0.523 ± 0.105	0.468 ± 0.096	0.482 ± 0.125	0.446 ± 0.132
bondrate	1.037 ± 0.270	1.114 ± 0.233	1.072 ± 0.217	1.184 ± 0.225	0.768 ± 0.243	**0.737 ± 0.251**
contact-lenses	0.519 ± 0.280	0.589 ± 0.259	0.517 ± 0.303	0.385 ± 0.198	0.243 ± 0.166	**0.221 ± 0.182**
eucalyptus	0.426 ± 0.038	0.420 ± 0.043	**0.411 ± 0.030**	0.414 ± 0.030	0.450 ± 0.035	0.477 ± 0.035
newthyroid	0.059 ± 0.040	0.055 ± 0.042	**0.054 ± 0.042**	0.057 ± 0.049	0.124 ± 0.056	0.097 ± 0.051
pasture	0.322 ± 0.125	0.333 ± 0.120	0.370 ± 0.125	0.359 ± 0.142	**0.307 ± 0.116**	0.318 ± 0.112
squash-stored	**0.349 ± 0.156**	0.427 ± 0.148	0.433 ± 0.172	0.391 ± 0.149	0.415 ± 0.171	0.411 ± 0.181
squash-unstored	0.309 ± 0.180	0.367 ± 0.140	0.426 ± 0.157	0.348 ± 0.159	0.488 ± 0.187	**0.228 ± 0.130**
tae	**0.471 ± 0.070**	0.459 ± 0.081	0.484 ± 0.079	0.513 ± 0.086	0.553 ± 0.091	0.537 ±0.096
winequality-red	1.258 ± 0.069	1.093 ± 0.072	1.095 ± 0.067	**1.068 ± 0.069**	1.078 ± 0.044	**1.069 ± 0.041**

Table 9: Mean ranking for the generalization sets of the real ordinal datasets.

Method	$\bar{R}_{MZE}$	$\bar{R}_{MAE}$	$\bar{R}_{MMAE}$
KDLOR	3.2000	2.900	3.100
SVORIM	3.150	3.200	3.900
SVORIEX	3.450	3.700	4.100
RED-SVM	3.000	3.300	3.200
p-OGMLVQ	4.550	4.550	3.900
a-OGMLVQ	3.650	3.350	2.800

terpretable models with generalization performance approaching that of more complex models, while retaining the advantage of constant (as opposed to linear) time classification effort and linear (as opposed to cubic) time training effort. Last but not least, LVQ models are particularly well suited for life-long learning.

5. Conclusion

We have proposed a new approach to classify data with ordered categories within the learning vector quantization framework. This approach extends the existing p-OGMLVQ method to more natural cost function and more intuitive prototype updating rules.

In contrast to p-OGMLVQ, which pairs the identified correct and incorrect prototypes in distinct couples and then effectively defines a cost for each such couple, our approach defines a single unified cost for all the identified prototypes. Consequently, the updating rule derived from our cost function explicitly reflects the global ordinal relations amongst the prototype classes. Furthermore, we proposed to automatically adapt the bandwidth parameters of the weighting functions during training.

Empirical experiments have been conducted to verify the effectiveness of our ordinal approach. The experimental results show that the performance of our ordinal approach is better than p-OGMLVQ. We also compared our approach with other ordinal classifiers. The results show that our approach could obtain comparable performance sometimes superior performance, especially in terms of (the most relevant) MMAE performance measure.

Acknowledgements

Peter Tiňo was supported by the EPSRC grant EP/L000296/1. Dr. Fengzhen Tang would like to thank Dr. Bailu Si from Shenyang Institute of Automation, Chinese Academy of Sciences and Dr. Kerstin Bunte from University of Birmingham for the discussion and comments.

[1] H.-T. Lin, L. Li, Reduction from cost-sensitive ordinal ranking to weighted binary classification, Neural Computation 24 (5) (2012) 1329–1367.

[2] C.-W. Seah, I. W. Tsang, Y.-S. Ong, Transductive ordinal regression, IEEE Transactionss on Neural Networks and Learning Systems 23 (7) (2012) 1074–1086.

[3] S. Fouad, P. Tiňo, Adaptive metric learning vector quantization for ordinal classification, Neural Computation 24 (11) (2012) 2825–2851.

[4] J. Sánchez-Monedero, P. A. Gutiérrez, P. Tiňo, C. Hervás-Martínez, Exploitation of pairwise class distances for ordinal classification, Neural Computation 25 (9) (2013) 2450–2485.

[5] P. A. Guti érrez, P. C. H.-M. Tiňo, Ordinal regression neural networks based on concentric hyperspheres, Neural Networks 59 (0) (2014) 51 – 60.

[6] P. Tino, C. Schittenkopf, G. Dorffner, Volatility trading via temporal pattern recognition in quantized financial time series, Pattern Analysis and Application 4 (4) (2001) 283–299.

[7] H. Dikkers, L. Rothkrantx, Support vector machines in ordinal classification: an application to corporate credit scoring, Neural Network World 3 (1) (2012) 59–70.

[8] R. Lall, M. J. Campbell, W. S. J., K. Morgan, A review of ordinal regression models applied on health-related quality of life assessments., Statistic Methods Medical Research 11 (2012) 49 – 67.

[9] H. Yan, Cost-sensitive ordinal regression for fully automatic facial beauty assessment, Neurocomputing 129 (2014) 334 –342.

[10] S. Kramer, B. Widmer, G.and Pfahringer, M. de Groeve, Prediction of ordinal classes using regression trees, Fundamenta Informaticae 47 (1-2) (2001) 1–13.

[11] E. Frank, M. Hall, A simple approach to ordinal classification, in: Proceedings of the European Conference on Machine Learning, Vol. 2167, 2001, pp. 145–165.

[12] R. Herbrich, T. Graepel, K. Obermayer, Large Margin Rank Boundaries for Ordinal regression, MIT Press, Cambridge, MA, 2000, Ch. 7, pp. 115–132.

[13] K. Crammer, Y. Singer, Pranking with ranking, in: Advances in Neural Information Processing Systems 14, MIT Press, 2001, pp. 641–647.

[14] A. Shashua, A. Levin, Ranking with large margin principle: Two approaches, in: Advances in Neural Information Processing System 15, 2003, pp. 937–944.

[15] W. Chu, S. S. Keerthi, New approaches to support vector ordinal regression, in: Proceedings of the 22nd International Conference on Machine learning, (ICML'05), 2005, pp. 145–152.

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

[17] L. Li, H. Lin, Ordinal regression by extended binary classification, in: Advances in Neural Information Processing System, 2006, pp. 865–872.

[18] W. Chu, Z. Ghahramani, Gaussian process for ordinal regression, Journal of Machine Learning Research 6 (2005) 1019–1041.

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

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

[21] T. Kohonen, Learning vector quantization for pattern recognition, Technical report tkkf-a601, Helsinki Univeristy of Technology, Espoo, Finland. (1986).

[22] T. Kohonen, The handbook of Brain Theory and Neural Networks, MIT Press, 1995, Ch. Learning vector quantization, pp. 537–540.

[23] A. Sato, K. Yamada, Generalized learning vector quantization, in: D. Touretzky, M. Mozer, M. Hasselmo (Eds.), Advances in Neural Information Processing System 8, MIT Press, 1996, pp. 423–429.

[24] H.-T. Lin, L. Li, Reduction from cost-sensitive ordinal ranking to weighted binary classification, Neural Computation.

[25] B. Hammer, Villmann, Generalized relevance learning vector quantization, Neural Networks 15 (8–9) (2002) 1059–1068.

[26] P. Schneider, M. Biehl, B. Hammer, Hyperparameter learning in probabilistic prototype-based models, Neurocomputing 73 (7–9) (2010) 1117–1124.

[27] J. Sánchez-Monedero, P. A. Gutiérrez, P. Tiňo, C. Hervás-Martínez, Exploitation of pairwise class distances for ordinal classification, Neural Computation 25 (9) (2013) 2450–2485.

Appendix: Proof of Vanishing Gradient

In this appendix we will show that the terms (48), (49), (52) and (53) of the gradient of our error function vanish.

Note that derivative of the Heaviside function is the delta function δ, which is a symmetric function with $\delta(x) = 0$ for $x \neq 0$ and $\int_{\mathbb{R}} \delta(x) = 1$. We will first concentrate on terms (48) and (49). The integrand in (48) can be expanded as

$$\sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} \frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} H(g(j,\ell)+f(\ell)) \frac{\partial \mathcal{I}(j,l)}{\partial \boldsymbol{w}_i} \tag{68}$$

$$+ \sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} \frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} \mathcal{I}(j,l)\delta(g(j,\ell)+f(\ell)) \frac{\partial g(j,\ell)}{\partial \boldsymbol{w}_i} \tag{69}$$

$$+ \sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} \frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} \mathcal{I}(j,l)\delta(g(j,\ell)+f(\ell)) \frac{\partial f(\ell)}{\partial \boldsymbol{w}_i}. \tag{70}$$

First, we show that (68) vanishes. Denoting

$$\frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} \cdot H(g(j,\ell)+f(\ell)) \cdot \delta\left(\sum_{\boldsymbol{w}_q \in W_l} H(d_q - d_j) - |W_l|\right)$$

by $T(j,l,\ell)$, (68) can then be rewritten as

$$\begin{aligned}
&\sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} \Bigg[\frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} \cdot H(g(j,\ell)+f(\ell)) \\
&\delta\left(\sum_{\boldsymbol{w}_q \in W_{c_i}} H(d_q - d_j) - |W_{c_i}|\right) \sum_{\boldsymbol{w}_q \in W_l} \delta(d_q - d_j)\left(\frac{\partial d_q}{\partial \boldsymbol{w}_i} - \frac{\partial d_j}{\partial \boldsymbol{w}_i}\right)\Bigg] \\
= &\sum_{|l-\ell|\leq\tau} \sum_{\substack{\boldsymbol{w}_j \in W_l \\ j\neq i}} T(j,l,\ell)\delta(d_i - d_j) 2D_i \\
&+T(i,c_i,\ell) \sum_{|l-\ell|\leq\tau} \sum_{\substack{\boldsymbol{w}_q \in W_l \\ q\neq i}} \delta(d_q - d_i)(-2D_i).
\end{aligned}$$

This expression vanishes since δ is symmetric and non-vanishing only for $d_i = d_j$ and $d_q = d_i$.

We now show that (69) vanishes. Denoted by $Q(j,l,\ell)$ the term

$$\frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell)+\tilde{E}(\ell))^2} \cdot \mathcal{I}(j,l) \cdot \delta(g(j,\ell)+f(\ell)),$$

then, (69) yeilds:

$$\sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l} Q(j,l,\ell)\sum_{|s-\ell|\leq\tau}\sum_{\boldsymbol{w}_q\in W_s} H(d_q-d_j)\frac{\partial\mathcal{I}(q,s)}{\partial\boldsymbol{w}_i} \tag{71}$$

$$+\quad \sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l} Q(j,l,\ell)\sum_{|s-\ell|\leq\tau}\sum_{\boldsymbol{w}_q\in W_s} \mathcal{I}(q,s)\delta(d_q-d_j)\left[\frac{\partial d_q}{\partial\boldsymbol{w}_i}-\frac{\partial d_j}{\partial\boldsymbol{w}_i}\right] \tag{72}$$

Let

$$A(q,s)=H(d_q-d_j)\cdot\delta\left(\sum_{\boldsymbol{w}_t\in W_s}H(d_t-d_q)-|W_s|\right),$$

(71) vanishes because

$$\begin{aligned}
&\sum_{|s-\ell|\leq\tau}\sum_{\boldsymbol{w}_q\in W_s} H(d_q-d_j)\frac{\partial\mathcal{I}(q,s)}{\partial\boldsymbol{w}_i}\\
=\quad &\sum_{|s-\ell|\leq\tau}\sum_{\boldsymbol{w}_q\in W_s} H(d_q-d_j)\,\delta\left(\sum_{\boldsymbol{w}_t\in W_s}H(d_t-d_q)-|W_s|\right)\\
&\sum_{\boldsymbol{w}_q\in W_s}\delta(d_t-d_q)\left(\frac{\partial d_t}{\partial\boldsymbol{w}_i}-\frac{\partial d_q}{\partial\boldsymbol{w}_i}\right)\\
=\quad &\sum_{|s-\ell|\leq\tau}\sum_{\substack{\boldsymbol{w}_q\in W_s\\ q\neq i}} A(q,s)\delta(d_i-d_q)2D_i\\
&+A(i,c_i)\sum_{|s-c|\leq\tau}\sum_{\substack{\boldsymbol{w}_t\in W_s\\ t\neq i}}\delta(d_t-d_i)(-2D_i)
\end{aligned}$$

vanishes since δ is symmetric and non-vanishing only for $d_i=d_q$ and $d_t=d_i$.

(72) will also vanish, since it leads to:

$$\begin{aligned}
&\sum_{|l-\ell|\leq\tau}\sum_{\substack{\boldsymbol{w}_j\in W_l\\ j\neq i}} Q(j,l,\ell)\mathcal{I}(q,s)\delta(d_i-d_j)2D_i\\
+\quad &Q(i,c_i,\ell)\sum_{|l-\ell|\leq\tau}\sum_{\substack{\boldsymbol{w}_q\in W_s\\ q\neq i}}\mathcal{I}(q,s)\delta(d_q-d_i)(-2D_i).
\end{aligned}$$

Since both (71) and (72) are vanishing, (69) vanishes.

Finally, we show (70) vanishes. (70) leads to

$$\begin{aligned}
&\sum_{|l-\ell|\leq\tau}\sum_{\boldsymbol{w}_j\in W_l} Q(j,l,\ell)\left(\frac{\partial\tilde{N}(\ell)}{\partial\boldsymbol{w_i}}H(N(\ell)-\tilde{N}(\ell))\right.\\
&\left.+\,(N(\ell)-\tilde{N}(\ell))\delta(N(\ell)-\tilde{N}(\ell))\frac{\partial\tilde{N}(\ell)}{\partial\boldsymbol{w_i}}\right).
\end{aligned}$$

This term vanishes because

$$\frac{\partial \tilde{N}(\ell)}{\partial \boldsymbol{w_i}} = \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_j \in W_l} \delta(D - d_k) \left(\frac{\partial D}{\partial \boldsymbol{w_i}} - \frac{\partial d_k}{\partial \boldsymbol{w_i}} \right) \tag{73}$$

vanishes (δ is symmetric and non-vanishing only for $D = d_k$).

In summary, (68), (69), and (70) all vanish and hence the integrand in (48) vanishes as well. We will now show with analogous arguments that the integrand of (49) vanishes. We have:

$$\sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \frac{2\alpha_k d_k E(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} H(\tilde{g}(k,\ell) + \tilde{f}(\ell)) \frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \boldsymbol{w}_i} \tag{74}$$

$$+ \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \frac{2\alpha_k d_k E(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} \tilde{\mathcal{I}}(k,D) \delta(\tilde{g}(k,\ell) + \tilde{f}(\ell)) \frac{\partial \tilde{g}(k,\ell)}{\partial \boldsymbol{w}_i} \tag{75}$$

$$+ \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \frac{2\alpha_k d_k E(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} \tilde{\mathcal{I}}(k,D) \delta(\tilde{g}(k,\ell) + \tilde{f}(\ell)) \frac{\partial \tilde{f}(\ell)}{\partial \boldsymbol{w}_i}. \tag{76}$$

As before, we argue that (74), (75), and (76) vanish, separately.

First, (74) vanishes, since

$$\frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \boldsymbol{w}_i} = \delta(D - d_k)(\frac{\partial D}{\partial \boldsymbol{w}_i} - \frac{\partial d_k}{\partial \boldsymbol{w}_i}) = 0. \tag{77}$$

This is because δ is symmetric and non-vanishing only for $D = d_k$. Let us move on to (75). Denoting

$$\frac{2\alpha_k d_k E(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} \tilde{\mathcal{I}}(k,D) \delta(\tilde{g}(k,\ell) + \tilde{f}(\ell))$$

by $\tilde{G}(k,\ell)$, (75) leads to

$$\sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \tilde{G}(k,\ell) \sum_{|s-\ell|>\tau} \sum_{\boldsymbol{w}_q \in W_s} H(d_q - d_k) \frac{\partial \tilde{\mathcal{I}}(q,D)}{\partial \boldsymbol{w}_i} \tag{78}$$

$$+ \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \tilde{G}(k,\ell) \sum_{|s-\ell|>\tau} \sum_{\boldsymbol{w}_q \in W_s} \tilde{\mathcal{I}}(q,D) \delta(d_q - d_k)(\frac{\partial d_q}{\partial \boldsymbol{w}_i} - \frac{\partial d_k}{\partial \boldsymbol{w}_i}) \tag{79}$$

Now, (78) vanishes, since we have already shown that $\frac{\partial \tilde{\mathcal{I}}(q,D)}{\partial \boldsymbol{w}_i} = 0$. The term (79) leads to

$$\sum_{|l-\ell|>\tau} \sum_{\substack{\boldsymbol{w}_k \in W_l \\ k \neq i}} \tilde{G}(k,\ell) \tilde{\mathcal{I}}(i,D) \delta(d_i - d_k) 2D_i$$

$$+ \quad \tilde{G}(i,\ell) \sum_{|s-\ell|>\tau} \sum_{\substack{\boldsymbol{w}_q \in W_s \\ q \neq i}} \tilde{\mathcal{I}}(q,D) \delta(d_q - d_i)(-2D_i),$$

which vanishes, since δ is symmetric and non-vanishing only for $d_i = d_k$ and $d_q = d_i$.

Finally, (76) vanishes as it leads to:

$$\sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \tilde{G}(k,\ell) \left(-H(\tilde{N}(\ell) - N(\ell)) \frac{\partial \tilde{N}(\ell)}{\partial \boldsymbol{w_i}} \right.$$
$$\left. - ((\tilde{N}(\ell) - N(\ell))\delta(\tilde{N}(\ell) - N(\ell)) \frac{\partial \tilde{N}(\ell)}{\partial \boldsymbol{w_i}} \right)$$

and this term vanishes, since $\frac{\partial \tilde{N}(\ell)}{\partial \boldsymbol{w_i}} = 0$, as shown in (73).

We now turn our attention to terms (52) and (53):

$$\sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} \frac{2\alpha_j d_j \tilde{E}(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} H(g(j,\ell) + f(\ell)) \frac{\partial \mathcal{I}(j,l)}{\partial \Omega}) \tag{80}$$

$$+ \sum_{|l-\ell|\leq\tau} \sum_{\boldsymbol{w}_j \in W_l} Q(j,l,\ell) \left(\frac{\partial g(j,\ell)}{\partial \Omega} + \frac{\partial f(\ell)}{\partial \Omega} \right) \tag{81}$$

$$- \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \frac{2\alpha_k d_k E(\ell)}{(E(\ell) + \tilde{E}(\ell))^2} H(\tilde{g}(k,\ell) + \tilde{f}(\ell)) \frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \Omega} \tag{82}$$

$$- \sum_{|l-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_l} \tilde{G}(k,\ell) \left(\frac{\partial \tilde{g}(k,\ell)}{\partial \Omega} + \frac{\partial \tilde{f}(\ell)}{\partial \Omega} \right). \tag{83}$$

The partial derivatives $\frac{\partial \mathcal{I}(j,l)}{\partial \Omega})$, $\frac{\partial g(j,\ell)}{\partial \Omega}$, $\frac{\partial f(\ell)}{\partial \Omega}$, $\frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \Omega}$, $\frac{\partial \tilde{g}(k,\ell)}{\partial \boldsymbol{w}_i}$, and $\frac{\partial \tilde{f}(\ell)}{\partial \boldsymbol{w}_i}$ all vanish as

$$\frac{\partial \mathcal{I}(j,l)}{\partial \Omega} = \delta(\sum_{\boldsymbol{w}_q \in W_l} H(d_q - d_j) - |W_l|)$$
$$\cdot \sum_{\boldsymbol{w}_q \in W_l} \delta(d_q - d_j)(\frac{\partial d_q}{\partial \Omega} - \frac{\partial d_j}{\partial \Omega}), \tag{84}$$

$$\frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \Omega} = \delta(D - d_k) \left(\frac{\partial D}{\partial \Omega} - \frac{\partial d_k}{\partial \Omega} \right). \tag{85}$$

and δ is symmetric and non-vanishing only for $d_q = d_j$, and $D = d_k$. We have:

$$\begin{aligned}
\frac{\partial g(j,\ell)}{\partial \Omega} &= \sum_{|s-\ell|\le\tau} \sum_{\boldsymbol{w}_j \in W_s} \Big(\frac{\partial \mathcal{I}(j,l)}{\partial \Omega} H(d_q - d_j) \\
&\quad + \mathcal{I}(q,s)\delta(d_q - d_j)(\frac{\partial d_q}{\partial \Omega} - \frac{\partial d_j}{\partial \Omega}) \Big) \\
&= \sum_{|s-\ell|\le\tau} \sum_{\boldsymbol{w}_j \in W_s} \mathcal{I}(q,s)\delta(d_q - d_j)(\frac{\partial d_q}{\partial \Omega} - \frac{\partial d_j}{\partial \Omega}), \\
\frac{\partial \tilde{g}(k,\ell)}{\partial \Omega} &= \sum_{|s-\ell|>\tau} \sum_{\boldsymbol{w}_q \in W_s} \Big(\frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \Omega} H(d_q - d_k) \\
&\quad + \tilde{\mathcal{I}}(q,D)\delta(d_q - d_k)(\frac{\partial d_q}{\partial \Omega} - \frac{\partial d_k}{\partial \Omega}) \Big) \\
&= \sum_{|s-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_s} \tilde{\mathcal{I}}(q,D)\delta(d_q - d_k)(\frac{\partial d_q}{\partial \Omega} - \frac{\partial d_k}{\partial \Omega}).
\end{aligned}$$

The last two terms vanish since δ is symmetric and non-vanishing only for $d_q = d_j$ and $d_q = d_k$.

From

$$\frac{\partial \tilde{N}(\ell)}{\partial \Omega} = \sum_{|s-\ell|>\tau} \sum_{\boldsymbol{w}_k \in W_s} \frac{\partial \tilde{\mathcal{I}}(k,D)}{\partial \Omega} = 0,$$

it follows that

$$\begin{aligned}
\frac{\partial f(\ell)}{\partial \Omega} &= \frac{\partial \tilde{N}(\ell)}{\partial \Omega} H(N(\ell) - \tilde{N}(\ell)) \\
&\quad + (N(\ell) - \tilde{N}(\ell))\delta(N(\ell) - \tilde{N}(\ell)) \frac{\partial \tilde{N}(\ell)}{\partial \Omega} \\
&= 0, \\
\frac{\partial \tilde{f}(\ell)}{\partial \Omega} &= -\frac{\partial \tilde{N}(\ell)}{\partial \Omega} H(\tilde{N}(\ell) - N(\ell)) \\
&\quad - (\tilde{N}(\ell) - N(\ell))\delta(\tilde{N}(\ell) - N(\ell)) \frac{\partial \tilde{N}(\ell)}{\partial \Omega} \\
&= 0.
\end{aligned}$$