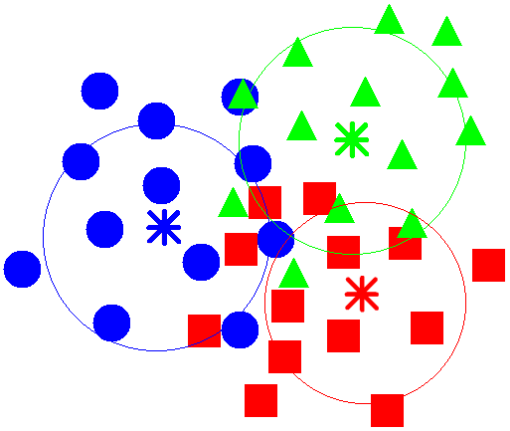
university of groningen / faculty of mathematics and natural sciences

LaMaLVQ: Warping Space



Department of Computing Science

Master Thesis

November 18, 2013

Author: Sander Kelders

Primary Supervisor: Michael Biehl

Abstract

LVQ is a prototype based classification technique. This technique can be extended to MLVQ to include metric learning. LMNN is a classification technique based on kNN and extended to incorporate metric learning. MLVQ and LMNN show many similarities in the way distance is defined. A combination of the techniques is therefore an enticing experiment. LaMaLVQ attempts to combine the metric learning of LMNN and the low space requirements of LVQ into one, by letting the LVQ part position the prototypes and the LaMa part compute an appropriate metric for the prototype configuration. This technique shows some differing but interesting results. For example: letting the two parts share the same optimization goal may seem like a good idea, but can have detrimental effect on classification. Also, LaMaLVQ shows a tendency to push prototypes away from the dataset.

Introduction

Humans are very good at certain seemingly easy tasks. In our every day lives we perform many without even thinking about it: Looking for our keys we find them lying on the table. Going out we take our coat with us because the sky looks like it is about to rain; While doing groceries we pick up the apples and leave the oranges next to them if we do not like oranges. Something as simple as looking for your keys means looking around, identifying objects in the chaos surrounding you, selecting likely candidates to be your keys (not the wall for example), examining a likely candidate and finally making a decision if the object is in fact your keys. When we look at the sky we use our past experiences to determine if it is likely to rain. When you look at such tasks in detail they become quite complex and difficult to reproduce by an artificial agent such as a computer or a robot.

By examining such processes of detection, analysis and decision making we are able to automate repetitive and boring jobs and let machines do dangerous or dirty jobs. We also learn more about human learning and processes we do not even think about.

The above examples follow a pattern of detection, analysis and decision making. Detection is (among others) dealt with in the field of Computer Vision, where an image is examined and shapes and features are extracted to determine the presence or absence of objects. The analysis or feature extraction is a very broad subject in which sensors quantify features (such as color, size, texture, etc) of the object under scrutiny. Since a computer can only compute, anything which can be expressed as a number can function as a feature. The decision making is another wide field in which many variations and strategies exist. Deciders can vary from a systems with multiple different classifiers and a voting system to a single perceptron. In this article we will focus on a part of the decision making: classification. Classification is the act of identifying to which of a set of categories a new observation (pattern) belongs to.

Looking at a fruit and determining the kind of fruit is a typical example of classification. (Determining it is a fruit in the first place and not for example a rock is classification as well.) When we look at a fruit and decide it is an apple or an orange we use our memory to recall what an apple or an orange is supposed to look like: all the apples and oranges we have seen in

our lives of which we either have been told were apples and oranges (early in our lives) and all the apples and oranges we decided for ourselves were apples and oranges (later in our lives) have given us an abstract idea of what an apple and an orange is supposed to look like. Through past experiences we have in our minds prototypical examples of concepts such as apples and oranges or say tables.
Learning Vector Quantization (LVQ) uses this representation of a prototypical example. This algorithm was first proposed by T. Kohonen[8]. It represents a given dataset of labeled patterns by a number of pre-labeled prototypes and uses a heuristic scheme to position these prototypes at places where they can correctly represent their respective classes.
The process of classification leans heavily on the act of comparing patterns to each other and a measure of 'sameness' or difference is inescapable. In many classification processes this difference is measured by the 'distance' from pattern to pattern. The concept of 'distance' itself is quite suggestive as to how this is measured and many times the model of our three dimensional world (if needed extended to an n dimensional world) is used: a Euclidean distance measure. However this is not necessarily the best option. Even though our three dimensional world may work this way a data-space of arbitrary features has no link to physical space.
Instead of using an arbitrary metric such as a Euclidean metric we could let the metric depend on the data. Large Margin Nearest Neighbours (LMNN) takes such a metric learning approach. It uses the available patterns to determine a suitable metric. With this metric it uses k Nearest Neighbours[4] to classify new patterns.
Where LMNN needs the entire dataset to classify a new pattern, LVQ just needs the prototypes. LVQ also has a variant akin to metric learning: Matrix LVQ [11]. This variant has many similarities with LMNN in defining the distance measure. This makes the combination of both techniques an easy step and the combination only needs the prototypes and a metric for classification. This combination of LVQ and LMNN: Large Margin Learning Vector Quantization (LaMaLVQ) uses LVQ to position the prototypes and LMNN (the LaMa part) to compute a suitable metric for a given prototype configuration and a dataset.

Except for the **Introduction** this article is comprised of the sections **Related Work**, **Model**, **Methods**, **Experiments**, **Results**, **Discussion** and **Conclusion**. A short discussion of related articles will be discussed in the **Related Work** section. The problem statement, existing techniques and their adjustments are discussed in the **Model** section, followed by the **Methods** section in which the most important aspects of the implementation are discussed. The **Experiments** section explains the settings and ways experiments were conducted and the **Results** section shows the results and discusses some observations of said results. The observations are analyzed and and generalized more in the **Discussion** section which is followed by a **Conclusion** about the entire set of experiments.

Related Work

Pattern classification is a wide subject and many approaches exist, but many find their origin in Bayesian theory [9]. Parametric techniques try to model class densities with statistics and Bayesian parameter estimation. A famous example is the Fisher's Linear Discriminant [5] which tries to maximize between class variance while minimizing within class variance. Fisher achieved this by warping the data space with a weight vector. This can be extended to a weight matrix which warps the data space by letting each dimension contribute to each dimension of the warped space. Principal Component Analysis uses this to reduce dimensionality, projecting the data onto a lower dimensional space. Although this reduces dimensionality it does not necessarily improve classification results. Bar-Hillel et al. describe a related way to warp the data space and improve classification in *Learning a Mahalanobis Metric from Equivalence Constraints* [1]. In this article they present Relevant Component Analysis (RCA) which tries to amplify relevant components and decrease irrelevant noise components. These relevant and irrelevant components are identified by examining the variability of both data which is related and unrelated to each other(either through earlier class assignment or other prior available information). With this information a transformation matrix is computed to warp the data, making it easier to classify with another tool. This transformation matrix can also be called a metric since it can be used to determine distances between data points. It shows how close related component analysis and discriminants are to metric learning. Metric learning is a relatively new subject in machine learning whose goal is to find a metric to better separate classes and simplify classification. One of the combined techniques: LMNN, is another approach to metric learning. Also note that these methods originate from statistics and because the metric construction in metric learning is similar to the construction of a Mahalanobis distance in some articles these metrics are called Mahalanobis metrics or Mahalanobis distances.

Many metric learning algorithms try to find an optimal metric for the problem at hand. Ideally the problem should contain no locally optimal solutions: ideally the problem should be convex. Therefore metric learning problems often use tools from convex optimization. The subject of Semi Definite Programming[10] is particularly useful since metrics must (among others) produce non-negative distances.

Chechik et al. show an application of metric learning in *Large Scale Online Learning of Image Similarity Through Ranking*[3]. In this article they present the *OASIS* algorithm: *Online Algorithm for Scalable Image Similarity* which applies metric learning to image similarity search. The large scale of information is managed by only considering three samples at a time. A pairwise similarity is defined and metric updates are based on the relative similarity of the three images. After training OASIS for three days 35% of the ten nearest neighbours of a test image were found to be semantically relevant.

Metric learning can also be applied to the unsupervised problem of clustering. In *Distance Metric Learning, with Application to Clustering with Side-Information* Xing et al.[14] describe a technique to find a suitable met-

ric with limited information. Given a set of tuples which are previously determined to be similar and an optional set of tuples which are previously determined to be dissimilar a metric is constructed which pulls patterns which are similar together and pushes patterns which are dissimilar apart. When no dissimilarities are given other criteria are used to prevent the algorithm from finding the trivial solution $A = 0$ and let the metric collapse the dataset to one point.

LVQ was first proposed by Kohonen in *The self-organizing map*[8]. Later Sato and Yamada found the learning rule in Kohonen's LVQ does not satisfy convergence conditions. In *Generalized Learning Vector Quantization*[15] they formulated a cost function which together with the prototype updates does satisfy convergence conditions. The Generalized cost function calculates the distance to the closest correct and closest incorrect prototype and takes the ratio of the difference and the squared sum. Minimizing the generalized cost function has been proven to converge, but is still a heuristic.

A more principled approach is presented in *Soft Learning Vector Quantization*[12] in which a cost function based on statistics is proposed. The cost function in this case is a likelihood ratio of probability densities: densities denoting the chance a data point is generated by a mixture model of the correct class and densities denoting the chance a data point is generated by an incorrect class. The prototypes are taken as means of multivariate Gaussians and updates are based on maximizing the likelihood ratio in the cost function.

In *Generalized Relevance Learning Vector Quantization* Hammer and Villmann[7] present a LVQ based algorithm which incorporates learning the importance of input dimensions: Relevance Learning. In the distance calculation the input dimensions are given varying importance: weights. As the prototypes are positioned through minimizing the generalized cost function the weights are updated based on the same cost function. This idea was later expanded to Matrix LVQ in *Adaptive relevance matrices in Learning Vector Quantization* by Schneider, Biehl & Hammer [11] . Matrix LVQ uses a full matrix to weight input dimensions in distance calculations. Matrix elements are also updated according to the same cost function which positions the prototypes. Other than in Relevance LVQ (RLVQ) Matrix LVQ (MLVQ) also takes correlations between dimensions into account.

The use and construction of the matrix in MLVQ is very similar to the way matrices are used in the field of metric learning: $\mathbf{\Omega}^\top \mathbf{\Omega} = \mathbf{\Lambda}$. Although MLVQ took a different route to develop it is akin to metric learning. As metric learning is akin to dimensionality reduction so is MLVQ. In *Limited Rank Matrix Learning, discriminative dimension reduction and visualization*[2] Bunte et al. describes a LVQ based algorithm which incorporates a relevance matrix of limited rank. The matrix is again updated according to the same cost function as the prototypes, but here $\mathbf{\Omega}$ is rectangular, resulting in a matrix of limited rank. When the algorithm finishes, $\mathbf{\Omega}$ which is part of the learned metric can be used to project the dataset onto a lower dimensional space.

Model

Here we will discuss the problem statement, existing techniques and their adjustments to fit together to LaMaLVQ.

Problem

Given a set $\mathcal{S}$ of n-dimensional labeled patterns and a set of class labels $\mathcal{C}$, with $\{\boldsymbol{\xi}, \sigma\} \in \mathcal{S}, \sigma \in \mathcal{C}, \boldsymbol{\xi} \in \mathbb{R}^n$. Classify a new unknown pattern as a known class.
Classification is based on similarity and dissimilarity to previous available training data. A measure of similarity/dissimilarity is needed to compare a novel pattern to the training set. The numerical nature of $\mathcal{S}$, being comprised of labeled feature vectors, lends itself for a Distance-Based approach: a *distance* measure $\mathcal{D}(\mathbf{x}, \mathbf{y})$ measuring the *distance* from one point $\mathbf{x}$ to another point $\mathbf{y}$ in *feature space*.
A much used distance measure is the Euclidean distance we know from our own three dimensional world, however the feature space is not necessarily Euclidean. Therefore we need a distance measure which can change to suit the dataset. This distance measure, or metric, needs to satisfy a few properties to be a real metric:

1. $\mathcal{D}(\mathbf{x}, \mathbf{y}) + \mathcal{D}(\mathbf{y}, \mathbf{z} \geq \mathcal{D}(\mathbf{y}, \mathbf{z}) \ \forall \mathbf{x}, \mathbf{y}, \mathbf{z}$ (Triangular inequality)
2. $\mathcal{D}(\mathbf{x}, \mathbf{y}) \geq 0 \ \forall \mathbf{x}, \mathbf{y}$ (Non-negativity)
3. $\mathcal{D}(\mathbf{x}, \mathbf{y}) = \mathcal{D}(\mathbf{y}, \mathbf{x})$ (Symmetry)
4. $\mathcal{D}(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$ (Uniqueness)

A metric which satisfies, *Triangular inequality, Non-negativity* and *Symmetry*, but not *Uniqueness* is called a pseudo metric.

If we choose a metric such as $\mathcal{D}_{\boldsymbol{\Omega}}$ from (1) we obtain a family of metrics parameterized by the matrix $\boldsymbol{\Omega}$. When $\boldsymbol{\Omega}$ is of full rank $\mathcal{D}_{\boldsymbol{\Omega}}$ is a metric and a pseudo metric otherwise, which can be shown by a few fairly simple proofs. By defining $\boldsymbol{\Lambda} = \boldsymbol{\Omega}^\top \boldsymbol{\Omega}$ we obtain $\mathcal{D}_{\boldsymbol{\Lambda}}$, which is the same form of metric used in GMLVQ [11]. A real valued matrix of this form is guaranteed to be positive semi-definite [13] and it shows us the relevance of each dimensions on the diagonal and the correlations of dimensions in the off diagonal elements.

$$\mathcal{D}_{\boldsymbol{\Omega}}(\mathbf{x}, \mathbf{y}) = ||\boldsymbol{\Omega}(\mathbf{x} - \mathbf{y})||_2^2 = (\mathbf{x} - \mathbf{y})^\top \boldsymbol{\Lambda}(\mathbf{x} - \mathbf{y}) = \mathcal{D}_{\boldsymbol{\Lambda}}(\mathbf{x}, \mathbf{y}) \quad \text{where } \boldsymbol{\Lambda} = \boldsymbol{\Omega}^\top \boldsymbol{\Omega} \tag{1}$$

LMNN

Large Margin Nearest Neighbour (or LMNN for short) is a variation on k-Nearest Neighbours developed by Weinberg and Saul[13], which incorporates metric learning. It strives to find a metric which will better classify new patterns than an arbitrarily chosen and fixed distance measure (such

as the frequently used Euclidean distance). This technique uses a Euclidean distance measure, but parameterizes it with a matrix $\mathbf{\Lambda}$, obtaining a family of metrics as in (1). To improve kNN classification a metric is constructed which pushes patterns of different classes apart and pulls patterns of the same class together. LMNN minimizes the cost function (2) to achieve this.

$$\epsilon(\Lambda) = (1-\mu) \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_i) + \mu \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \sum_l (1-y_{il})[1 + \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_i) - \mathcal{D}(\boldsymbol{\xi}_l, \boldsymbol{\xi}_i)]_+ \tag{2}$$

$$\text{where } [z]_+ = \max(z, 0)$$

$$\text{where } y_{il} = \begin{cases} 1 & \text{if } \sigma_i = \sigma_l \\ 0 & \text{if } \sigma_i \neq \sigma_l \end{cases}$$

To minimize 2 we need to understand the concepts: *Target Neighbours* and *Invaders*, which are demonstrated in Figure 1 and explained below.

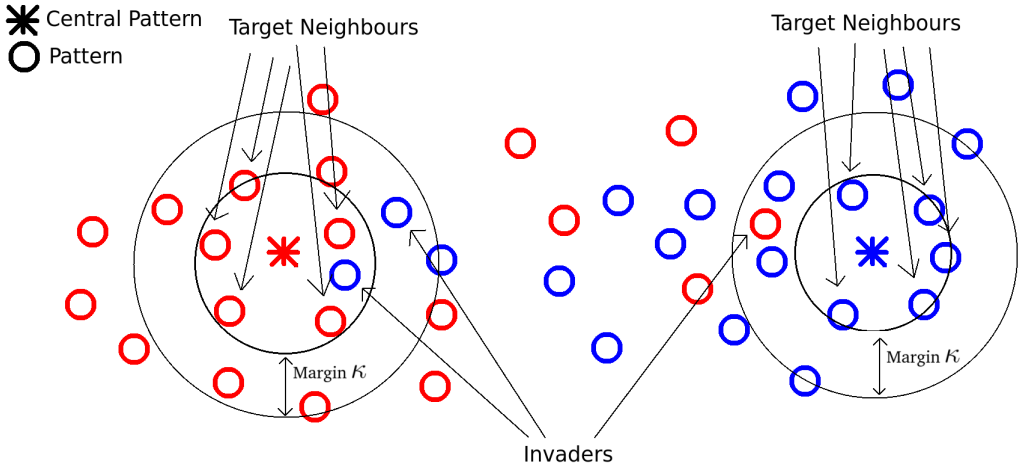


Figure 1: An example dataset with two classes. Two patterns of different class are taken as example central patterns.

Just as kNN, LMNN is parameterized by k the number of target neighbours. In kNN k determines the number of closest patterns on which the classification of a new pattern is based. In LMNN k determines the number of closest patterns of the same class, which are used to determine the perimeter in which patterns of a different class should not invade. The *Target Neighbours* as defined in (3):

$$\Phi_k^{\boldsymbol{\xi}_p} = \{\boldsymbol{\xi}_i | \forall i, j \in \mathbb{N}, i < j \wedge \boldsymbol{\xi}_i, \boldsymbol{\xi}_j \in \mathcal{S} \wedge \sigma_i = \sigma_j = \sigma_p \wedge \mathcal{D}(\boldsymbol{\xi}_i, \boldsymbol{\xi}_p) \leq \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_p) \wedge i \leq k\} \tag{3}$$

Patterns are attracted to each other and repelled from one another based on a perimeter around each pattern: an area which should not be invaded by patterns of a different class and in which all same class patterns should clump together as close as possible. This perimeter is defined by a number (k) of target neighbours and a margin κ (and of course the metric). For each pattern the k closest patterns of the same class are identified. Those are the *target neighbours* for that pattern. The largest distance d between the pattern and its target neighbours is identified. Then, the perimeter of a pattern is defined as the area around it which has distance to it closer than $d + \kappa$, as defined in (4):

$$\begin{aligned} P(\boldsymbol{\xi}_i, \kappa) &= \{\mathbf{x} | \mathcal{D}(\mathbf{x}, \boldsymbol{\xi}_i) \leq d + \kappa\} \\ \text{where } d &= \max_{\boldsymbol{\xi}_j} (\mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_i)), \boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i} \end{aligned} \tag{4}$$

For ease of discussion we will call the pattern which perimeter we are talking about the central pattern. We will call all patterns of a different label than the central pattern which invade the perimeter invaders, defined by (5):

$$\Psi_k^{\boldsymbol{\xi}_p} = \{\boldsymbol{\xi}_l | \exists \boldsymbol{\xi}_j, \boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_p} : \mathcal{D}(\boldsymbol{\xi}_l, \boldsymbol{\xi}_p) \leq \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_p) + \kappa\} \tag{5}$$

This cost function (2) consists of two parts:

$$(1 - \mu) \sum_{i, \boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_i)$$

which penalizes long distances between a central pattern and its target neighbours and

$$\mu \sum_{i, \boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \sum_l (1 - y_{il}) [1 + \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\xi}_i) - \mathcal{D}(\boldsymbol{\xi}_l, \boldsymbol{\xi}_i)]_+$$

which penalizes invaders. The parameter μ determines in what proportion the target neighbours and invaders affect the cost.

The problem modelled by equation (2) has been proven to be convex [13] and can be solved using a semidefinite program(SDP). A semidefinite program is a linear program with the added constraint of semi-definiteness on a symmetric matrix. The general form of a SDP is given in (6), where $\mathbf{X} \succeq 0$ denotes semi-definiteness of $\mathbf{X}$ and $\mathbf{C}, \mathbf{A}_1, ..., \mathbf{A}_n$ are symmetrical matrices.

$$\begin{aligned} &\text{minimize } \mathbf{tr}(\mathbf{CX}) \\ &\text{subject to } \mathbf{tr}(\mathbf{A}_k \mathbf{X}) = b_k \\ &\mathbf{X} \succeq 0 \end{aligned} \tag{6}$$

Recall that $\mathbf{x} \cdot \mathbf{x}^\top = [x_i x_j]_{ij}$ is a symmetrical matrix and if $\mathbf{C} = \mathbf{x} \cdot \mathbf{x}^\top$ then

$$\begin{aligned}
\mathbf{tr}(\mathbf{CX}) &= \\
&= \mathbf{tr}\Big(\Big[\sum_k \mathbf{C}_{ik}\mathbf{X}_{kj}\Big]_{ij}\Big) \\
&= \sum_{kl} \mathbf{C}_{lk}\mathbf{X}_{kl} \\
&= \sum_{kl} x_k x_l \mathbf{X}_{kl} \\
&= \mathbf{x}^\top \mathbf{X} \mathbf{x}
\end{aligned}$$

In order to apply this to our cost function (2) we define
$\mathbf{C} = \sum_{i,\Phi_k^{\boldsymbol{\xi}_i}} (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)(\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)^\top$,
$\mathbf{A}_k = \sum_{i,j \in \Phi_k^{\boldsymbol{\xi}_i}, l} ((\boldsymbol{\xi}_i - \boldsymbol{\xi}_l)(\boldsymbol{\xi}_i - \boldsymbol{\xi}_l)^\top) - ((\boldsymbol{\xi}_i - \boldsymbol{\xi}_j)(\boldsymbol{\xi}_i - \boldsymbol{\xi}_j)^\top)$
and all $b_k = 1$.
Now we introduce nonnegative slack variables ζ_{ijl} for each triple ijl to measure the loss and we obtain (7). We note that $(\mathbf{x}\ \mathbf{x}^\top - \mathbf{y}\ \mathbf{y}^\top)\mathbf{X} = \mathbf{x}^\top \mathbf{X}\mathbf{x} - \mathbf{y}^\top \mathbf{X}\mathbf{y}$, substitute $\boldsymbol{\Lambda}$ for $\mathbf{X}$ and introduce variable μ and we end up with the semidefinite program in Table 1.

$$\begin{aligned}
\text{minimize } & \Big(\sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)(\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)^\top \Big) \bullet \mathbf{X} + \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}, l} (1 - y_{il})\zeta_{ijl} \\
\text{subject to } & \Big(\sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}, l} ((\boldsymbol{\xi}_l - \boldsymbol{\xi}_i)(\boldsymbol{\xi}_l - \boldsymbol{\xi}_i)^\top) - ((\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)(\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)^\top), \mathbf{X} \Big) \geq 1 - \zeta_{ijl} \\
& \zeta_{ijl} \geq 0 \\
& \mathbf{X} \succeq 0
\end{aligned} \tag{7}$$

Minimize $(1-\mu)\sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i) + \mu \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}, l} (1 - y_{il})\zeta_{ijl}$
Subject to:
(1) $(\boldsymbol{\xi}_l - \boldsymbol{\xi}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_l - \boldsymbol{\xi}_i) - (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_j - \boldsymbol{\xi}_i) \geq 1 - \zeta_{ijl}$
(2) $\zeta_{ijl} \geq 0$
(3) $\boldsymbol{\Lambda} \succeq 0$

Table 1: Semidefinite program for LMNN

With the SDP in table 1 we can find an optimal solution for the cost function of (2) and find a metric which pushes different class patterns apart and pulls same class patterns together.

LVQ

Learning Vector Quantization (or LVQ for short) is a prototypes based supervised classification technique [11]. It works by representing a dataset by a number of prototypical instances: the prototypes. These prototypes are then used to classify new patterns much in the same as way kNN does, however the winner takes all principle is most frequently used: only the closest prototype determines the classification.
Positioning the prototypes in a prototypical position is the most important (and time consuming) aspect of LVQ. Prototypes are initially put somewhere in the data space. Then patterns from the dataset are randomly presented to the prototypes which triggers an update of one or more prototypes. These updates can be simple heuristics, but are more often gradient descent based updates which try to optimize a cost function. When a cost function is used the training of the prototypes is a stochastic gradient descent method.
Cost functions take the form of (8) where Θ is a monotonic function and u_i the cost of the pattern with subscript i.

$$\sum_i \Theta(u_i) \tag{8}$$

A much used cost function is the generalized LVQ cost function (9).

$$\sum_i \frac{\mathcal{D}_J(\boldsymbol{\xi}_i) - \mathcal{D}_K(\boldsymbol{\xi}_i)}{\mathcal{D}_J(\boldsymbol{\xi}_i) + \mathcal{D}_K(\boldsymbol{\xi}_i)} \tag{9}$$

Here Θ is the identity function $\Theta(\mathbf{x}) = \mathbf{x}$. Where $\mathcal{D}_J(\boldsymbol{\xi}_i)$ is the distance to the closest (correct) prototype with the same class label as $\boldsymbol{\xi}_i$ and $\mathcal{D}_K(\boldsymbol{\xi}_i)$ the distance to the closest (incorrect) prototype with a different class label than $\boldsymbol{\xi}_i$. $\mathcal{D}_J(\boldsymbol{\xi}_i)$ corresponds to the closest correct prototype $\boldsymbol{\omega}_J$ and $\mathcal{D}_K(\boldsymbol{\xi}_i)$ corresponds to the closest incorrect prototype $\boldsymbol{\omega}_K$.

If we take derivatives with respect to $\boldsymbol{\omega}_J$ and $\boldsymbol{\omega}_K$ we obtain the updates in (10) and (11):

$$\Delta\boldsymbol{\omega}_J = 2\eta \frac{\mathcal{D}_K(\boldsymbol{\xi}_i)}{(\mathcal{D}_J(\boldsymbol{\xi}_i) + \mathcal{D}_K(\boldsymbol{\xi}_i))^2} \boldsymbol{\Lambda}(\boldsymbol{\xi}_i - \boldsymbol{\omega}_J) \tag{10}$$

$$\Delta\boldsymbol{\omega}_K = -2\eta \frac{\mathcal{D}_J(\boldsymbol{\xi}_i)}{(\mathcal{D}_J(\boldsymbol{\xi}_i) + \mathcal{D}_K(\boldsymbol{\xi}_i))^2} \boldsymbol{\Lambda}(\boldsymbol{\xi}_i - \boldsymbol{\omega}_K) \tag{11}$$

where $0 < \eta < 1$ is the learning rate which determines the magnitude of each learning step.

The distance measure in LVQ can be can be parameterized by a matrix as in (1) as described by Schneider, Biehl and Hammer in [11]. When this matrix is also updated according to the cost function it is yet another form of metric learning. This similarity of distance measure with LMNN makes LVQ and LMNN very inviting to try and combine.

As a last note it has to be said that although a very nice technique LVQ also has its pitfalls. In general the positioning of the prototypes is a problem which is not convex. Local optima might exists and a gradient descent method can get stuck in one such local optimum. This can be mitigated by initializing the prototypes in positions which are likely close to the optimal position.

LaMaLVQ

Large Margin LVQ (or LaMaLVQ for short) is the combination of LMNN and LVQ. The two techniques work alternating, taking a LM step and a LVQ step each round:

1. LVQ positions the prototypes. This is done by minimizing a cost function used by the LVQ part which may or may not be different from the cost function used by the LaMa part. The prototypes are pushed and pulled towards (possibly local) optimal positions according to the dataset, the current metric and the LVQ cost function.
2. The LaMa part takes the prototypes and dataset and calculates the optimal metric for this prototype configuration. This is done by minimizing the LaMaLVQ cost function as explained in the **Solver** subsection in the **Methods** section.

We present here four variations of LaMaLVQ two of which determine how a metric is computed and two of which determine how often a metric is computed. We therefore put these variation in two categories:

1. LaMa mode: Single or Consecutive
2. LaMa version : Pattern or Prototype

When a metric is computed this is done based on the target neighbours, which are determined by the previous metric. When a new metric is computed the target neighbours may have changed and the new metric might no longer be optimal. Therefore a new metric can be computed with the new target neighbours. This continues until the metric does not change significantly or the maximum number of metrics have been computed. This LaMa mode is consecutive mode. If a metric is computed only once we refer to it as single mode.
A metric can be computed according to a cost function based on the prototypes and surrounding pattern, but it can also be computed by only taking the prototypes into consideration. These are the two different LaMa versions: Pattern version and Prototype version respectively. Details of these versions can be found in the following sections **Pattern version** and **Prototype version**.
A third variation is possible, though this is not exclusive to LaMaLVQ as it can also be applied to LVQ. Instead of using one matrix to define a metric for the whole dataset and all prototypes it is also possible to define a metric for each class or even for each prototype.

1. Global Metric: one metric for all patterns and prototypes

2. Class-specific Metric : a metric for each class: $\mathcal{D}_{\mathbf{\Lambda}_\sigma}$
3. Prototype-specific Metric : a metric for each prototype: $\mathcal{D}_{\mathbf{\Lambda}_i}$

Here we will only use a global metric or a metric for each class.
When using Class-specific Metrics each time a distance calculation is made one metric is selected to do the calculation with: the metric corresponding to the class of the prototype. There is only one case in which the distance between two prototypes is computed: during metric computation. During this computation the metric under scrutiny is used.

LMNN uses a unity margin $\kappa = 1$ because the margin does not affect the classification and only sets the scale for $\mathbf{\Lambda}$ [13]. However, using a larger or smaller margin does affect the number of invaders and as such the computation time. Therefore we use a margin based on the dataset. We determine the minimum and maximum of each dimension and we take a percentage of the Frobenius norm of the difference: (12).

$$\begin{aligned} \kappa &= p * ||\mathbf{ma} - \mathbf{mi}||_{frobenius}, 0 < p < 1 \qquad (12) \\ \text{where} \quad \mathbf{mi} &= (\min_i(\boldsymbol{\xi}_i)_1, ..., \min_i(\boldsymbol{\xi}_i)_n)^\top \\ \mathbf{ma} &= (\max_i(\boldsymbol{\xi}_i)_1, ..., \min_i(\boldsymbol{\xi}_i)_n)^\top \end{aligned}$$

In the following sections we will discuss how and if we have to change the LMNN and LVQ techniques to work together. We will show the two versions of LaMa which calculate a metric in a slightly different way. We also will show an additional way to position the prototypes in the LVQ step which is more in line with the LaMaLVQ cost function.

Pattern version

The LMNN technique constructs a metric by making same class patterns attract each other and different class patterns repel each other. The technique is completely based on the patterns in the dataset. LVQ tries to represent the dataset by prototypes and eliminate the need to retain the entire dataset for classification. Therefore we need a technique which incorporates the prototypes in the metric construction. Instead of making a metric in which patterns are attracted and repelled to and by other patterns we construct a metric which attracts patterns to same class prototypes and repels patterns from different class prototypes.
We alter the cost function in (2) slightly to (13):

$$\boldsymbol{\epsilon}(\Lambda) = (1-\mu) \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\omega}_i}} \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\omega}_i) + \mu \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\omega}_i}} \sum_l (1-y_{il})[1 + \mathcal{D}(\boldsymbol{\xi}_j, \boldsymbol{\omega}_i) - \mathcal{D}(\boldsymbol{\xi}_l, \boldsymbol{\omega}_i)]_+ \tag{13}$$

$$\text{where } [z]_+ = \max(z, 0)$$

$$\text{where } y_{il} = \begin{cases} 1 & \text{if } \sigma_i = \sigma_l \\ 0 & \text{if } \sigma_i \neq \sigma_l \end{cases}$$

in which the first pattern of each difference calculation is replaced by a prototype. Table 2 shows the SDP for the Pattern version in which $\boldsymbol{\Lambda} \succeq 0$ denotes $\boldsymbol{\Lambda}$ is a semidefinite matrix:

Minimize $\sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} (\boldsymbol{\xi}_j - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_j - \boldsymbol{\omega}_i) + \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i},l} (1-y_{il}) \zeta_{ijl}$
Subject to:
(1) $(\boldsymbol{\xi}_l - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_l - \boldsymbol{\omega}_i) - (\boldsymbol{\xi}_j - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\xi}_j - \boldsymbol{\omega}_i) \geq \kappa - \zeta_{ijl}$
(2) $\zeta_{ijl} \geq 0$
(3) $\boldsymbol{\Lambda} \succeq 0$

Table 2: Semidefinite program for LaMaLVQ pattern version

This is the *pattern* version of LaMaLVQ. Although it is based partially on the prototypes it is called the Pattern version as opposed to the Prototype version which is based solely on the prototypes.

Prototype version

Since classification is based solely on the prototypes we also include a version which constructs a metric based solely on the prototypes. This Prototype version constructs a metric in which same class prototypes are attracted and different class prototypes are repelled from one another.
We alter the LMNN cost function in (2) to (14):

$$\boldsymbol{\epsilon}(\Lambda) = (1-\mu) \sum_{i,\boldsymbol{\omega}_j \in \Phi_k^{\boldsymbol{\omega}_i}} \mathcal{D}(\boldsymbol{\omega}_j, \boldsymbol{\omega}_i) + \mu \sum_{i,\boldsymbol{\omega}_j \in \Phi_k^{\boldsymbol{\omega}_i}} \sum_l (1-y_{il})[1 + \mathcal{D}(\boldsymbol{\omega}_j, \boldsymbol{\omega}_i) - \mathcal{D}(\boldsymbol{\omega}_l, \boldsymbol{\omega}_i)]_+ \tag{14}$$

$$\text{where } [z]_+ = \max(z, 0)$$

$$\text{where } y_{il} = \begin{cases} 1 & \text{if } \sigma_i = \sigma_l \\ 0 & \text{if } \sigma_i \neq \sigma_l \end{cases}$$

in which all patterns are replaced by prototypes. Table 3 shows the resulting SDP:

Obviously all classes need to be represented by at least two prototypes if this technique is to be used. Also note that if we want to consider this version an approximation of the Datapoint version (and by proxy to LMNN) a large

Minimize $\sum_{i,\boldsymbol{\omega}_j \in \Phi_k^{\boldsymbol{\omega}_i}} (\boldsymbol{\omega}_j - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\omega}_j - \boldsymbol{\omega}_i) + \sum_{i,\boldsymbol{\omega}_j \in \Phi_k^{\boldsymbol{\omega}_i},l} (1 - y_{il}) \zeta_{ijl}$
Subject to:
(1) $(\boldsymbol{\omega}_l - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\omega}_l - \boldsymbol{\omega}_i) - (\boldsymbol{\omega}_j - \boldsymbol{\omega}_i)^\top \boldsymbol{\Lambda} (\boldsymbol{\omega}_j - \boldsymbol{\omega}_i) \geq \kappa - \zeta_{ijl}$
(2) $\zeta_{ijl} \geq 0$
(3) $\boldsymbol{\Lambda} \succeq 0$

Table 3: Semidefinite program for LaMaLVQ prototype version

number of prototypes need to be used. This in turn means the datasets need to be quite large lest the datasets will be overrepresented causing 'dead' prototypes. In this research the datasets have been kept relatively small to reduce computation time.

LaMa cost function based updates

The LVQ and LMNN technique are two different techniques and as such they (in general) optimize different cost functions to come to an optimal classification. It is possible to use both techniques mostly unchanged, but with each optimizing something different this might hamper the classification.
Therefore, additionally to the GLVQ update we define an update function based on the LaMa Pattern version cost function (13). For an online update we would need to recalculate the target neighbours and invaders at every step. If we let Pr be the number of prototypes and Pa be the number of patterns, this would increase the complexity of the LVQ step from linear: $Pr\ Pa$ to quadratic $(Pr\ Pa)Pa$. Though correct, this is highly impractical when the number of patterns or number of prototypes increases.
To keep computation down we define a batch update. This update simply identifies all target neighbours and invaders, calculates the total update and applies the update each epoch. The update per pattern is defined in (15).

$$\Delta\boldsymbol{\omega} = \begin{cases} \eta\ 2\boldsymbol{\Lambda}(\boldsymbol{\xi} - \boldsymbol{\omega}) & \text{if } \boldsymbol{\xi} \in \Phi_k^{\boldsymbol{\omega}} \\ -\eta\ 2\boldsymbol{\Lambda}(\boldsymbol{\xi} - \boldsymbol{\omega}) & \text{if } \boldsymbol{\xi} \in \Psi_k^{\boldsymbol{\omega}} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Methods

Here we will discuss the most important aspects of the implementation: the overall algorithm and the computation of the metric in the Solver.

Algorithm

The LaMaLVQ algorithm alternates between LVQ and LaMa steps, starting with LVQ. The LVQ step puts the prototypes in a (possibly local) optimal place according to a given cost function and update method and the LaMa step computes an optimal metric according to the given LaMa cost function.

All of the above are described in the **Model** section.

As mentioned before in the **Model** section we have a number of ways to use LaMaLVQ:

1. Single mode LaMaLVQ
2. Consecutive mode LaMaLVQ
3. Pattern version LaMaLVQ
4. Prototype version LaMaLVQ
5. Global Metric LaMaLVQ
6. Class-specific Metric LaMaLVQ

Of these variations Single and Consecutive mode are mutually exclusive as are Pattern and Prototype version and Global and Class-specific metric, but can otherwise be mixed and matched. Although these are the top level parameters, some more are needed to use the LaMaLVQ algorithm. We list the most important parameters. The LaMa parameters are:

1. **LaMa mode**: Single or Consecutive: determines a single or more passes over the dataset to determine the metric.
2. **LaMa version**: Pattern or Prototype: determines what data to consider when determining the metric.
3. **LaMa metric**: Global or Class :determines how many metrics are present and what data they encompass.
4. $\mathbf{p}_{\boldsymbol{\kappa}}$ (marginpercentage): determines (along with the target neighbours) the perimeter around the central pattern as defined in (12).
5. **k** : determines how many target neighbours will be used. If a more than the available target neighbours are specified, all available patterns are used as target neighbours.
6. **maxConsecRounds**: the maximum number of passes allowed in Consecutive mode
7. **rounds**: The number of times a LVQ and LaMa step are applied

Some parameters only affect the LVQ part:

1. **protNrMode** : the way in which the number of prototypes per class are specified: absolute (a number) or percentage (a percentage of the number of instances in the class)
2. **nrprototypes** : the number of prototypes per class if specified by an absolute number
3. **protperc** : determines the number of prototypes if a percentage is specified. The number of prototypes per class is a percentage of the average size off all present classes
4. **learningrate** : the fraction of an update used to actually update a prototype

5. **epochs** : the number of passes through the dataset each LVQ step
6. **updatemode** : the manner in which prototypes are positioned during the process
7. **initmode** : determines the initial positioning of the prototypes

Algorithm 1 gives the overall outline of the LaMaLVQ algorithm.

```
Data: Dataset, LaMaparameters
Result: PrototypeConfiguration, Metrics
numberOfRounds = LaMaparameters.rounds;
for numberOfMetrics do
  | Metrics[i] = I;
end
initializePrototypes(LaMaparameters.initmode);
for numberOfRounds do
  | LVQ(LaMaparameters, Metrics);
  | for numberOfMetrics do
  |   | solve(LaMaparameters, Metrics[i]);
  | end
end
```

Algorithm 1: LaMaLVQ algorithm

Solver

The solver is the part of the algorithm which solves the Semi Definite Program. It computes the optimal metric for the given dataset. The problem has been proven to be convex [13] and as such a gradient descent method is a viable approach. For a complete description refer to the appendix of "Distance Metric Learning for Large Margin Nearest Neighbor Classification"[13]. Here we will give a quick overview.
First we note that the cost function (13) can be rewritten as (16):

$$\boldsymbol{\epsilon}(\Lambda) = \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \mathbf{tr}(\Lambda \mathbf{C}_{ij}) + \sum_{\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}, l} (1 - y_{il})[\kappa + \mathbf{tr}(\Lambda \mathbf{C}_{ij}) - \mathbf{tr}(\Lambda \mathbf{C}_{il})]_+ \tag{16}$$

where $\mathbf{C}_{ij} = (\boldsymbol{\omega}_i - \boldsymbol{\xi}_j)(\boldsymbol{\omega}_i - \boldsymbol{\xi}_j)^\top$
and $[z]_+ = \max(z, 0)$

From this form we can easily obtain the gradient (17):

$$\mathbf{G} = \sum_{i,\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}} \mathbf{C}_{ij} + \sum_{\boldsymbol{\xi}_j \in \Phi_k^{\boldsymbol{\xi}_i}, l} (1 - y_{il})(\mathbf{C}_{ij} - \mathbf{C}_{il}) \tag{17}$$

If we now apply a gradient descent method we can obtain the optimal metric. However, finding all target neighbours and invaders at each step is very costly and highly impractical. We note that the first part of the gradient (with only target neighbours) (17) does not change during the process.

This leaves us with finding only the triples of prototype, target neighbours, invader. This is however the most costly step and we need another observation to get a practically obtainable solution. Therefore we note that most of the dataset will not lie within the perimeter for most of the time and calculating distances to data points which are not likely to be inside the perimeter is a waste of time. So the gradient is only computed using a set of active data points, which is periodically computed exactly by considering all data points. Data points enter the active set if they have been found an invader in an exact computation. True invaders are determined by considering only the active set and the gradient is updated by data points which leave or enter the invader set.

Equations (16) and (17) show the rewritten cost function and gradient for the Pattern version of LaMaLVQ. The Prototype version can be obtained by substituting the data points in $\mathbf{C}_{ij}$ for prototypes.

Experiments

All experiments are setup using a set of parameters. We list the most important below:

1. **metricType**: Global or Class-specific metric
2. **marginPercentage**: Determines the margin used (p in (12))
3. **LaMaVersion**: Data point, Prototype or Pure LVQ which does not calculate new metrics
4. **LaMaMode**: Single of Consecutive
5. **invaderInterval**: Determines how many iterations pass in the solver before the invaders are precisely computed.
6. **protNrMode**: Absolute or percentage.
7. **protperc**: Determines the number of prototypes per class as a percentage of the average size of all present classes.
8. **nrprots**: Determines the number of prototypes per class.
9. **learningrate**: Determines the rate of adaptation in an LVQ update as in (10), (11) and (15).
10. **epochs**: Determines the number of sweeps through the dataset in the LVQ step.
11. **updatemode**: Determines the way the prototypes are updates in the LVQ step.
12. **matrixUpdatemode**: Determines the way the metric is updated in the LVQ step.
13. **initmode**: Determines how the prototypes are initialized, "average class initialization" positions the prototypes at the average of the class they represent.
14. **normalization**: Determines how the dataset is normalized.

15. **costfunction**: Determines what cost function is used.
16. **rounds**: Determines the the number of times a LVQ step and LaMa step are performed.
17. **folds**: Determines the number of divisions used for cross fold validation
18. **datasetReduction**: Determines how much of the dataset is used in the actual experiment. The percentage is applied per class, so all classes present in the original dataset are present in the reduced dataset.

Some parameters barely change if at all from experiment to experiment. These parameters along with their value are listed in Table 4. If not noted otherwise these parameters have this default value.

marginPercentage	0.01
invaderInterval	10
learningrate	0.01
epochs	20
initmode	average class initialization
matrixUpdatemode	Do Nothing
normalization	znorm
rounds	30
folds	10
datasetReduction	1.0 for Iris and Wine, 0.025 for Letter

Table 4: Universal test settings

Datasets

In this paper we use several datasets all of which were taken from the UCI Machine Learning Repository [6]. The characteristics of these datasets are shown in Table 5.

Dataset	number of patterns	number of classes	number of dimensions
Iris	150	3	4
Wine	178	3	13
Letter	20000	26	16

Table 5: Data set characteristics

The Letter dataset is very large and was therefore not useful in experimentations as a whole. For this reason only a portion (which was newly selected each experiment) of this dataset was used in experimentation.

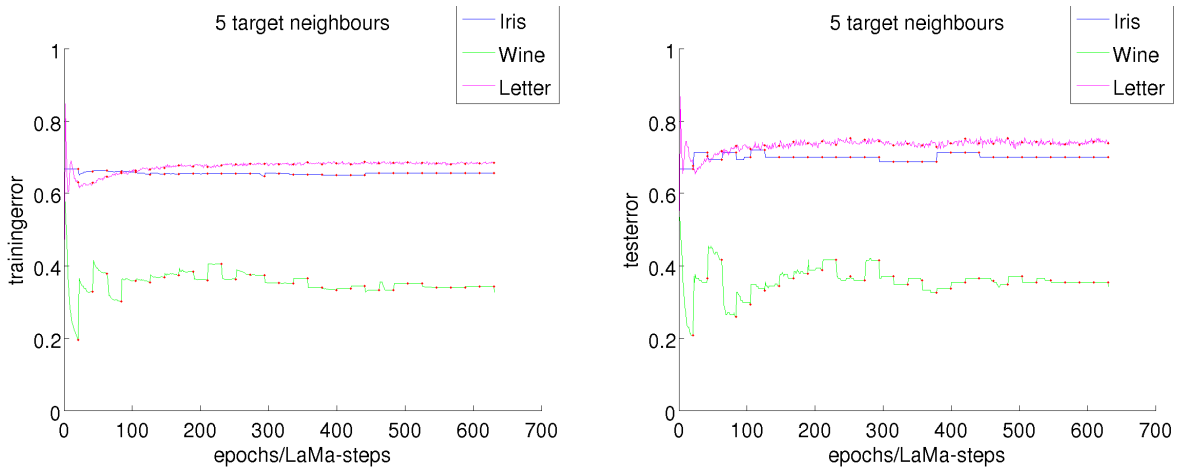
Results

Here we show the results of the experiments. The values of the cost progress have been normalized to values between 0 and 1. The red dots signify the points just before a LaMa step has been performed and a new metric has been computed.

Datapoint version with batch update

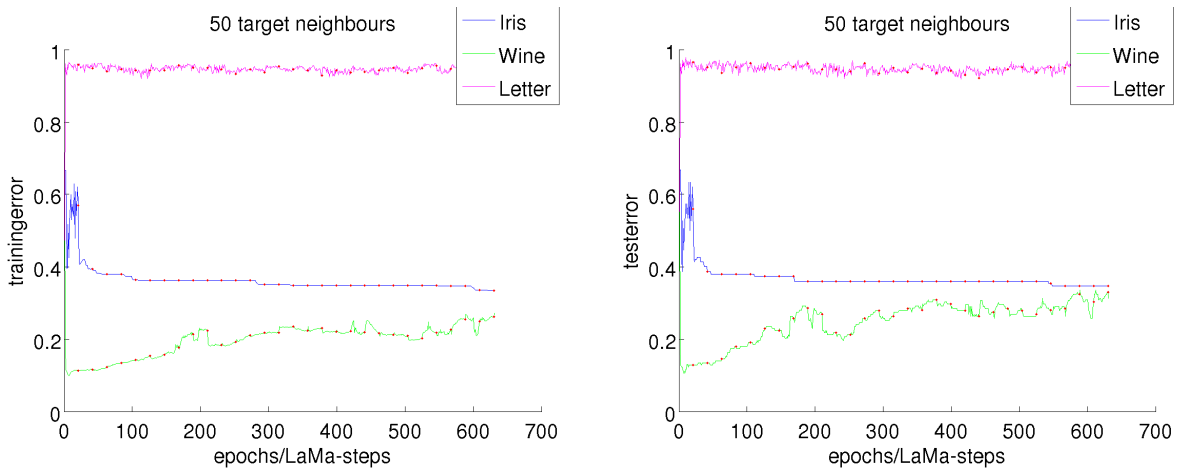
metricType	single metric
LaMaVersion	Datapoint version
LaMaMode	single LaMa
protNrMode	absolute
nrprots	1
updatemode	LaMa batch update
costfunction	LaMaLVQ Datapoint Version

Table 6: Datapoint batch update test settings



(a) training error

(b) test error

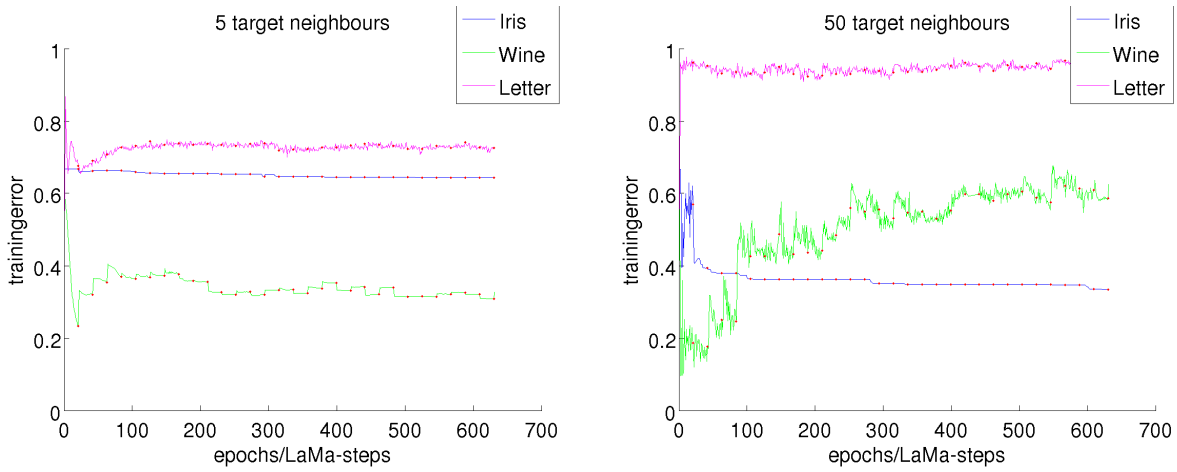


(c) training error

(d) test error

Figure 2: Batch update with Datapoint version: training and test error for 5 and 50 target neighbours

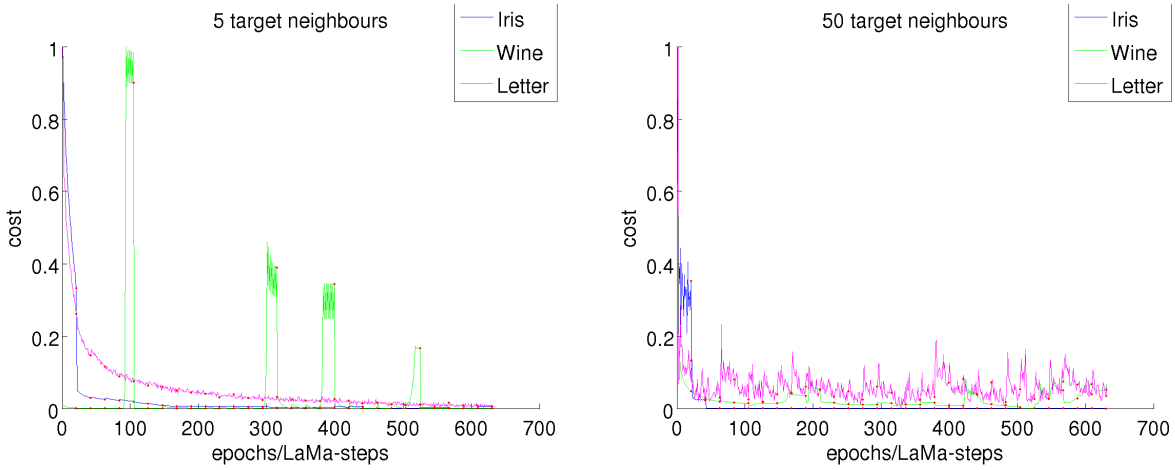
Figure 2 show the results of LaMaLVQ using the Datapoint version and a LaMa batch update. The exact settings are displayed in Table 6. The testerror mirrors the training error with the training error being slightly more smooth Therefore we will only show the training error in further experiments. The Letter dataset classification invariably worsens with the increase of target neighbours. The Wine dataset starts out improving in classification, but over the course of more rounds worsens again. The Iris dataset classification does not change much. Though when all available data points are used as target neighbours (when the number of target neighbours reaches 50) an improvement does show. This experiment shows some improvement in classification with the simpler datasets, though the Letter dataset shows no improvement at all.
Some learning curves in this experiment show a sudden increase or decrease in classification. This is most clear with the Wine dataset. This can be explained by the use of LaMa steps. When a new metric is computed, this can drastically change the data space and classification. Moreover different target neighbours might be found after a LaMa step than which were used to compute the new metric. This behaviour is also present in the progress of the cost function.



(a) training error (b) test error

Figure 3: Batch update with Datapoint version using consecutive LaMa: training error for 5 and 50 target neighbours

Figure 3 show the results when using the consecutive LaMa version along with the Datapoint a LaMa batch update. These results are very similar to that of single LaMa. In fact little difference exists between the results of these two versions and in further experiments only the single LaMa version will be shown.



(a) 5 target neighbours

(b) 50 target neighbours

Figure 4: Batch update with Datapoint LaMa version: cost function progress for 5 and 50 target neighbours

Figure 4 shows the cost progress of LaMaLVQ using the Datapoint version and a LaMa batch update. Sometimes a LaMa step is accompanied by a sharp increase or decrease in cost. This usually indicates a change in target neighbours. Also note that a LaMa step does not in general decrease the cost even though it is designed to. As soon as the new metric is computed the target neighbours might change and completely change the cost again. Another observation is that a smaller number of target neighbours constitutes (in general) a more stable cost progress. The metric does not change as much with few target neighbours as with many.

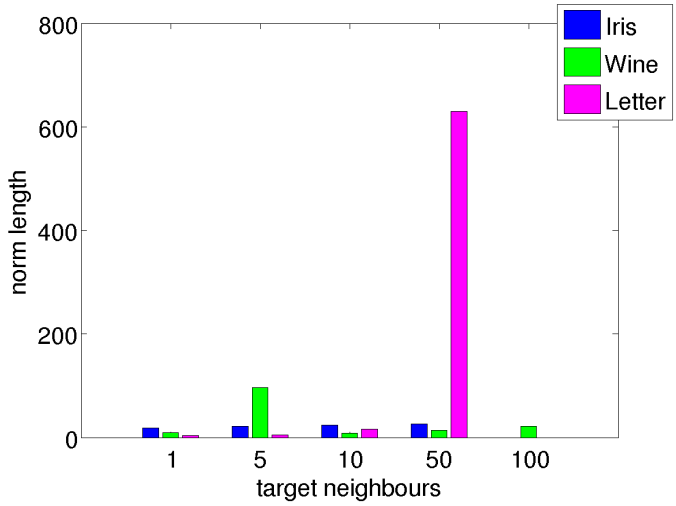
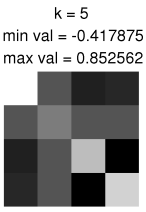
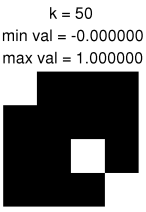


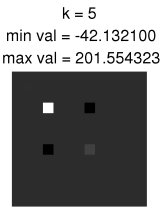
Figure 5: Batch update with Datapoint LaMa version: average prototype norms for different numbers of target neighbours



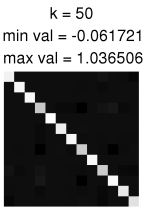
(a) Iris



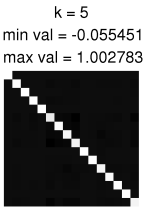
(b) Iris



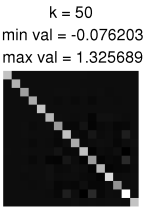
(c) Wine



(d) Wine



(e) Letter



(f) Letter

Figure 6: Average of resulting matrices of LaMa LVQ with LaMa batch update, no normalization or zeroing of diagonal

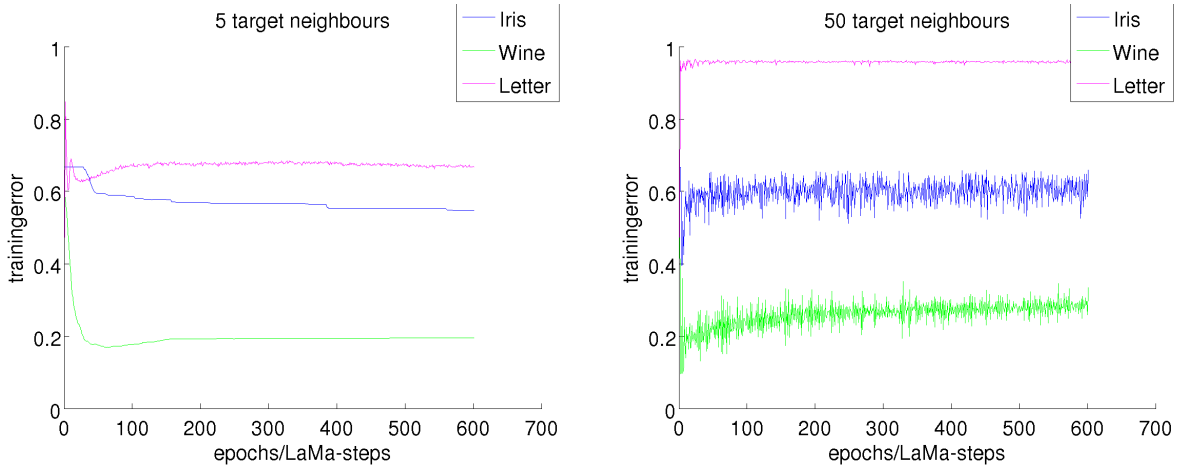
A conspicuous aspect of Figure 2 is the increase of the Letter training error going up to nearly 100%. When we look at the average norms of the

prototypes in Figure 5 we immediately see the norm for the Letter dataset using 50 target neighbours is extremely high. Since all datasets are normalized a large average norm could mean the prototypes are positioned far away from dataset. This is still dependent on the form of the metric, though. A metric with large magnitude entries can cause large magnitude updates and position prototypes far away from the dataset as well. The data space may in fact be so warped that prototypes with a large magnitude are actually close to the class they represent. The result of a training error of near a 100% tells us otherwise, though and looking at the resulting metrics in Figure 6 the scale of the corresponding metric is not exceptionally large, indicating the prototypes have moved away from the dataset. Another striking feature is the the average norm of the Wine dataset using five target neighbours. Although not as large as the Letter norm of 50 target neighbours, it is exceptionally high. But as we look at the training error we see nothing out of the ordinary. When we look at the resulting metric in Figure 6 we find the cause: the scale of the resulting metric has become very high between -42 and 201. With such a metric calculated distances can get very high and corresponding updates equally high resulting in large steps. Even though the prototypes have been pushed away the classification has not suffered that much, so a large prototype norm does not necessarily mean a bad classification when the metric is to blame. In fact, in the warped space the metric defines the prototypes could be very close to the dataset.

Pure LVQ with batch update

metricType	single metric
LaMaVersion	Pure LVQ
LaMaMode	N/A
protNrMode	absolute
nrprots	1
updatemode	LaMa batch update
costfunction	LaMaLVQ Datapoint version

Table 7: Pure LVQ batch update test settings

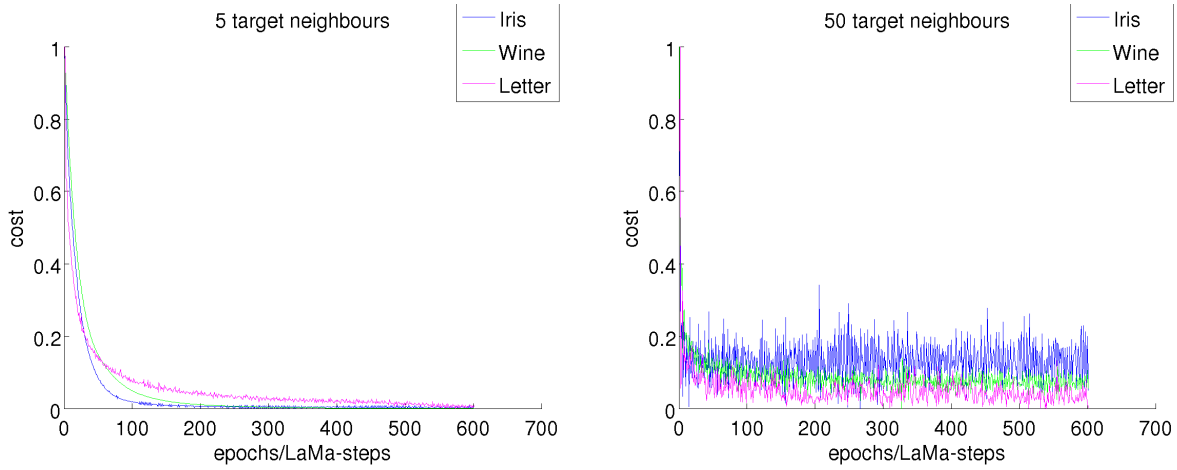


(a) 5 target neighbours

(b) 50 target neighbours

Figure 7: Batch update without LaMa update: training error for 5 and 50 target neighbours

Figure 7 shows the training error when only LVQ is applied using the batch update based on the LaMa cost function. This update scheme seems well suited for the wine dataset. With a larger number of target neighbours the training error goes down, though it does become more erratic as well. The Iris training set produces a worse result. A larger number of target neighbours reduces the training error only slightly. The Letter dataset produces different results again, producing a worse training error as the number of target neighbours increases.



(a) 5 target neighbours

(b) 50 target neighbours

Figure 8: Batch update without LaMa update: cost progress for 5 and 50 target neighbours

The cost progress in Figure 8 behaves as expected save for a more erratic progress as the number of target neighbours grows. Without the metric updates there are no sudden jumps in cost. The batch nature of the update can cause a sudden shift in target neighbours causing a quick increase or decrease in cost. The more target neighbours are used the more pronounced these shifts are. Still the cost progress in general shows a steady decrease. Combined with a disappointing training error this indicates the LaMa cost function does not work well with all datasets.

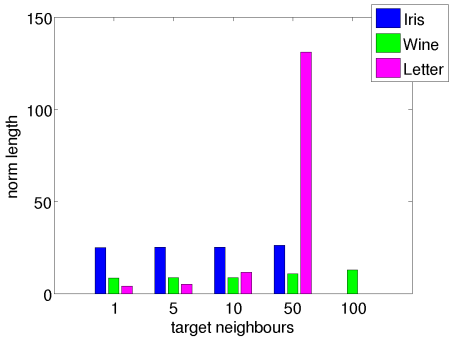


Figure 9: Batch update without LaMa update average prototype norms for different numbers of target neighbours

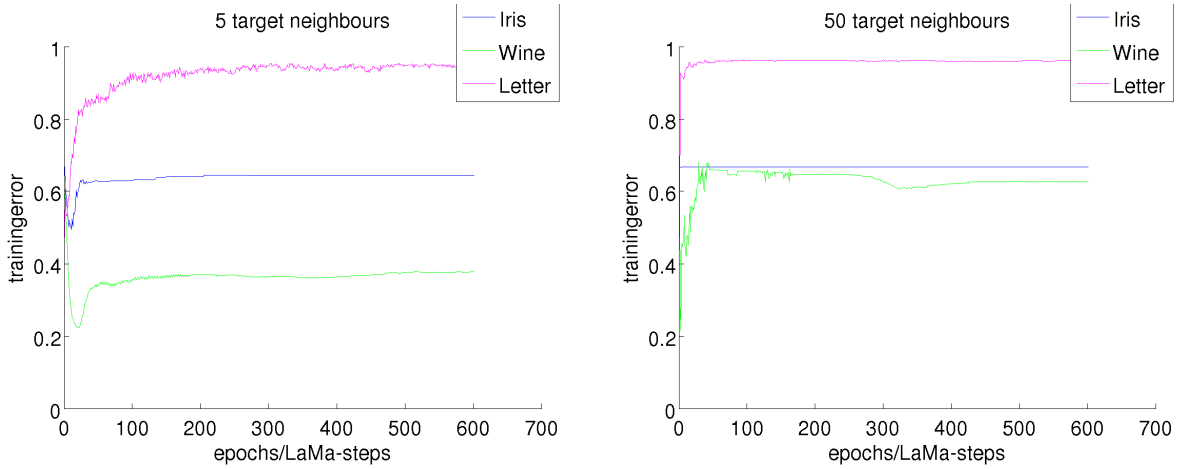
The average norm of the prototypes (shown in Figure 9) show a relatively large norm for the Iris dataset. Since no metric updates are performed

in this experiment this denotes the prototypes are positioned far away from the Iris dataset and more so than the prototypes from the Letter and Wine datasets are. The Letter norm using 50 target neighbours on the other hand is extremely large, the prototypes have been pushed very far away. This is reflected in the training error which deteriorates to 100%.

Pure LVQ version with batch update for both prototypes and matrix

metricType	single metric
LaMaVersion	Pure LVQ
LaMaMode	N/A
protNrMode	absolute
nrprots	1
updatemode	LaMa batch update
matrixUpdatemode	LaMa batch update
costfunction	LaMaLVQ Datapoint Version

Table 8: Pure LVQ batch update with metric LaMa batch update test settings

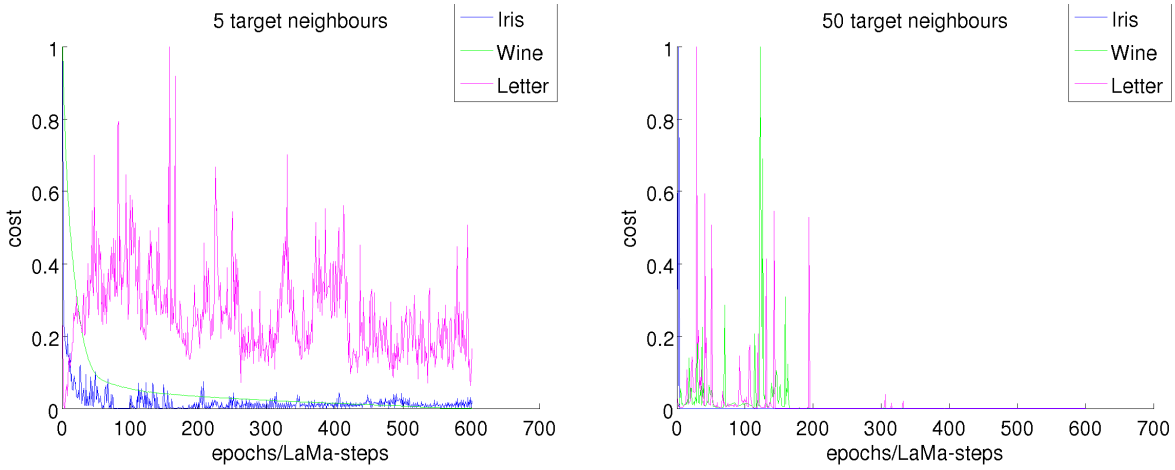


(a) 5 target neighbours

(b) 50 target neighbours

Figure 10: Batch update for both prototype and metric without LaMa update: train error for 1 and 50 target neighbours

If a constant drastic update of the metric does not give us such good results, but the absence of metric updates is no better, maybe we can find a middle ground. Figure 10 shows the training error of applying only LVQ according to the LaMa cost function with a batch update for both the prototypes and the metric. The training error does not improve. In fact most curves are worse than the LaMa metric update.

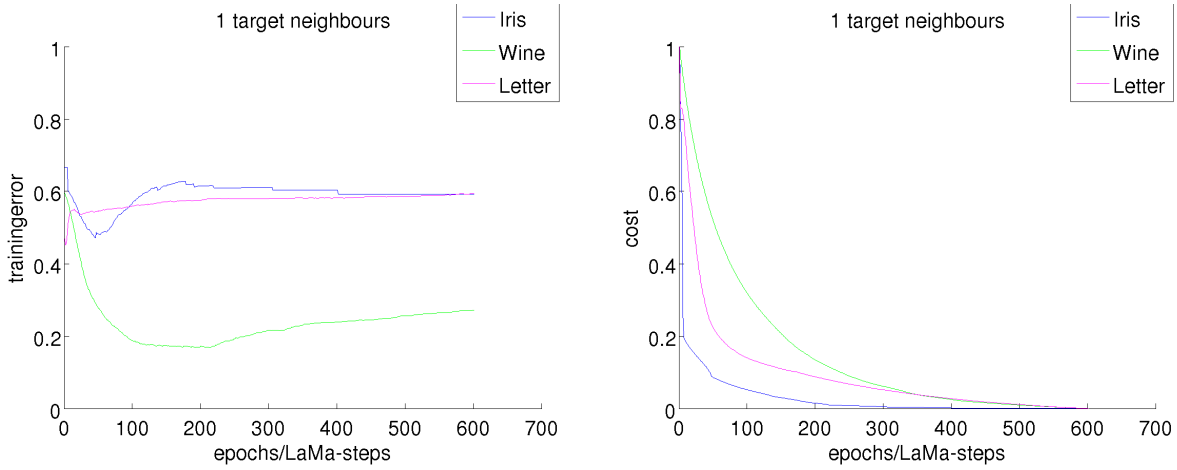


(a) 5 target neighbours

(b) 50 target neighbours

Figure 11: Batch update for both prototype and metric without LaMa update: cost progress for 5 and 50 target neighbours

Also note the cost progress in Figure 11 which (except for a few exceptions) is not smooth at all. This is again due to the changes in target neighbours which a metric update might cause.



(a) training error

(b) cost progress

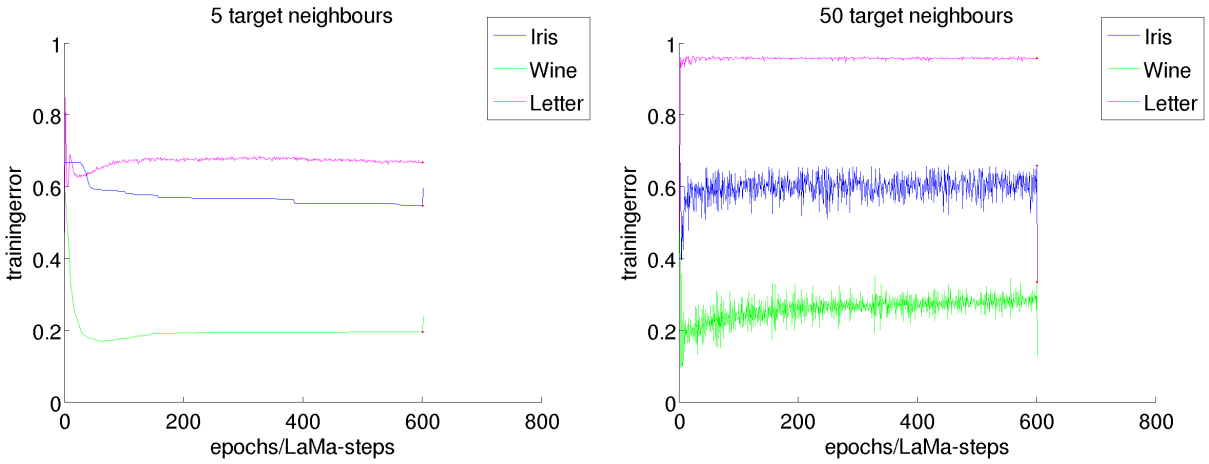
Figure 12: Batch update for both prototype and metric without LaMa update: training error and cost progress for 1 target neighbours

But when the number of target neighbours is low as in Figure 12 the cost retains its smoothness. In this case target neighbours either barely change or target neighbours are more smoothly transferred. The training error here shows a smooth cost progress is not necessarily a good or bad thing for classification as it depends on how well the cost function models a particular dataset.

Enhanced LVQ

rounds	1
epochs	600
metricType	single metric
LaMaVersion	Datapoint version
LaMaMode	single LaMa
protNrMode	absolute
nrprots	1
updatemode	LaMa batch update
costfunction	LaMaLVQ Datapoint Version

Table 9: Enhanced LVQ with LaMa batch update test settings



(a) 5 target neighbours

(b) 50 target neighbours

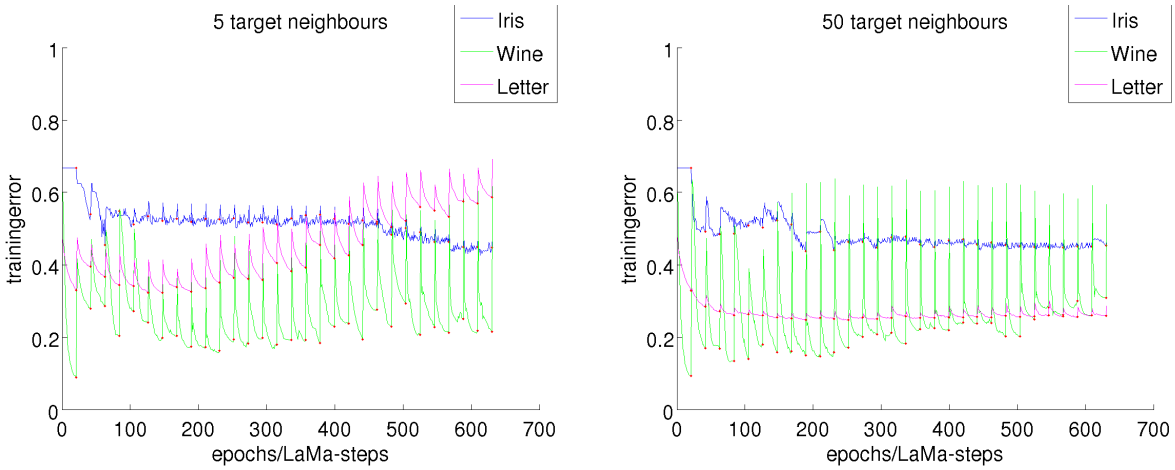
Figure 13: Enhanced LVQ with LaMa batch update: training error for 5 and 50 target neighbours

If LVQ with a LaMa batch update can give us nice smooth results maybe we can use this by performing only one metric update at the end. Figure 13 shows the training error of this experiment. Results differ somewhat between the number of target neighbours used, though a large number of target neighbours seems to give the best improvement at the end. Still, only the fairly simple datasets Wine and Iris improve in classification.

Datapoint version with generalized update

metricType	single metric
LaMaVersion	Datapoint version
LaMaMode	single LaMa
protNrMode	absolute
nrprots	1
updatemode	generalized update
costfunction	LaMaLVQ Datapoint Version

Table 10: Datapoint generalized update test settings



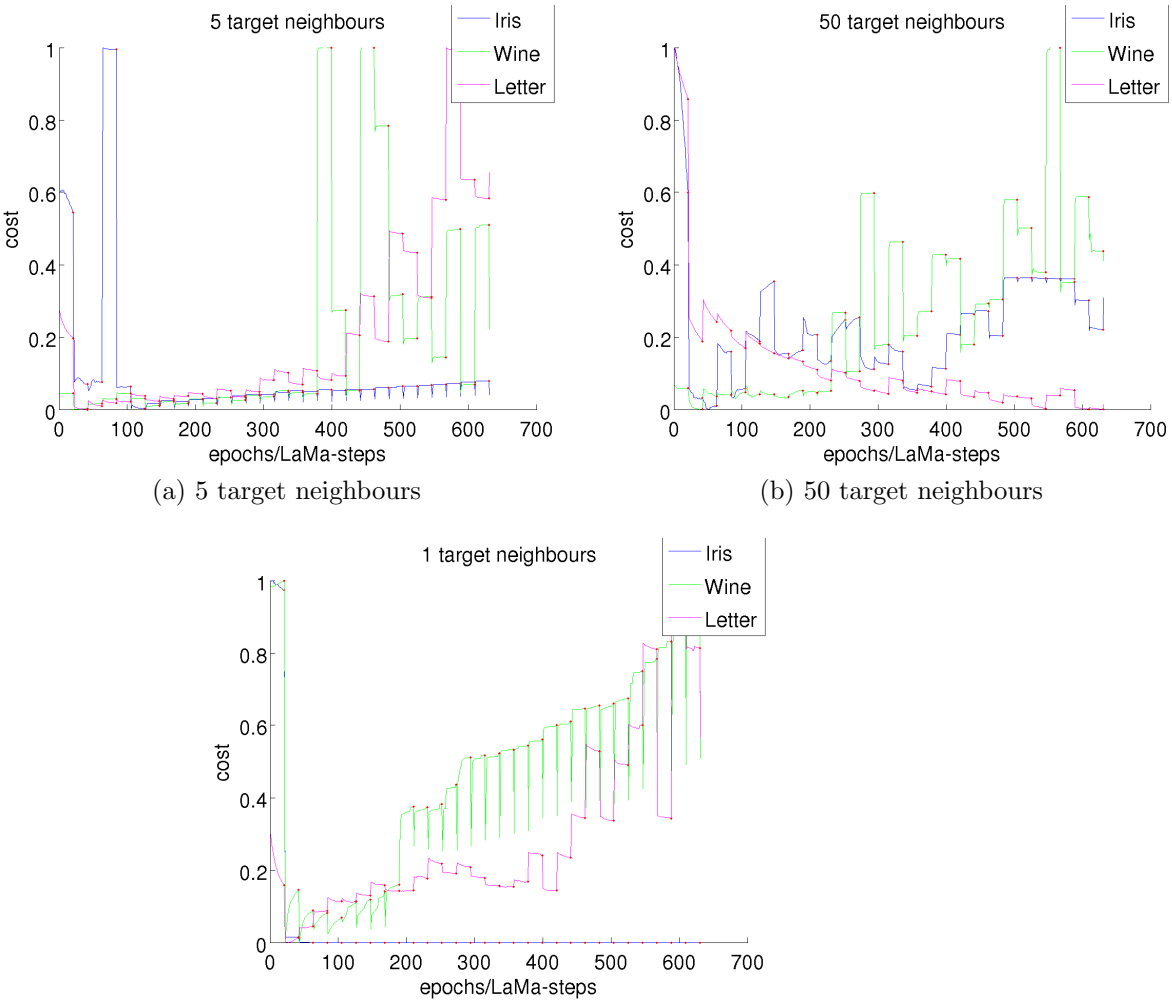
(a) 5 target neighbours

(b) 50 target neighbours

Figure 14: Generalized update with Datapoint LaMa version: training error for 5 and 50 target neighbours

When we look at Figure 14 (the training error of the Datapoint version with an LVQ update based on the generalized cost function), the first thing that stands out is the jutting up and down of the training error. This is most pronounced with the Wine dataset. Note that all spikes upward happen just after a LaMa step. The metric changes, drastically changing the classification as opposed to the Datapoint version with LaMa batch update with which the classification changed only slightly after a LaMa step. The difference here of course being the positioning of the prototypes. When the prototypes are already positioned according to the LaMa cost function a metric change according to that same function will not change the classification much. However when both positioning of the prototypes and the obtaining of the metric try and optimize a different cost function both steps can drastically change the classification. The Wine dataset shows this most clearly and the Iris and Letter dataset show this in a less drastic fashion. The changing of the metric worsens the classification considerably after which the prototypes are repositioned and quickly recover the classification. This procedure sometimes

can change the classification favourably or unfavourably. It can go either way as the Iris dataset seems to benefit from it, while the Wine and Letter dataset classification deteriorate on the whole.
Surprisingly, optimizing two different cost functions at the same time does not seem to be detrimental to the classification at all. In this experiment, using 50 target neighbours seems to improve classification of all datasets, with the Letter dataset in the lead with a training error slightly above 20%.



(a) 5 target neighbours

(b) 50 target neighbours

(c) 1 target neighbour

Figure 15: Generalized update with Datapoint LaMa version: cost progress for 5, 50 and 1 target neighbours

The cost function in Figure 15 shows more instability than when using the LaMa batch update, quickly increasing and decreasing after most LaMa

steps. The Wine dataset sometimes shows the inverse of what is happening in the training error, reducing in cost right after a LaMa step and quickly increasing again to a comparable cost after a prototype repositioning. We can see this most clearly in Figure 15c.

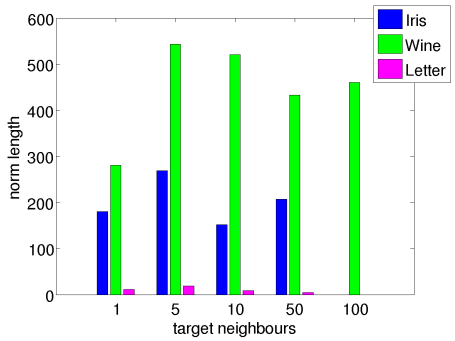
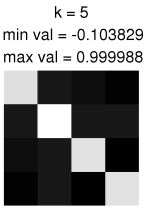
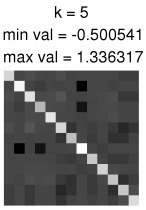


Figure 16: Generalized update with Datapoint version LaMa metric average prototype norms for different numbers of target neighbours

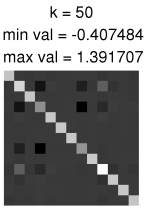


(a) Iris

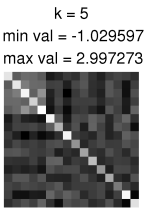
(b) Iris



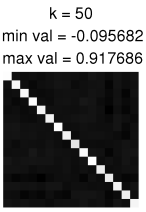
(c) Wine



(d) Wine



(e) Letter



(f) Letter

Figure 17: Average of resulting matrices of LaMa LVQ with generalized update, no normalization or zeroing of diagonal

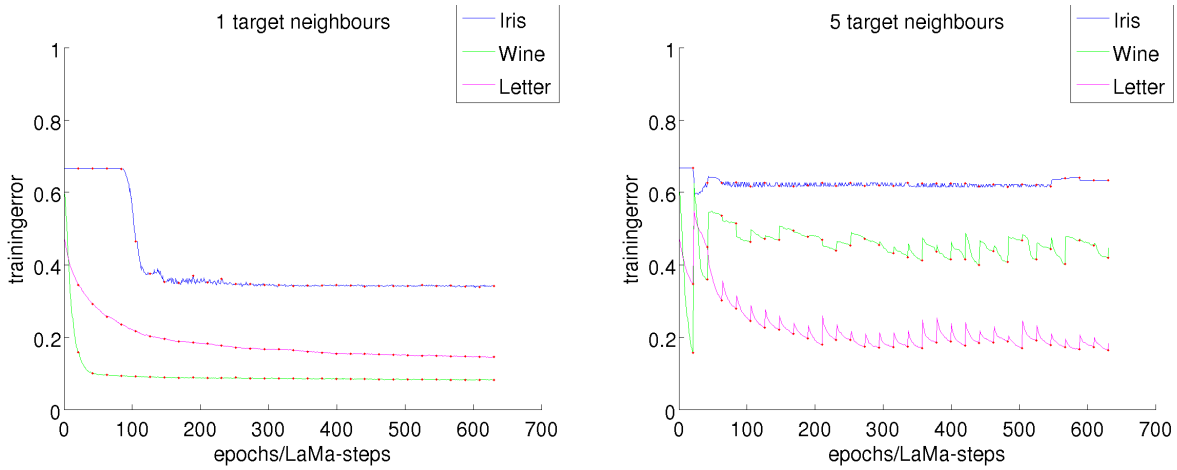
The average prototype norm and metrics in Figures 16 and 17 respectively show some interesting information. The Iris and Wine norms are very

high (with the Wine dataset taking the cake) and the norms of the Letter dataset are relatively small. No metric has an exceptional scale. Earlier a high average norm and a non exceptional metric meant the training error would be high, but here the training error of the Iris and Wine dataset have not deteriorated. This is most likely caused by the small number of classes these datasets contain. These datasets have only three classes so even if data points would pick prototypes at random they would on average be correctly classified one third of the time. Therefore prototypes do not have to be positioned at very precise locations to get the results we see in Figure 14. The Letter dataset on the other hand has 26 classes and prototypes which are far away are much more likely to be closer to different class clusters than the specific cluster they represent. Indeed, the good classification of the Letter dataset in this experiment is partially owing to the low prototype norm.

Prototype version with generalized update

metricType	single metric
LaMaVersion	Prototype version
LaMaMode	single LaMa
protNrMode	percentage
protperc	0.1 for Iris and Wine, 0.2 for Letter
updatemode	generalized update
costfunction	LaMaLVQ Prototype Version

Table 11: Prototype version generalized update test settings

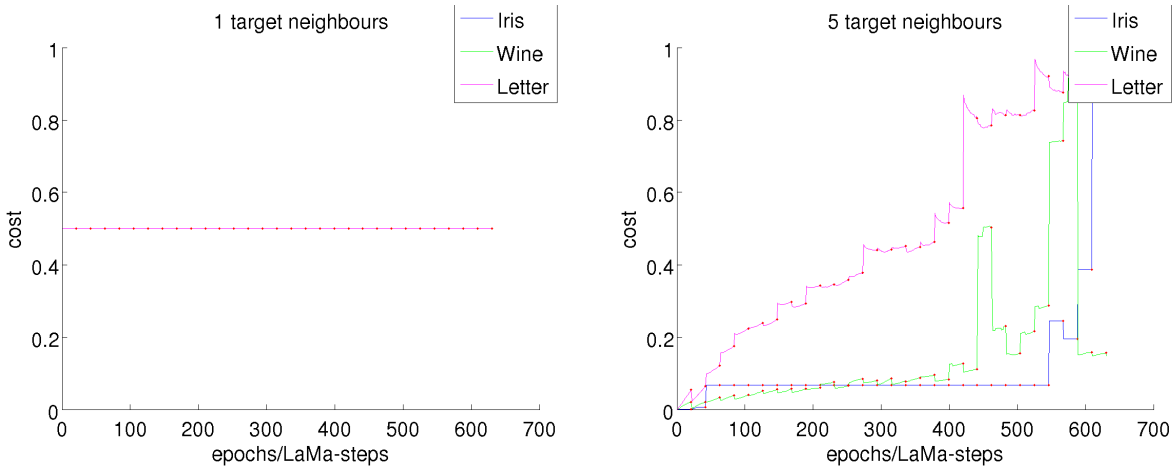


(a) 1 target neighbours

(b) 5 target neighbours

Figure 18: Generalized update with Prototype LaMa version: training error for 1 and 5 target neighbours

Figure 18 show the training error of using the Prototype version of LaMaLVQ with a generalized update. Here all datasets including the Letter dataset show a significant improvement in classification. The Iris and Wine datasets' classification worsens when more target neighbours are used, but the Letter dataset retains its classification, although it is less smooth and the LaMa steps are more pronounced.



(a) 1 target neighbours

(b) 5 target neighbours

Figure 19: Generalized update with Prototype LaMa version: cost progress for 1 and 5 target neighbours

The cost progress using one target neighbour in Figure 19 seems very suspicious. It does not change at all. In reality it does change a little, but the prototypes are bunched very close together. The distance to each other is smaller than the distance threshold used by the implementation to prevent errors.
The cost progress using five target neighbours seems completely removed from the training error. A metric update can suddenly change target neighbours as before and the LVQ update scheme is not based on the cost function either.

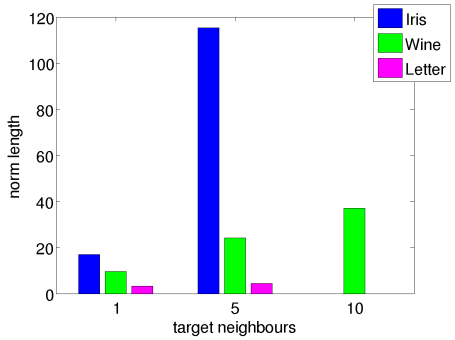
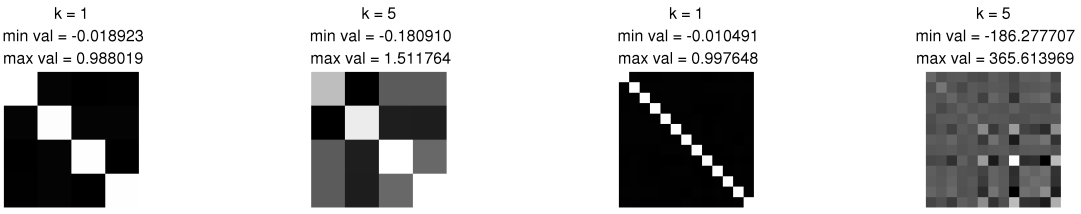
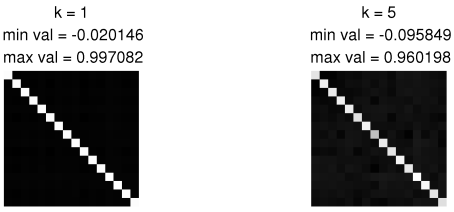


Figure 20: Generalized update with Prototype version LaMa metric average prototype norms for different numbers of target neighbours



(a) Iris (b) Iris (c) Wine (d) Wine



(e) Letter (f) Letter

Figure 21: Average of resulting matrices of Prototype version LaMaLVQ with generalized update, no normalization or zeroing of diagonal

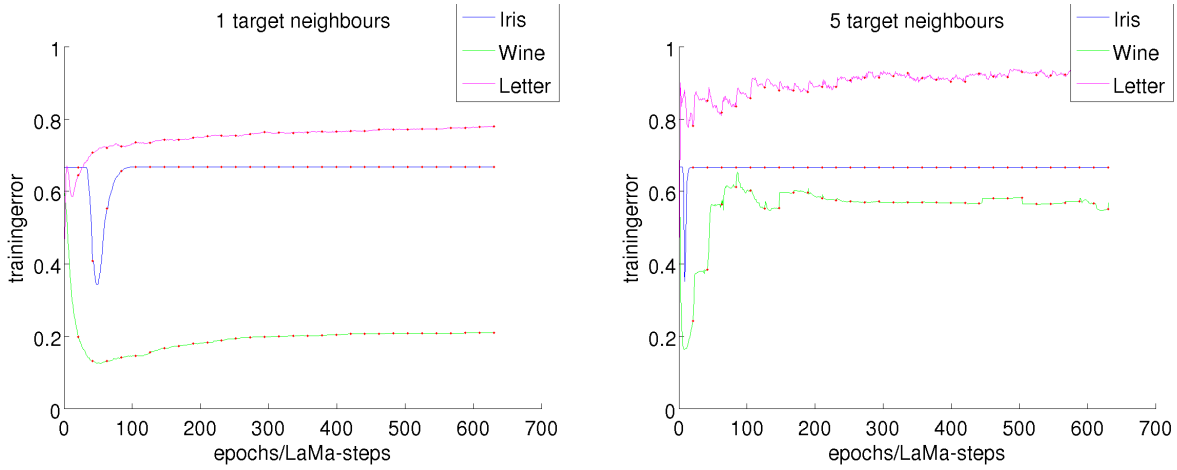
When we search for explanations in the prototype average norms and metrics in Figures 20 and 21 respectively we see the decline in the Iris dataset classification can be attributed to the high prototype norm and non exceptional metric. The norm of the Wine dataset increases as well, though not as dramatically as the Iris norm. The metric also shows a large scale to match the prototype norm, however since this is the Prototype version of LaMaLVQ the metric is optimized according to only the prototype positions. With both

the prototypes and the metric being quite removed from the dataset the classification suffers. The norm and scale of metric of the Letter dataset are both quite low, meaning the prototypes are still close to the dataset. With both the prototypes and the metric closely tied to the dataset the Prototype version of LaMaLVQ gives quite a good representation of the dataset.
Also note that all experiments which show an improvement to classification are connected to metrics with a dominant diagonal.

Prototype version with batch update

metricType	single metric
LaMaVersion	Prototype version
LaMaMode	single LaMa
protNrMode	percentage
protperc	0.1 for Iris and Wine, 0.2 for Letter
updatemode	LaMacost batch update
costfunction	LaMaLVQ Prototype Version

Table 12: Prototype version batch update test settings



(a) 1 target neighbours

(b) 5 target neighbours

Figure 22: Batch update with Prototype LaMa version: training error for 1 and 5 target neighbours

Figure 22 shows the training error of using the Prototype version with the LaMa batch update. It shows comparable results to Figure 2. This shows that a change in LaMa versions is not the only reason why the Letter dataset in Figure 18 improves in classification.

Discussion

The first experiment which uses a LaMa batch update and metric update improves classification for the relatively simple Wine and Iris datasets only. The sudden and drastic change in metric causes a step-like progress both in the training error and in the cost progress. The metric changes dominate the progress, while the positioning of the prototypes seem to have only slight impact and only serve to generate a different metric by slightly different positioning. Here we can already see two problems of this approach:
The first is the halting of classification improvement by early optimization. When you look at the training error of the Iris dataset in Figures 2d and 2c the curve show the beginning of a smooth downward curve, but after a few LaMa updates have been performed the curve shoots up flats out. LVQ was busy positioning the prototypes according to the LaMa cost function when the metric was optimized according to that same cost function. This results in the positions of the prototypes suddenly becoming (near) optimal according to the cost function, although according to the classification this might not be optimal at all and classification might even deteriorate. Still the prototype positions are considered (near) optimal and will move in a different direction if at all. The improvement in classification gets halted by the metric suddenly minimizing the cost of the prototypes configuration.
The second problem can be seen in the training error of the Wine dataset in Figures 2b and 2a. Here after a promising descent of the training error, just after the first LaMa update the curve shoots up and stays at a more or less stable, higher point. When the new metric was computed drastically changed the distance calculation and thus the prototype updates pushing these in a different direction. After the prototypes went through another positioning the new metric computation produced a metric which largely stabilized the prototype configuration and therefore a more and more stable classification. The changes after LaMa steps also lessen. This means the target neighbours changed less and less. Since the training error has been much lower, a local minimum has been found.
A local (as opposed to a global) minimum has been found, because after the first LaMa update the prototypes were pushed in a different direction. This direction stabilized the metric even though it was detrimental to the classification. This is caused by the 'nearsightedness' of the LaMa step. It takes only a number of target neighbours into account and bases the metric on only a portion of the dataset.

The second experiment, which does not use LaMa steps and only applies LVQ according to the LaMa batch update, shows some varying but hopeful results. Although the different datasets react different to the LaMa batch update, the training error and cost progress show a smooth curve for a low number of target neighbours. The curve gets a bit more erratic with more target neighbours, but on the whole a clear progress can be seen. The more or less smooth decline of the cost progress merely shows the correctness of the implementation of the LaMa batch update according to the cost function, but the steepness shows that the more target neighbours are used the faster the cost function is optimized and an optimum is reached. A cost

function using a simple Euclidean distance measure might not be suitable for some datasets like the Letter dataset, since the training error worsens as the number of target neighbours increases.

A less drastic approach is attempted in Figure 10 where no LaMa steps are performed and the metric is only updated by LVQ. The results are invariably worse than LVQ without a matrix update. The cost progress also shows instability, quickly increasing and decreasing. This is due to the batch nature of the matrix updates. The update of all concerned data points are collected and applied as a whole to the matrix. This causes a drastic change in metric which might change the target neighbours. Although the update is designed to lower the cost, a drastic change in metric can cause a drastic increase or decrease in cost. With this in mind a smoother update scheme might do the trick: an online update. However, this would require a re-computation of the target neighbours and invaders at every step. This a very costly computation which in he scope of this article was not feasible.

What causes all this confusion with new target neighbours is again the 'nearsightedness' of the metric optimization. The optimization only takes into account the current target neighbours. When the optimum has been reached new target neighbours might be identified and the new metric might not be optimal for these new target neighbours. Computing multiple metrics as in the consecutive LaMa technique is not a solution either, since the existence of target neighbours which generate a metric with themselves as the target neighbours is not guaranteed. Consecutive metric optimizations might start oscillating between a number of sets of target neighbours. Because of this property a LaMa step is not guaranteed to improve classification or even reduce the cost of the function it is optimizing. With this knowledge we might conclude that we need to incorporate the selection of the target neighbours in the metric calculation. However, making the target neighbours another variable would make the cost function non-convex, making finding an optimal solution very difficult.

The results of Figure 13 show improvements in classification over only LVQ in some cases. In many other cases the double optimization changes little. In this experiment the metric was optimized only once at the end of a long LVQ step. In the experiments with a smooth training error and cost progress the last LaMa step does not always make a significant difference. Here the reverse of the past discussed experiments happens: First the cost function is optimized by positioning the prototypes then the LaMa step optimizes the cost function by optimizing the metric. However the prototype positions already optimized the cost function (and apparently found a global optimum as opposed to a local optimum) and an optimization of the metric produces a nearly identical metric which does not change the training error or the cost. Only in the instances where the progress is more erratic does the LaMa step at the end make a difference. Apparently the optimum here is surrounded by local optima which the prototypes alternate between. Here the LaMa step cuts the proverbial knot forcing the prototype positions to be optimal by changing the metric.

When we use LaMaLVQ with a different LVQ update scheme such as the generalized update we see more evidence of the incompatibility of the LaMaLVQ cost function with the used datasets. The results in Figures 14 and 15 show many points at which a LaMa update is performed where the cost drops drastically, but the training error increases in an equally drastic fashion.

Now let us take a look at the results of the Prototype version of LaMaLVQ. Right off the bat we see a more positive image. The results in Figure 18 show a stable decline in training error for all datasets. Here the prototypes stay close together and the the metric is only dependent on the prototypes. Assuming the prototypes stay near the center of the cluster they represent, the metric causes the clusters to stay separate in the warped space. Both the stability and the training error deteriorate again when the number of target neighbours is increased, except for the Letter dataset. The Letter dataset has many more classes than the Iris and Wine dataset and therefore more prototypes are used to represent the dataset. As the Prototype version looks only at the prototypes to determine the metric, the Letter dataset has much more information to obtain a suitable metric than the Wine or Iris dataset. This results in a metric which is better suited for the classification. Also note most of the classification improvement once again evaporates when the LVQ update switches back to LaMa batch in Figure 22.

Conclusion

The combination of LVQ and LMNN produces varying results depending on the sort of dataset and LaMa variation used. The optimization of a metric in one step causes sudden shifts in the perception of the space in which the data resides. This can cause a sudden increase or decrease in classification error and causes overall instability in the classification and cost progress. Other optimization schemes which change the metric more smoothly either show the same behaviour (MLVQ with LaMa batch metric update) or are too costly (MLVQ with an online LaMa update).
Optimizing a metric too early may lead to halting improvement or even deterioration of classification. Positioning of the prototypes and optimization of the metric does not need to follow the exact same goal. Identical optimization goals for both aspects can have detrimental effect on classification, since early positioned prototypes might not have arrived on near optimal positions, but a metric update can make the positions near optimal with regard to the cost function while ignoring classification. Differing optimization goals can achieve good classification by letting one part of the optimization explore new directions when another part of the scheme has already optimized according to its own goal.
Although the LMNN problem (and as extension the metric computation in LaMaLVQ) the problem LaMaLVQ tries to solve is not. In fact LaMaLVQ is even more susceptible to local minima than LVQ. Not only can the proto-

type positioning find a local minimum. A metric update can create a local minimum near the prototypes by optimizing the metric with regard to the prototype configuration. Or the 'nearsightedness' of the metric optimization can push the prototypes towards a local minimum where the target neighbours (and by extension the metric) will be more stable.
LaMaLVQ sometimes pushes prototypes away from the dataset. This can be quite detrimental to classification, since the metric partially or wholly determined by the prototype configuration. Classification of datasets with many classes suffer more than datasets with fewer classes when this occurs. Badly positioned prototypes can still correctly classify a portion of the dataset if they represent a large area. However, the Prototype version is best applied when it is possible to use many prototypes to represent dataset. With more prototypes, more information is available on which to base a newly computed metric.

Acknowledgements

The datasets used in this article were made available by The UCI Machine Learning Repository [6].

References

[1] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *J. Mach. Learn. Res.*, 6:937–965, December 2005.

[2] Kerstin Bunte, Petra Schneider, Barbara Hammer, Frank-Michael Schleif, Thomas Villmann, and Michael Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26(0):159 – 173, 2012.

[3] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research, JMLR*, pages 1109–1135, 2010.

[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1), January 1967.

[5] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188, 1936.

[6] A. Frank and A. Asuncion. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2010. http://archive.ics.uci.edu/ml.

[7] Barbara Hammer and Thomas Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15:1059–1068, 2002.

[8] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(13):1 – 6, 1998.

[9] D.G. Stork R.O. Duda, P.E. Hart. *Pattern Classification*. John Wiley & Sons, 2001.

[10] L. VandenBerghe S. Boyd. *Convex Optimization*. Cambridge University Press, 2004.

[11] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in Learning Vector Quantization. *Neural Computation*, 21(12):3532–3561, 2009.

[12] Sambu Seo and Klaus Obermayer. Soft learning vector quantization. *NEURAL COMPUTATION*, 15:1589–1604, 2002.

[13] Killian Q. Weinberg and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[14] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. pages 505–512, 2002.

[15] A.S. Sato & K. Yamada. Generalized learning vector quantizarion. *Advances in Neural Information Processing Systems*, 8:423–429, 1996.