Chapter 2

Learning Vector Quantization

Abstract

This chapter serves as a general introduction to the concept of prototype based classification. Under the family of Learning Vector Quantization (LVQ), we introduce different variants we used in our study. In our findings, Generalized Matrix Learning Vector Quantization (GMLVQ) was outstanding in handling both classification and the weighting or selection of features. By name, the classification task of this learning model is based on highly ranked features. In this chapter, we explain the working of this algorithm.

2.1 Introduction

The previous chapter introduced supervised learning, a major branch of machine learning where each example is a pair consisting of an input object and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

A wide range of supervised learning algorithms are available but the choice of using one algorithm over the other will depend on the problem at hand.

Learning Vector Quantization (LVQ) is a prototype-based supervised classification algorithm and a special case of an artificial neural network, more precisely that applies a winner-take-all (Maass 2000) approach. The classification algorithms was first introduced in (Kohonen 1986) and since then various modifications have been suggested in the literature. LVQ and its variants are a family of classifiers that have found much popularity within the machine learning field for various reasons. For review of relevance learning, LVQ and prototype-based systems in general see, for instance (Biehl et al. 2016).

In this thesis we use a special kind of dataset that is attracting more scientists to investigate crop disease using spectral data. By nature of spectral data, it presents us with high dimensions of the input space and this became our critical consideration. The input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. Hence, choosing a classifier that handles the aspect of feature selection and dimensionality reduction of the input space was of high importance.

Similar to the LVQ family is the k-Nearest Neighbour algorithm (Cover and Hart 1967) which skips the training phase and directly uses all the available training data in the classification of a new data point. The algorithm is based on the intuitive "Birds of the same feather flock together" meaning that similar things exist in close proximity. To classify a new data point, k of its closest neighbours are obtained using some measure of closeness, for example, Euclidean distance, and the average class of these k neighbours is awarded to the new data point. However, the algorithm gets significantly slower as the number of training data points N increases since the distance to each point needs to be calculated every time.

Several LVQ variants were proposed e.g LVQ1 and LVQ2.1 by Kohonen aiming at faster convergence and better approximation of Bayesian decision boundaries, respectively. Other variants have been suggested in (Sato and Yamada 1995, Schneider et al. 2007).

In relation to our current study, successful studies were conducted, e.g. (Mwebaze et al. 2011) developed the corresponding divergence-based LVQ (DLVQ) schemes applied to cassava dataset to identify cassava diseases. A closely related study also in (Mwebaze and Biehl 2016) uses prototype-based classification by exploring the right configuration of type of algorithm and type of features extracted from the leaves to optimally diagnose crop disease in cassava plants. More studies on the use of LVQ schemes are highlighted in different chapters in this thesis.

2.2 Learning Vector Quantization and its variants

In this section, we give a description of LVQ variants we investigated on including: Generalized LVQ, Generalized Relevance LVQ and Generalized Matrix LVQ that was very applicable in answering some of our research questions. By definition, we will consider data sets of the form:

$$\{\mathrm{x}^{\mu}, y^{\mu}\}_{\mu=1}^{\mathrm{P}} \tag{2.1}$$

where $\mathrm{x}^{\mu} \in \mathbb{R}^{N}$ are feature vectors and the labels $y^{\mu} \in 1, 2, ...C$ specify their class membership.

The LVQ system is defined by a set of M prototype vectors $\mathrm{w}^{j} \in \mathbb{R}^{N}$ which carry labels $c(\mathrm{w}^{j}) \in \{1, 2, ...C\}$ such that $W = \{\mathrm{w}^{j}, c(\mathrm{w}^{j})\}_{j=1}^{M}$.

The system can be set up with one or more prototype vectors per class. Prototype vectors are identified in the feature space and ideally serve as typical representatives of their classes.

A nearest prototype classifier (NPC) assigns a given feature vector $\mathrm{x} \in \mathbb{R}^{N}$ to the *closest* prototype with respect to some meaningful distance measure.

Most frequently, standard Euclidean distance $d(\mathrm{w}, \mathrm{x})$ is employed. The corresponding NPC assigns x to the class $c(\mathrm{w}^{\mathrm{J}})$ of the closest prototype with $d(\mathrm{x}, \mathrm{w}^{\mathrm{J}}) \leqslant d(\mathrm{x}, \mathrm{w}^{\mathrm{j}})$ for all j.

2.2.1 Classical LVQ

This is the a heuristic algorithm introduced in (Kohonen 1986). Prototypes are updated based on how close they are to a presented data point given the class of the prototype and that of the data point. The training process is represented by the following steps:.

1. Randomly select a training sample $(\{\mathrm{x}^{\mu}, y^{\mu}\})$

2. Determine the winning prototype $(\mathrm{w}^{\mathrm{J}})$ with $d(\mathrm{x}, \mathrm{w}^{\mathrm{J}}) \leqslant d(\mathrm{x}, \mathrm{w}^{\mathrm{j}})$, $\mathrm{j} = 1......\mathrm{M}$

3. Update $(\mathrm{w}^{\mathrm{J}})$ according to:

$$\mathrm{w}^{\mathrm{J}} \longleftarrow \mathrm{w}^{\mathrm{J}} + \eta \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{J}}), \quad (\text{if } c(\mathrm{w}^{\mathrm{J}}) = \mathrm{y}).$$
$$\mathrm{w}^{\mathrm{J}} \longleftarrow \mathrm{w}^{\mathrm{J}} - \eta \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{J}}), \quad (\text{if } c(\mathrm{w}^{\mathrm{J}}) \neq \mathrm{y})$$

Should the same feature vector be observed again, w^{J} will give rise to a decreased (increased) distance, if the label $c(\mathrm{w}^{\mathrm{J}})$ and y agree (disagree).

2.2.2 Generalized LVQ

Generalized LVQ (GLVQ) is one key variant of LVQ that was introduced in (Sato and Yamada 1995). The algorithm uses an objective (cost) function in the training of the LVQ system. The advantage of an objective function based LVQ system is that one can use gradient methods (online or batch) to optimize it. With the training data in form $\{\mathrm{x}^{\mu}, y^{\mu}\}_{\mu=1}^{\mathrm{P}}$, the cost function is defined by:

$$E(W) = \sum_{\mu=1}^{p} \Phi\left(\frac{d(\mathrm{x}^{\mu}, \mathrm{w}^{J}) - d(\mathrm{x}^{\mu}, \mathrm{w}^{K})}{d(\mathrm{x}^{\mu}, \mathrm{w}^{J}) + d(\mathrm{x}^{\mu}, \mathrm{w}^{K})}\right) \tag{2.2}$$

where w^{J} denotes the closest correct prototype with $c(\mathrm{w}^{\mathrm{J}}) = y^{\mu}$ and w^{K} is the closest incorrect prototype with $c(\mathrm{w}^{\mathrm{K}}) \neq y^{\mu}$, also termed as winner-takes all rule (Crammer et al. 2002). The logistic sigmoid function Φ determines the active regions of the algorithm: $\Phi(x) = x$. Training constitutes the minimization of $E(W)$ with respect to the model parameters. The learning algorithm is defined in (Sato and Yamada 1995) in terms stochastic gradient descent.

2.2.3 Generalized Matrix LVQ

The cost function in Eq.(2.2) has been extended to other distance metrics. A generalization bound has been derived for GRLVQ which uses an adaptive metric (Hammer and Villmann 2002, Strickert et al. 2001, Villmann et al. 2015). The distance metric is defined as the squared weighted euclidean metric $d^{\lambda}(\mathrm{x}, \mathrm{w}) = \sum_{i}^{N} \lambda_i (\mathrm{x}_i - \mathrm{w}_i)^2$ where $\lambda_i \geqslant 0$ and $\sum_i \lambda_i = 1$.

Another powerful extension of the basic LVQ concept is Generalized Matrix LVQ (GMLVQ). The learning algorithm can be seen by the following steps:

1. Randomly select a training sample $(\{\mathrm{x}^{\mu}, y^{\mu}\})$

2. Determine the closest correct prototype $(\mathrm{w}^{\mathrm{J}})$ with $c(\mathrm{w}^{\mathrm{J}}) = y$, and $d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{J}}) \leqslant d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{j}})$ for all w^{j} with $c(\mathrm{w}^{\mathrm{j}}) = y$, and the closest incorrect prototype, $(\mathrm{w}^{\mathrm{K}})$ with $c(\mathrm{w}^{\mathrm{K}}) \neq y$, and $d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{K}}) \leqslant d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{j}})$, for all w^{j} with $c(\mathrm{w}^{\mathrm{j}}) \neq y$

3. Update (w) according to:

$$\mathrm{w}^{\mathrm{L}} \longleftarrow \mathrm{w}^{\mathrm{J}} + \epsilon \cdot \Lambda \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{J}}), \quad (y = c(\mathrm{w}^{\mathrm{J}}))$$
$$\mathrm{w}^{\mathrm{K}} \longleftarrow \mathrm{w}^{\mathrm{K}} - \epsilon \cdot \Lambda \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{K}}), \quad (y \neq c(\mathrm{w}^{\mathrm{K}}))$$

4. Update Ω according to:

$$\Omega \longleftarrow \Omega + \epsilon \cdot \Omega \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{J}})(\mathrm{x} - \mathrm{w}^{\mathrm{J}})^{\top}, \quad (\text{if } (y = c(\mathrm{w}^{\mathrm{J}})))$$
$$\Omega \longleftarrow \Omega - \epsilon \cdot \Omega \cdot (\mathrm{x} - \mathrm{w}^{\mathrm{K}})(\mathrm{x} - \mathrm{w}^{\mathrm{K}})^{\top}, \quad (\text{if } (y \neq c(\mathrm{w}^{\mathrm{K}})))$$

These processes are followed by a normalization step such that Tr(. . .) = 1 for $\Lambda = \Omega^{\top}\Omega$. $\epsilon \in [0, 1]$ is the learning rate for the metric parameter and the matrix Λ is updated in a such way that the distance $d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{J}})$ is decreased in case of a correct classification, while $d^{\Lambda}(\mathrm{x}, \mathrm{w}^{\mathrm{K}})$ increases, if the sample (x, y) is misclassified.

GMLVQ algorithm proposed in (Schneider et al. 2007, Schneider et al. 2009b) employs a full matrix $\Lambda \in \mathbb{R}^{N \times N}$ of relevances that describes the importance of the individual features in the classification task. Here, the distance measure $d^{\Lambda}(\mathrm{x}, \mathrm{w})$ is defined as:

$$d^{\Lambda}(\mathrm{x}, \mathrm{w}) = (\mathrm{x} - \mathrm{w})^{\top}\Lambda(\mathrm{x} - \mathrm{w}) \tag{2.3}$$

where the parameterization $\Lambda = \Omega^{\top}\Omega$ guarantees that Λ is positive semi-definite and that $d^{\Lambda}(\mathrm{x}, \mathrm{w}) \geqslant 0$ for arbitrary matrices $\Omega \in \mathbb{R}^{N \times N}$. Now the squared distance reads:

$$d^{\Lambda}(\mathrm{x}, \mathrm{w}) = \sum_{ijk}(\mathrm{x}_i - \mathrm{w}_i)\Omega_{ki}\Omega_{kj}(\mathrm{x}_j - \mathrm{w}_j). \tag{2.4}$$

To obtain the adaptation formula, we compute the derivatives with respect to w and Ω. The derivative of d^{Λ} with respect to w yields:

$$\nabla_{\mathrm{w}} d^{\Lambda} = -2\Lambda(\mathrm{x} - \mathrm{w}) = -2\Omega^{\top}\Omega(\mathrm{x} - \mathrm{w}). \tag{2.5}$$

Derivative with respect to single element $\Omega_l m$ gives

$$\begin{aligned}\frac{\partial d^{\Lambda}}{\partial \Omega_{lm}} &= \sum_j (\mathrm{x}_l - \mathrm{w}_l)\Omega_{mj}(\mathrm{x}_j - \mathrm{w}_j) + \sum_i (\mathrm{x}_i - \mathrm{w}_i)\Omega_{il}(\mathrm{x}_m - \mathrm{w}_m) \\ &= (\mathrm{x}_l - \mathrm{w}_l)[\Omega(\mathrm{x} - \mathrm{w})]_m + (\mathrm{x}_m - \mathrm{w}_m)[\Omega(\mathrm{x} - \mathrm{w})]_l\end{aligned} \tag{2.6}$$

Thus, we get the update equations:

$$
\begin{aligned}
\Delta \mathrm{w_J} &= \epsilon \cdot \Phi^{'}(\mu(\mathrm{x})) \cdot \mu^{+}(\mathrm{x}) \cdot \Omega^{\top}\Omega \cdot (\mathrm{x} - \mathrm{w_J}) \\
\Delta \mathrm{w_K} &= -\epsilon \cdot \Phi^{'}(\mu(\mathrm{x})) \cdot \mu^{-}(\mathrm{x}) \cdot \Omega^{\top}\Omega \cdot (\mathrm{x} - \mathrm{w_K})
\end{aligned}
\tag{2.7}
$$

Where

$$
\mu^{+} = \frac{2d^{\Lambda}(\mathrm{x} - \mathrm{w^K})}{(d^{\Lambda}(\mathrm{x} - \mathrm{w^K}) + d^{\Lambda}(\mathrm{x} - \mathrm{w^J}))^2}
$$

$$
\mu^{-} = \frac{2d^{\Lambda}(\mathrm{x} - \mathrm{w^J})}{(d^{\Lambda}(\mathrm{x} - \mathrm{w^J}) + d^{\Lambda}(\mathrm{x} - \mathrm{w^K}))^2}
$$

These updates correspond to the standard Hebb terms of LVQ, pushing the closest correct prototype ($\mathrm{w_J}$) towards the considered data point and the closest wrong prototype ($\mathrm{w_K}$) away from the considered data point. For the update of the matrix elements Ω_{lm} we get

$$
\begin{aligned}
\Delta\Omega_{lm} = -\epsilon \cdot \Phi^{'}(\mu(\mathrm{x})) \cdot \Big(& \mu^{+}(\mathrm{x}) \cdot \Big([\Omega(\mathrm{x} - \mathrm{w_J})]_m (\mathrm{x}_l - \mathrm{w}_{\mathrm{J},l}) + [\Omega(\mathrm{x} - \mathrm{w_J})]_l (\mathrm{x}_m - \mathrm{w}_{\mathrm{J},m}) \Big) \\
& - \mu^{-}(\mathrm{x}) \cdot \Big([\Omega(\mathrm{x} - \mathrm{w_K})]_m (\mathrm{x}_l - \mathrm{w}_{\mathrm{K},l}) + [\Omega(\mathrm{x} - \mathrm{w_K})]_l (\mathrm{x}_m - \mathrm{w}_{\mathrm{K},m}) \Big) \Big)
\end{aligned}
\tag{2.8}
$$

The learning rate for the metric can be chosen independently of the learning rate for the prototypes. After each update, Λ is normalised to prevent the algorithm from degeneration. Therefore, we set $\sum_i \Lambda_{ii} = \sum_{ij} \Omega_{ij}^2 = 1$ which fixes the sum of diagonal elements and, here, the sum of eigenvalues. Learning stops after a maximal number of epochs are reached.

At the end of the learning process the algorithm provides a set of prototypes w_j, their labels $c(\mathrm{w}_j)$, and a task specific discriminative distance d^{Λ}. If features have the same magnitude, the diagonal elements Λ_{ii} of the dissimilarity matrix can be interpreted as overall relevances of every feature i for the classification. The off-diagonal elements Λ_{ij} with $j \neq i$ weigh the pairwise correlations between features i and j. High absolute values in the matrix denote highly relevant features, while values near zero can be seen as less important for the classification accuracy.

The above description corresponds to updates based on stochastic gradient descent described in (Schneider et al. 2009a). In our experiments, we employed a batch gradient descent algorithm which uses an automatic step size adaption scheme aiming at minimizing the cost function in Eq (2.2). This form of update is described in (Papari et al. 2011). Specifically, we have used a publicly available implementation of batch GMLVQ, see (Biehl 2017).

Part I

Disease Diagnosis with Leaf Images