Learning vector quantization for (dis-)similarities

Barbara Hammer, Daniela Hofmann, Frank-Michael Schleif, Xibin Zhu
CITEC center of excellence, Bielefeld University, Germany

March 5, 2013

Abstract

Prototype-based methods often display very intuitive classification and learning rules. However, popular prototype based classifiers such as learning vector quantization (LVQ) are restricted to vectorial data only. In this contribution, we discuss techniques how to extend LVQ algorithms to more general data characterized by pairwise similarities or dissimilarities only. We propose a general framework how the methods can be combined based on the background of a pseudo-Euclidean embedding of the data. This covers the existing approaches kernel generalized relevance LVQ and relational generalized relevance LVQ, and it opens the way towards two novel approach, kernel robust soft LVQ and relational robust soft LVQ. Interestingly, also unsupervised prototype based techniques which are based on a cost function can be put into this framework including kernel and relational neural gas and kernel and relational self-organizing maps (based on Heskes' cost function). We demonstrate the performance of the LVQ techniques for similarity or dissimilarity data in several benchmarks, reaching state of the art results.

1 Introduction

Since electronic data sets increase rapidly with respect to size and complexity, humans have to rely on automated methods to access relevant information from such data. Apart from classical statistical tools, machine learning has become a major technique in the context of data processing since it offers a wide variety of inference methods. Today, a major part of applications is concerned with the inference of a function or classification prescription based on a given set of examples, accompanied by data mining tasks in unsupervised machine learning scenarios and more general settings

as tackled e.g. in the frame of autonomous learning. In this contribution, we focus on classification problems.

There exist many different classification techniques in the context of machine learning ranging from symbolic methods such as decision trees to statistical methods such as Bayes classifiers. Because of its often excellent classification and generalization performance, the support vector machine (SVM) constitutes one of the current flagships in this context, having its roots in learning theoretical principles as introduced by Vapnik and colleagues [4]. Due to its inherent regularization of the result, it is particularly suited if high dimensional data are dealt with. Further, the interface to the data is given by a kernel matrix such that, rather than relying on vectorial representations, the availability of the Gram matrix is sufficient to apply this technique.

With machine learning techniques becoming more and more popular in diverse application domains and the tasks becoming more and more complex, there is an increasing need for models which can easily be interpreted by practitioners: for complex tasks, often, practitioners do not only apply a machine learning technique but also inspect and interpret the result such that a specification of the tackled problem or an improvement of the model becomes possible [40]. In this setting, a severe drawback of many state-of-the-art machine learning tools such as the SVM occurs: they act as black-boxes. In consequence, practitioners cannot easily inspect the results and it is hardly possible to change the functionality or assumptions of the model based on the result of the classifier.

Prototype-based methods enjoy a wide popularity in various application domains due to their very intuitive and simple behavior: they represent their decisions in terms of typical representatives contained in the input space and a classification is based on the distance of data as compared to these prototypes [19]. Thus, models can be directly inspected by experts since prototypes can be treated in the same way as data. Popular techniques in this context include standard learning vector quantization (LVQ) schemes and extensions to more powerful settings such as variants based on cost functions or metric learners such as robust soft LVQ (RSLVQ) or generalized LVQ (GLVQ), for example [32, 37, 38, 36]. These approaches are based on the notion of margin optimization similar to SVM in case of GLVQ [37], or based on a likelihood ratio maximization in case of RSLVQ, respectively [38]. For GLVQ and RSLVQ, a behavior which closely resembles standard LVQ2.1 results in limit cases. The limit case of RSLVQ does not necessarily achieve optimum behavior already in simple model situations similar to LVQ2.1, as has been investigated in the context of the theory of online learning

[2]. Nevertheless, it displays excellent generalization ability in the standard intermediate case, see e.g. [36] for an extensive comparison of the techniques.

With data sets becoming more and more complex, input data are often no longer given as simple Euclidean vectors, rather structured data or dedicated formats can be observed such as sequences, graphs, tree structures, time series data, functional data, relational data etc. as occurs in bioinformatics, linguistics, or diverse heterogeneous databases. Several techniques extend statistical machine learning tools towards non-vectorial data: kernel methods such as SVM using structure kernels, recursive and graph networks, functional methods, relational approaches, and similar [9, 33, 11, 31, 14].

Recently, popular prototype-based algorithms have also been extended to deal with more general data. Several techniques rely on a characterization of the data by means of a matrix of pairwise similarities or dissimilarities only rather than explicit feature vectors. In this setting, median clustering as provided by median self-organizing maps, median neural gas, or affinity propagation characterizes clusters in terms of typical exemplars [10, 20, 8]. More general smooth adaptation is offered by relational extensions such as relational neural gas or relational learning vector quantization [13]. A further possibility is offered by kernelization such as proposed for neural gas, self-organizing maps, or different variants of learning vector quantization [29, 41, 30]. By formalizing the interface to the data as a general similarity or dissimilarity matrix, complex structures can be easily dealt with: structure kernels for graphs, trees, alignment distances, string distances, etc. open the way towards these general data structures [27, 11].

In this contribution, we consider the question how to extend cost function based LVQ variants such as RSLVQ (11) or GLVQ (2) to similarity or dissimilarity data, respectively. We propose a general way based on an implicit pseudo-Euclidean embedding of the data, and we discuss in how far instantiations of this framework differ from each other. Using this framework, we cover existing techniques such as kernel GLVQ [30] and relational GLVQ [15], and investigate novel possibilities such as kernel and relational RSLVQ. These techniques offer valid classifiers and training methods for an arbitrary symmetric similarity or dissimilarity. Some mathematical properties, however, such as an interpretation via a likelihood ratio or interpretation of learning as exact gradient, are only guaranteed in the Euclidean case for some of the possible choices, as we will discuss in this article. In this context, we investigate the effect of corrections of the matrix to make data Euclidean. The effectivity of the novel techniques is demonstrated in a couple of benchmarks.

Now, we first introduce standard LVQ for Euclidean vectors, in partic-

ular the two cost-function based variants GLVQ and RSLVQ. Afterwards, we review facts about similarity and dissimilarity data and their pseudo-Euclidean embedding. Based on this embedding, kernel and relational variants of LVQ can be introduced for similarities or dissimilarities, respectively. Training can take place essentially in two ways, mimicking the corresponding Euclidean counterparts or via direct gradients, respectively, whereby the same local optima of the cost function are present in the Euclidean case, but a numerical scaling of the gradients is observed. We exemplarily derive new models, kernel RSLVQ and relational RSLVQ in this framework. Experiments are based on the setting as proposed in [6], investigating the effect of different preprocessing steps and learning techniques in comparison to the results of SVM and a k-nearest neighbor classifier. We conclude with a discussion.

2 Learning vector quantization

Learning vector quantization (LVQ) constitutes a very popular class of intuitive prototype based learning algorithms with successful applications ranging from telecommunications to robotics [19]. Basic algorithms as proposed by Kohonen include LVQ1 which is directly based on Hebbian learning, and improvements such as LVQ2.1, LVQ3, or OLVQ which aim at a higher convergence speed or better approximation of the Bayesian borders. These types of LVQ schemes have in common that their learning rule is essentially heuristically motivated and a valid cost function does not exist [3]. One of the first attempts to derive LVQ from a cost function can be found in [32] with an exact computation of the validity at class boundaries in [36]. Later, a very elegant LVQ scheme which is based on a probabilistic model and which can be seen as a more robust probabilistic extension of LVQ2.1 has been proposed in [38]. We shortly review these two proposals.

2.1 Generalized learning vector quantization

Assume data $\xi_i \in \mathbb{R}^n$ with $i = 1, \ldots, N$ are labeled y_i where labels stem from a finite number of different classes. A GLVQ network is characterized by m prototypes $w_j \in \mathbb{R}^n$ with priorly fixed labels $c(w_j)$. Classification takes place by a winner takes all scheme:

$$\xi \mapsto c(w_j) \text{ where } d(\xi, w_j) \text{ is minimum} \tag{1}$$

with squared Euclidean distance $d(\xi, w_j) = \|\xi - w_j\|^2$, breaking ties arbitrarily.

For training, it is usually assumed that the number and classes of prototypes are fixed. In practice, these are often determined using cross-validation, or a further wrapper technique is added to obtain model flexibility. Training aims at finding positions of the prototypes such that the classification accuracy of the training set is optimized. GLVQ also takes the generalization ability into account, using the costs

$$\sum_i \frac{d(\xi_i, w^+) - d(\xi_i, w^-)}{d(\xi_i, w^+) + d(\xi_i, w^-)} \tag{2}$$

where w^+ constitutes the closest prototype with the same label as ξ_i and w^- constitutes the closest prototype with a different label than ξ_i. The nominator is negative iff ξ_i is classified correctly, thus GLVQ tries to maximize the number of correct classifications. In addition, it aims at an optimization of the hypothesis margin $d(\xi_i, w^-) - d(\xi_i, w^+)$ which determines the generalization ability of the method [37].

Training takes place by a simple stochastic gradient descent, i.e. given a data point ξ_i, adaptation takes place via

$$\Delta w^+ \sim -\frac{2 \cdot d(\xi_i, w^-)}{(d(\xi_i, w^+) + d(\xi_i, w^-))^2} \cdot \frac{\partial d(\xi_i, w^+)}{\partial w^+} \tag{3}$$

$$\Delta w^- \sim \frac{2 \cdot d(\xi_i, w^+)}{(d(\xi_i, w^+) + d(\xi_i, w^-))^2} \cdot \frac{\partial d(\xi_i, w^-)}{\partial w-} \tag{4}$$

From an abstract point of view, we can characterize GLVQ as a classifier, which classification rule is based on a number of quantities

$$D(\xi, w) := (d(\xi_i, w_j))_{i=1,\ldots,N, j=1,\ldots,m} \tag{5}$$

Training aims at an optimization of a cost function of the form

$$f(D(\xi, w)) \tag{6}$$

by means of the gradients

$$\frac{\partial f(D(\xi, w))}{\partial w_j} = \sum_{i=1}^{m} \frac{\partial f(D(\xi, w))}{\partial d(\xi_i, w_j)} \cdot \frac{\partial d(\xi_i, w_j)}{\partial w_j} \tag{7}$$

with respect to the prototypes w_j or the corresponding stochastic gradients for one point ξ_i.

2.2 Robust soft learning vector quantization

Robust soft LVQ (RSLVQ) models data by a mixture of Gaussians and derives learning rules as a maximization of the log likelihood ratio of the given data. In the limit of small bandwidth σ, a learning rule which is similar to LVQ2.1 but which performs adaptation in case of misclassification only, is obtained [38].

Here, we restrict to the standard model which assumes equal prior and bandwidth of the modes. Mixture component j defines the probability

$$p(\xi|j) = (2\pi\sigma^2)^{-n/2} \cdot \exp(-d(\xi, w_j)/\sigma^2) \tag{8}$$

This induces the probability of an unlabeled data point

$$p(\xi|W) = \frac{1}{m} \cdot \sum_j p(\xi|j) \tag{9}$$

with parameters W of the model. The probability of a labeled data point is

$$p(\xi, y|W) = \frac{1}{m} \cdot \sum_{j\,:\,c(w_j)=y} p(\xi|j)\,. \tag{10}$$

Learning aims at an optimization of the log likelihood ratio

$$L = \sum_i \log \frac{p(\xi_i, y_i|W)}{p(\xi_i|W)}\,. \tag{11}$$

A stochastic gradient ascent yields the following update rules, given data point ξ_i with label y_i:

$$\Delta w_j \sim \begin{cases} -(P_y(j|\xi_i) - P(j|\xi_i)) \cdot \partial d(\xi_i, w_j)/\partial w_j & \text{if } c(w_j) = y_i \\ P(j|\xi_i) \cdot \partial d(\xi_i, w_j)/\partial w_j & \text{if } c(w_j) \neq y_k \end{cases} \tag{12}$$

with the probabilities

$$P_y(j|\xi_i) = \frac{\exp(-d(\xi_i, w_j)/\sigma^2)}{\sum_{j:c(w_j)=y_i} \exp(-d(\xi_i, w_j)/\sigma^2)} \tag{13}$$

and

$$P(j|\xi_i) = \frac{\exp(-d(\xi_i, w_j)/\sigma^2)}{\sum_j \exp(-d(\xi_i, w_j)/\sigma^2))} \tag{14}$$

With small bandwidth, a learning rule similar to LVQ2.1, learning from mistakes, results thereof.

Given a novel data point ξ, its class label can be determined by means of the most likely label y corresponding to a maximum value $p(y|\xi, W) \sim p(\xi, y|W)$. For typical settings, this rule can usually be approximated by a simple winner takes all rule as in GLVQ (1). It has been shown in [38], for example, that RSLVQ often yields excellent results while preserving interpretability of the model due to prototypical representatives of the classes in terms of the parameters w_j.

Note that the objective of RSLVQ and GLVQ training is in both cases a function of the form $f(D(\xi, w))$. Similarly, the classification depends on the vector $D(\xi, w)$ only.

3 Extensions to dis-/similarity data

In modern applications, data are often no longer vectorial. Rather, complex structures are dealt with for which a problem specific similarity or dissimilarity measure has been designed. Examples include biological sequences, mass spectra, or metabolic networks, where complex alignment techniques, background information, or general information theoretical principles, for example, drive the comparison of data points [28, 22, 18]. In these settings, it is possible to compute pairwise similarities or dissimilarities of the data rather than to arrive at an explicit vectorial representation.

The question occurs how LVQ algorithms can be extended to this setting. Two different principles have been proposed in the literature: kernel GLVQ assumes a valid Gram matrix and extends GLVQ by means of kernelization, see [30]. In contrast, relational GLVQ assumes the more general setting of possibly non-Euclidean dissimilarities, and extends GLVQ to this setting by an alternative expression of distances based on the given dissimilarity data [15]. We will argue that both instances can be unified as LVQ variants referring to the pseudo-Euclidean embedding of similarity or dissimilarity data, respectively.

3.1 Pseudo-Euclidean embedding

Assume data are characterized by pairwise similarities $s_{ij} = s(\xi_i, \xi_j)$ or dissimilarities $d_{ij} = d(\xi_i, \xi_j)$ only, referring to the corresponding matrices as S and D, respectively. Here, we do not have any prior information about the shape of ξ_i, in particular, it is not necessarily represented in vectorial form. We assume symmetry, i.e. $S = S^t$ and $D = D^t$ as well as zero diagonal in D, i.e. $d_{ii} = 0$. The first question is how these two representations D and S are related.

How to turn similarities into dissimilarities and vice versa?

There exist classical methods to turn similarities to dissimilarities and vice versa, see e.g.[27]: given a similarity, a dissimilarity is obtained by the transformation

$$\Phi : S \to D, d_{ij} = s_{ii} - 2s_{ij} + s_{jj} \tag{15}$$

while the converse is obtained by double centering

$$\Psi : D \to S, s_{ij} = -\frac{1}{2}\left(d_{ij} - \frac{1}{N}\sum_i d_{ij} - \frac{1}{N}\sum_j d_{ij} + \frac{1}{N^2}\sum_{i,j} d_{ij}\right) \tag{16}$$

While it holds that the composition of these two transforms $\Psi \circ \Phi = \mathbf{I}$, $\mathbf{I}$ being the identity, the converse, $\Phi \circ \Psi$ yields the identity iff data are centered, since offsets of data which are characterized by dissimilarities are arbitrary and hence not reconstructable from D. That means, if S is generated from vectors via some quadratic form, the vectors should be centered in the origin. So essentially, for techniques which rely on dissimilarities of data, we can treat similarities or dissimilarities as identical via these transformations. The same holds for similarity based approaches only if data are centered.

How to represent data in vectorial form?

The key observation is that every finite data set which is characterized by pairwise similarities or dissimilarities can be embedded in a so-called pseudo-Euclidean vector space. Essentially, this is a finite dimensional real-vector space of dimensionality N, characterized by the signature $(p, q, N - p - q)$, which captures the degree up to which elements are Euclidean. Distances along the first p dimensions are Euclidean whereas the next q dimensions serve as correction factors to account for the non-Euclidean elements of the dissimilarity d. In the following, we follow the excellent presentation of pseudo-Euclidean spaces as derived in [27].

Assume a similarity matrix S or corresponding dissimilarity matrix D is given. Since S is symmetric, a decomposition

$$S = Q\Lambda Q^t = Q|\Lambda|^{1/2} I_{pq} |\Lambda|^{1/2} Q^t \tag{17}$$

with diagonal matrix Λ and orthonormal columns in the matrix Q can be found. I_{pq} denotes the diagonal matrix with the first p elements 1, the next q elements -1, and $N - p - q$ elements 0. By means of this representation, the number of positive and negative eigenvalues of S is made explicit as p

and q, respectively. We set $\xi_i = \sqrt{|\Lambda_{ii}|}q_i$, q_i being column i of Q. Further, we define the quadratic form

$$\langle u, v\rangle_{pq} = u_1v_1 + \ldots + u_pv_p - u_{p+1}v_{p+1} - \ldots - u_{p+q}v_{p+q} \tag{18}$$

Then we find

$$s_{ij} = \langle \xi_i, \xi_j\rangle_{p,q} \tag{19}$$

For a given dissimilarity matrix, we can consider the matrix $\Psi(D)$ obtained by double centering (16). This similarity matrix can be treated in the same way as S leading to vectors ξ_i such that

$$d_{ij} = \|\xi_i - \xi_j\|^2_{p,q} \tag{20}$$

where the symmetric bilinear form is associated to the quadratic form (18)

$$\|u-v\|^2_{pq} = |u_1-v_1|^2+\ldots+|u_p-v_p|^2-|u_{p+1}-v_{p+1}|^2-\ldots-|u_{p+q}-v_{p+q}|^2 \tag{21}$$

Thus, in both cases, vectors in a vector space can be found which induce the similarity or dissimilarity, respectively. The quadratic form in this vector space, however, is not positive definite. Rather, the first p components can be considered as standard Euclidean contribution whereas the next q components serve as a correction. This vector space is referred to as pseudo-Euclidean space with its characteristic signature $(p, q, N - p - q)$.

Note that dissimilarities defined via $\|u - v\|^2_{pq}$ or similarities defined via $\langle u, v\rangle_{pq}$ can become negative, albeit, often, the negative part is not large in practical applications. Similarities or dissimilarities stem from a Euclidean vector space iff $q = 0$ holds.

3.2 LVQ for (dis-)similarities

The pseudo-Euclidean embedding allows us to transfer LVQ based classifiers to similarity or dissimilarity data. Essentially, we embed data and prototypes in pseudo-Euclidean space and we instantiate the squared 'distance' $d(\xi_i, w_j)$ used in LVQ algorithms by the pseudo-Euclidean dissimilarity $\|\xi_i - w_j\|^2_{pq}$. Albeit this is no longer a 'distance' strictly speaking, we will address this quantity as such in the following.

Distance computation in LVQ for (dis-)similarities

In principle, this procedure already provides a valid classifier. However, a couple of questions arise in this context which will yield to relational or kernel LVQ schemes as used in practical applications, respectively:

(1) An explicit embedding has cubic complexity, can we avoid this? It is reasonable to restrict the position of prototypes to the convex hull of the data. Thus, we assume

$$w_j = \sum_l \alpha_{jl} \xi_l \tag{22}$$

where $\alpha_{jl} > 0$ with $\sum_l \alpha_{jl} = 1$. Then, we can compute for a given data point ξ_i:

$$\|\xi_i - w_j\|^2_{pq} = s_{ii} - 2\sum_l \alpha_{jl} s_{il} + \sum_{ll'} \alpha_{jl}\alpha_{jl'} s_{ll'} \tag{23}$$

Hence we can compute distances of all data points and prototypes based on pairwise data similarities only in quadratic time. Further, we do not need to represent prototypes w_j explicitly, rather, the coefficients α_{jl} are sufficient. Similarly, we find

$$\|\xi_i - w_j\|^2_{pq} = \sum_l \alpha_{jl} d_{il} - 1/2 \cdot \sum_{ll'} \alpha_{jl}\alpha_{jl'} d_{ll'} \tag{24}$$

provided $\sum_l \alpha_{jl} = 1$. Hence, as an alternative, we can compute distances via all pairwise dissimilarities of data in quadratic time.

This way, it is possible to compute an LVQ classifier based on pairwise dissimilarities or similarities only, representing prototypes only implicitly in terms of the coefficients α_{jl}. Often, the latter, using dissimilarities, is referred to as relational approach, see e.g. [15], while the former, using similarities, corresponds to kernelization of the classifier [30].

(2) How to provide out-of-sample extensions for a novel data point ξ? We assume that novel data points are represented in terms of their similarity or dissimilarity to the training points $s(\xi_i, \xi)$ or $d(\xi_i, \xi)$, respectively. Then, similarly, we obtain the distance

$$\|\xi - w_j\|^2_{pq} = s(\xi, \xi) - 2\sum_l \alpha_{jl} s(\xi, \xi_l) + \sum_{ll'} \alpha_{jl}\alpha_{jl'} s_{ll'} \tag{25}$$

which is based on known similarities and the coefficients only. Since the first term is a constant, we can simply drop it to compute the closest prototype for ξ. As an alternative, we find

$$\|\xi - w_j\|^2_{pq} = \sum_l \alpha_{jl} d(\xi, \xi_l) - 1/2 \cdot \sum_{ll'} \alpha_{jl}\alpha_{jl'} d_{ll'} \tag{26}$$

based on known dissimilarities and the coefficients of the prototypes.

(3) In how far does the result of the classification depend on the chosen embedding of prototypes? We have just derived formulas which compute

distances in terms of the similarities/dissimilarities only. Hence the result of the classification is entirely independent of the chosen embedding, any other embedding which yields the same similarities/dissimilarities will give the same result. Further, we can even ensure that the training process is independent of the concrete embedding, provided that learning rules are expressed in a similar way in terms of similarities or dissimilarities only. Thus, we now turn to possible training algorithms for these classifiers.

Training LVQ for (dis-)similarities

The principled way how to train such LVQ classifiers is essentially independent of the precise form of the cost function. For similarity or dissimilarity data, there exist two different possibilities to arrive at valid training rules, concrete instances of which can be found in [30, 15]. Here, we give a more fundamental view on these two possibilities and their differences.

(1) Optimization of the cost function by computing a stochastic gradient with respect to α_{jl}: The cost function of both, GLVQ (2) and RSLVQ (11) has the form $f(D(\xi, w))$ with $D(\xi, w) = (d(\xi_i, w_j))_{i=1,\ldots,N, j=1,\ldots,m}$ which becomes

$$f\left(\left(s_{ii} - 2\sum_l \alpha_{jl} s_{il} + \sum_{ll'} \alpha_{jl}\alpha_{jl'} s_{ll'}\right)_{i=1,\ldots,N, j=1,\ldots,m}\right) \tag{27}$$

for similarities or

$$f\left(\left(\sum_l \alpha_{jl} d_{il} - 1/2 \cdot \sum_{ll'} \alpha_{jl}\alpha_{jl'} d_{ll'}\right)_{i=1,\ldots,N, j=1,\ldots,m}\right) \tag{28}$$

for dissimilarities. We can smoothly vary w_j in pseudo-Euclidean space by adapting the coefficients α_{jl}. The latter can be adapted by a standard gradient technique. A gradient method with respect to α_{jl} is driven by the term

$$\frac{\partial f}{\partial \alpha_{jl}} = \sum_i \frac{\partial f((D(\xi, w))}{\partial d(\xi_i, w_j)} \cdot (-2 s_{il} + 2\sum_{l'} \alpha_{jl} s_{ll'}) \tag{29}$$

if similarities are considered or by the term

$$\frac{\partial f}{\partial \alpha_{jl}} = \sum_i \frac{\partial f(D(\xi, w))}{\partial d(\xi_i, w_j)} \cdot (d_{il} - \sum_{l'} \alpha_{jl} d_{ll'}) \tag{30}$$

for dissimilarities, providing adaptation rules for both cost functions by means of a gradient descent or ascent. We can, of course, use only one

term corresponding to ξ_i in case of a stochastic gradient technique. In these rules, only pairwise similarities or dissimilarities of data are required.

As an example, the corresponding adaptation rule of RSLVQ (12) for dissimilarities, which we will refer to as *relational RSLVQ* in the following, yields the update rule, given a data point ξ_i:

$$\Delta\alpha_{jl} \sim \begin{cases} -(P_y(j|\xi_i) - P(j|\xi_i)) \cdot (d_{il} - \sum_{l'} \alpha_{jl} d_{ll'}) & \text{if } c(w_j) = y_i \\ P(j|\xi_i) \cdot (d_{il} - \sum_{l'} \alpha_{jl} d_{ll'}) & \text{if } c(w_j) \neq y_k \end{cases} \tag{31}$$

where the probabilities are computed as before based on the dissimilarities $d(\xi_i, w_j)$ which are expressed via d_{ij}.

Note that the parameters α_{jl} are not yet normalized. This can be achieved in different ways, e.g. by explicit normalization after every adaptation step, or by the inclusion of corresponding barrier functions in the cost function, which yields additional regularizing terms of the adaptation. We will use an explicit normalization in the following, i.e. after every adaptation step, we divide the vector of coefficients by its component-wise sum.

(2) Optimization of the cost function by stochastic gradient techniques with respect to w_j: The gradient of the cost function with respect to w_j yields

$$\sum_i \frac{\partial f((D(\xi, w))}{\partial d(\xi_i, w_j)} \cdot \frac{\partial d(\xi_i, w_j)}{\partial w_j} \tag{32}$$

The dissimilarity d is defined in pseudo-Euclidean space as $d(\xi_i, w_j) = (\xi - w_j)^t \cdot I_{pq} \cdot (\xi - w_j)$ where I_{pq} is the diagonal matrix with p entries 1 and q entries -1 as before. Thus, we obtain $\partial d(\xi_i, w_j)/\partial w_j = -2 \cdot I_{pq}(\xi_i - w_j)$. Thus, we obtain the update in a stochastic gradient method, provided one data point ξ_i is chosen:

$$\Delta w_j \sim -\frac{\partial f((d(\xi_i, w_j))_{i,j})}{\partial d(\xi_i, w_j)} \cdot I_{pq} \left(\xi_i - \sum_l \alpha_{jl} \xi_l \right) \tag{33}$$

The idea of the learning rule as proposed in kernel GLVQ [30] is to decompose this update into the contributions of the coefficients α_{jl}. This is possible iff the update rule decomposes into a sum of the form $\sum_l \Delta\alpha_{jl}\xi_l$. In this case, an update of the coefficients which is proportional to the terms $\Delta\alpha_{jl}$ of this decomposition exactly mimics the effect of a stochastic gradient for the prototype w_j.

This decomposition, however, is usually not possible: While most components of (33) obey this form since they do not refer to components of the vector ξ_i, the ingredient I_{pq} refers to a vectorial operation which depends

on the pseudo-Euclidean embedding. Thus, it is in general not possible to turn this adaptation rule into a rule which can be done implicitly without explicit reference to the pseudo-Euclidean embedding.

In one very relevant special case, however, it is possible: assume data are Euclidean, i.e. $q = 0$. In this case, we can assume without loss of generality that p equals the dimensionality of the vectors ξ_i, since components beyond p do not contribute to the distance measure in the embedding. Thus, (33) becomes

$$\Delta w_j \sim \frac{\partial f((d(\xi_i, w_j))_{i,j})}{\partial d(\xi_i, w_j)} \cdot \left(\sum_l (\alpha_{jl} - \delta_{il}) \xi_l \right) \tag{34}$$

with Kronecker symbol δ_{il}. Hence we obtain the update

$$\Delta \alpha_{jl} \sim \begin{cases} \frac{\partial f((d(\xi_i, w_j))_{i,j})}{\partial d(\xi_i, w_j)} \cdot \alpha_{jl} & \text{if } l \neq i \\ \frac{\partial f((d(\xi_i, w_j))_{i,j})}{\partial d(\xi_i, w_j)} \cdot (\alpha_{jl} - 1) & \text{if } l = i \end{cases} \tag{35}$$

As an example, the update rule of RSLVQ (12) becomes

$$\Delta \alpha_{jl} \sim \begin{cases} -(P_y(j|\xi_i) - P(j|\xi_i))\alpha_{jl} & \text{if } l \neq i, c(w_j) = y_i \\ (P_y(j|\xi_i) - P(j|\xi_i))(1 - \alpha_{jl}) & \text{if } l = i, c(w_j) = y_i \\ P(j|\xi_i)\alpha_{jl} & \text{if } l \neq i, c(w_j) \neq y_i \\ -P(j|\xi_i)(1 - \alpha_{jl}) & \text{if } l = i, c(w_j) \neq y_i \end{cases} \tag{36}$$

We refer to this update rule as *kernel RSLVQ*, assumed similarities are used to compute the probabilities.

Note that this update constitutes a gradient technique only for Euclidean data. One can nevertheless apply it also for non-Euclidean settings, where the update step often at least improves the model since the positive parts of the pseudo-Euclidean space are usually dominant. Note that, again, normalization of the coefficients has to take place via direct normalization or barrier functions, for example.

(3) How are these update rules related? The most obvious difference of these two ways to update the coefficients consists in the fact that updates with respect to the coefficients follow a gradient technique whereas updates with respect to the weights, if done implicitly without explicit reference to the embedding, constitute a valid gradient method only if data are Euclidean.

But also in the Euclidean case where both updates follow a gradient technique, differences of the two update rules are observed. Prototypes depend linearly on the coefficients. Hence every local optimum of the cost

function with respect to the weights corresponds to a local optimum with respect to the coefficients and vice versa. As a consequence, the solutions which can be found by these two update rules coincide as regards the entire set of possible solutions provided the gradient techniques are designed in such a way that local optima are reached.

However, the single update steps of the two techniques are not identical, since taking the gradient does not commute with linear operations. Thus, it is possible that different local optima are reached in a single run even if the methods are started from the same initial condition.

Do mathematical guarantees such as the generalization ability transfer to the pseudo-Euclidean space?

We have already observed that one of the adaptation rules as proposed in the literature constitutes a valid gradient technique iff data are Euclidean. There are more severe reasons why it constitutes a desired property of the data to be Euclidean if referring to the theoretical motivation of the method in case of RSLVQ.

RSLVQ is derived as a likelihood ratio optimization technique. Since distances in pseudo-Euclidean space can become negative, a Gaussian distribution based on these distances does no longer constitute a valid probability distribution. Thus, Euclidean data are required to preserve the mathematical motivation of RSLVQ schemes as a likelihood optimization for (dis-)similarities.

GLVQ, in contrast, relies on the idea to optimize the hypothesis margin. Generalization bounds which depend on this hypothesis margin can be found which are based on the so-called Rademacher complexity of the function class induced by prototype based methods. Essentially, wide parts of the argumentation as given in [37] can be directly transferred to the pseudo-Euclidean setting (the situation might be more difficult if the rank of the space is not limited and Krein spaces come into the play.) The article [37] considers the more general setting where, in addition to adaptive prototypes, the quadratic form can be learned. Here, we consider the more simple function associated to GLVQ networks. Thereby, we restrict to the classification problems incorporating two classes 0 and 1 only, similar to [37].

In our case, the classification is based on the real-valued function

$$\xi \mapsto \left(\min_{w_i \,:\, c(w_i)=0} \|w_i - \xi\|_{pq}^2 - \min_{w_i \,:\, c(w_i)=1} \|w_i - \xi\|_{pq}^2 \right) \tag{37}$$

the sign of which determines the output class, and the size of which de-

termines the hypothesis margin. This function class is equivalent to the form

$$\xi \mapsto \left(\min_{w_i \,:\, c(w_i)=0} \left(\|w_i\|_{pq}^2 - 2\langle w_i, \xi \rangle_{pq} \right) - \min_{w_i \,:\, c(w_i)=1} \left(\|w_i\|_{pq}^2 - 2\langle w_i, \xi \rangle_{pq} \right) \right) \tag{38}$$

It is necessary to estimate the so-called Rademacher complexity of this function class relying on techniques as introduced e.g. in [1]. Since we do not refer to specifics of the definition of the Rademacher complexity, rather we refer to well-known structural results only, we do not introduce a precise definition at this place. Essentially, the complexity measures the amount of surprise in LVQ networks by taking the worst case correlation to random vectors.

As in [37] a structural decomposition can take place: this function can be decomposed into the linear combination of a composition of a Lipschitz-continuous function (min) and the function $\|w_i\|_{pq}^2 - 2\langle w_i, \xi \rangle_{pq}$. We can realize the bias $\|w_i\|_{pq}^2$ as additional weight if we enlarge every input vector by a constant component 1. Further, the sign of the components of w_i can be arbitrary, thus the signs in this bilinear form are not relevant and can be simulated by appropriate weights. Thus, we need to consider a linear function in the standard Euclidean vector space. As shown in [1] its Rademacher complexity can be bounded by a term which depends on the maximum Euclidean length of ξ and w_i and the square root of the number of samples for the evaluation of Rademacher complexity. The Euclidean lengths ξ and w_i can be limited in terms of the a priorly given similarities or dissimilarities: the vectorial representation of ξ corresponds to a column of $Q|\Lambda|^{1/2}$ with unitary Q and diagonal matrix of eigenvalues Λ. Thus, the Euclidean length of this vector is limited in terms of the largest eigenvalue of the similarity matrix S (or $\Psi(D)$). Since w_i is described as a convex combination, the same holds for w_i.

As a consequence, the argumentation of [37] can be transferred immediately to the given setting, i.e. large margin generalization bounds hold for GLVQ networks also in the pseudo-Euclidean setting. Since only the form of the classifier is relevant for this argumentation, but not the training technique itself, the same argumentation also holds for a classifier obtained using the RSLVQ cost function (11) in pseudo-Euclidean space, and it holds for both training schemes as introduced above.

How to enforce that data are Euclidean?

Albeit large margin generalization bounds transfer to the pseudo-Euclidean setting, it might be beneficial for the training prescription to transform data to obtain a purely Euclidean similarity or dissimilarity. How can data be transferred in such a way? There exist two prominent approaches, see e.g. [6, 27]:

1. *clip*: set all negative eigenvalues of the matrix Λ associated to the similarities to 0, i.e. use only the positive dimensions of the pseudo-Euclidean embedding. The corresponding matrix is referred to as Λ_{clip}. This preprocessing corresponds to the linear transformation $Q\Lambda_{\text{clip}}Q^t$ of the data. The assumption underlying this transformation is that negative eigenvalues are caused by noise in the data, and the given matrix should be substituted by the nearest positive semidefinite one.

2. *flip*: take $|\Lambda|$ instead of the matrix Λ, i.e. use the standard Euclidean norm in pseudo-Euclidean space. This corresponds to the linear transform $Q|\Lambda|Q^t$ of the similarity matrix. The motivation behind this procedure is the assumption that the negative directions contain relevant information. Hence the simple Euclidean norm is used instead of the pseudo-Euclidean one.

Since both corrections correspond to linear transformations, their out-of-sample extension is immediate. It has already been tested in the context of support vector machines (SVM) in [6] in how fare these preprocessing steps yield reasonable results, in some cases greatly enhancing the performance.

In summary, when extending LVQ schemes to general similarities or dissimilarities, we have the choices as described in Tab. 1. These yield to eight different possible combinations. In addition, we can further preprocess data into Euclidean form using e.g. clip or flip. Additional preprocessing steps have been summarized in [6], for example.

As explained above, the different design decision to arrive at LVQ for (dis-)similarities differ in the following sense:

- The cost functions of RSLVQ (11) and GLVQ (2) obey different principles, relevant differences being observable already in the Euclidean setting [36]. The motivation of RSLVQ as likelihood transfers to the Euclidean setting only, while large margin bounds of GLVQ can be transferred to the pseudo-Euclidean case.

cost function	data representation	training technique
GLVQ	similarity	gradient w.r.t. α_{jl}
RSLVQ	dissimilarity	gradient w.r.t. w_j

Table 1: Different possible choices when applying LVQ schemes for (dis-) similarity data

- When turning dissimilarities into dissimilarities and backwards, the identity is reached. When starting at similarities, however, data are centered using this transformation.
- Training can take place as gradient w.r.t the parameters α_{jl} or the prototypes w_j. The latter constitutes a valid gradient only if data are Euclidean, while the former follows a gradient also in the pseudo-Euclidean setting. In the Euclidean setting, the same set of local optima is valid for both methods, but the numerical update steps can be different resulting in different local optima in single runs.

We would like to point out that this argumentation is not restricted to LVQ schemes, rather it transfers to all prototype-based techniques provided that the model is based on distances of the form $D(\xi, w)$ and training takes place by an optimization of costs of the form $f(D(\xi, w))$. This holds also for the unsupervised prototype based clustering schemes neural gas and self-organizing maps (in the form as proposed by Heskes) [23, 16] and, indeed, an extension to kernel or relational approaches is possible [29, 41, 26], whereby the latter are often realized in form of batch updates which are particularly efficient for unsupervised scenarios due to closed form solutions of the respective optima [13, 5].

How to interpret prototypes for relational or kernel LVQ schemes?

One of the benefits of LVQ techniques consists in the fact that solutions are represented by a small number of representative prototypes which constitute members of the input space. In consequence, prototypes can be inspected in the same way as data in the vectorial setting. Since the dimensionality of points ξ is typically high, this inspection is often problem dependent: images, for example, lend itself to a direct visualization, oscillations can be addressed via sonification, spectra can be inspected as a graph which displays frequency versus intensity. Moreover, a low-dimensional projection of the data and prototypes by means of a nonlinear dimensionality reduction

technique offers the possibility to inspect the overall shape of the data set and classifier independent of the application domain.

Prototypes in relational or kernel settings correspond to positions in pseudo-Euclidean space which are representative for the classes if measured according to the given similarity/dissimilarity measure. Thus, prototype inspection faces two problems: (i) the pseudo-Euclidean embedding is usually only implicit, (ii) it is not clear whether dimensions in this embedding carry any semantic information. Thus, albeit prototypes are represented as linear combinations of data also in the pseudo-Euclidean setting, it is not clear whether these linear combinations correspond to a semantic meaning.

One approach which is taken in this context is to approximate a prototype by one or several exemplars, i.e. members of the data set, which are close by [17]. Thereby, the approximation can be improved if sparsity constraints for the prototypes are integrated while training. This way, every prototype is represented by a small number of exemplars which can be inspected like data. Another possibility is to visualize data and prototypes using some nonlinear dimensionality reduction technique. This enables an investigation of the overall shape of the classifier just as in the standard vectorial setting. Naturally, both techniques, a representation of prototypes by few exemplars as well as a projection to low dimensions incorporate errors depending on the dimensionality of the pseudo-Euclidean space and its deviation from the Euclidean norm.

4 Experiments

We test the various LVQ variants including two novel techniques which arise from these different combinations: relational RSLVQ trained using dissimilarities and gradients w.r.t. α_{jl} and kernel RSLVQ trained based on similarities and gradients w.r.t. w_j. In the literature, the corresponding settings for GLVQ can be found [30, 15]. We compare the methods to the support vector machine (SVM) and a k-nearest neighbor classifier (k-NN) on a variety of benchmarks as introduced in [6].

The data sets represent a variety of similarity matrices which are, in general, non-Euclidean. It is standard to symmetrize the matrices by taking the average of the matrix and its transposed. Further, the substitution of a given similarity by its normalized variant constitutes a standard preprocessing step, arriving at diagonal entries 1. Even in symmetrized and normalized form, the matrices do not necessarily provide a valid kernel. Hence, we also test the preprocessing steps clip and flip in comparison to a direct applica-

tion of the methods for the original data. We also report the signatures of the data whereby a cutoff at 0.0001 is made to account for numerical errors of the eigenvalue solver. We also report the number of used prototypes, which is chosen as a small multiple of the number of classes. Typically, overfitting does hardly occur in LVQ settings such that the sensitivity of the result on the number of prototypes is low, provided a sufficient flexibility is present.

- *Amazon47*: This data set consists of 204 books written by four different authors. The similarity is determined as the percentage of customers who purchase book j after looking at book i. This matrix is fairly sparse and mildly non-Euclidean with signature $(192, 1, 11)$. Class labeling of a book is given by the author. The number of prototypes which is chosen in all LVQ settings is 94.

- *Aural Sonar*: This data set consists of 100 wide band solar signals corresponding to two classes, observations of interest versus clutter. Similarities are determined based on human perception, averaging over 5 random probands for each signal pair. The signature is $(61, 38, 1)$. Class labeling is given by the two classes: target of interest versus clutter. The number of prototypes chosen in LVQ scenarios is 10.

- *Face Rec*: 945 images of faces of 139 different persons are recorded. Images are compared using the cosine-distance of integral invariant signatures based on surface curves of the 3D faces. The signature is given by $(45, 0, 900)$. The labeling corresponds to the 139 different persons. The number of prototypes is 139.

- *Patrol*: 241 samples representing persons in seven different patrol units are contained in this data set. Similarities are based on responses of persons in the units about other members of their groups. The signature is $(54, 66, 121)$. Class labeling corresponds to the seven patrol units. The number of prototypes is 24.

- *Protein*: 213 proteins are compared based on evolutionary distances comprising four different classes according to different globin families. The signature is $(169, 38, 6)$. Labeling is given by four classes corresponding to globin families. The number of prototypes is 20.

- *Voting*: Voting contains 435 samples with categorical data compared by means of the value difference metric. Class labeling into two classes

is present. The signature is $(16, 1, 418)$. The number of prototypes is 20.

- *Sonatas:* The sonatas data set contains complex symbolic data similar to [24]. It contains dissimilarities between 1,068 sonatas from the classical period (Beethoven, Mozart and Haydn) and the baroque era (Scarlatti and Bach). The data are in the MIDI file format, taken from the online MIDI collection Kunst der Fuge[1]. Their mutual dissimilarities are measured with the normalized compression distance (NCD), see [7], which is applied to a a specific preprocessing, which integrates invariances for music information retrieval, see [24]. The musical pieces are classified according to their composer. The signature of the data is $(1063, 4, 1)$. The number of prototypes for LVQ schemes is 5.

- *Chromosomes*: The Copenhagen chromosomes data set has been introduced in [21] as a benchmark for cytogenetics. 4,200 human chromosomes from 22 classes (the autosomal chromosomes) are given by grey-valued images. The images can be represented as strings measuring the thickness of the silhouettes of the chromosomes. These strings are compared using edit distance with insertion/deletion costs 4.5 [25]. The signature is $(1951, 2206, 43)$. The number of prototypes for LVQ schemes is 21.

Note that the data sets Voting, FaceRec, Sonatas, and Amazon 47 are almost Euclidean, while all others have a substantial part of negative eigenvalues.

For some of these data sets, results for the SVM and a k-NN classifier have been reported in [6]. Thereby, data are preprocessed using clip or flip to guarantee positive definiteness for SVM, if necessary. The latter is used with the RBF kernel and optimized meta-parameters in [6]. For multi-class classification, the one versus one scheme has been used.

In comparison, we train kernel and relational RSLVQ networks using the real data or its clip or flip, respectively.

Results of a 20-fold cross-validation with the same partitioning as proposed in [6] are reported. Prototypes are initialized by means of normalized random coefficients α_{jl} where the prior class label $c(w_l)$ determines the non-zero elements. Meta-parameters are optimized on the data sets using cross-validation. Further, while training, we guarantee that prototypes are contained in the convex hull of the data by setting negative coefficients to zero after every adaptation step and adding a normalization of the vector α_i to 1 after every adaptation step.

[1] www.kunstderfuge.com

	k-NN		SVM		KGLVQ		RGLVQ		KRSLVQ		RRSLVQ	
Amayon47	28.54	(0.83)	21.46	(5.74)	22.80	(5.38)	18.17	(5.39)	**15.37**	(0.36)	22.44	(5.16)
clip	28.78	(0.74)	21.22	(5.49)	21.95	(5.65)	23.78	(7.20)	**15.37**	(0.41)	25.98	(7.48)
flip	28.90	(0.68)	22.07	(6.25)	23.17	(6.10)	20.85	(4.58)	16.34	(0.42)	22.80	(4.96)
Aural Sonar	14.75	(0.49)	12.25	(7.16)	13.00	(7.70)	13.50	(5.87)	11.50	(0.37)	13.00	(7.50)
clip	17.00	(0.51)	12.00	(5.94)	14.50	(8.30)	13.00	(6.96)	**11.25**	(0.39)	13.25	(7.12)
flip	17.00	(0.93)	12.25	(6.97)	12.30	(5.50)	13.00	(6.96)	11.75	(0.35)	13.50	(7.63)
Face Rec	7.46	(0.04)	3.73	(1.32)	**3.35**	(1.29)	3.47	(1.33)	3.78	(0.02)	7.50	(1.49)
clip	7.35	(0.04)	3.84	(1.16)	3.70	(1.35)	3.81	(1.67)	3.84	(0.02)	7.08	(1.62)
flip	7.78	(0.04)	3.89	(1.19)	3.63	(1.16)	3.78	(1.48)	3.60	(0.02)	7.67	(2.21)
Patrol	22.71	(0.33)	15.52	(4.02)	11.67	(4.60)	18.02	(4.65)	17.50	(0.25)	17.71	(4.24)
clip	9.90	(0.16)	13.85	(4.39)	**8.96**	(3.90)	17.29	(3.45)	17.40	(0.29)	21.77	(7.10)
flip	10.31	(0.16)	12.92	(5.09)	9.74	(4.90)	18.23	(5.10)	19.48	(0.34)	20.94	(4.51)
Protein	51.28	(0.77)	30.93	(6.79)	27.79	(7.60)	28.72	(5.24)	26.98	(0.37)	5.58	(3.49)
clip	25.00	(0.74)	12.56	(5.46)	1.63	(2.10)	12.79	(5.36)	4.88	(0.17)	11.51	(5.03)
flip	7.79	(0.18)	1.98	(2.85)	12.33	(6.10)	3.49	(3.42)	**1.40**	(0.05)	4.42	(3.77)
Voting	5.00	(0.01)	5.06	(1.84)	6.55	(1.90)	9.14	(2.10)	5.46	(0.04)	11.26	(2.23)
clip	4.83	(0.02)	5.00	(1.84)	6.55	(1.90)	9.37	(2.02)	5.34	(0.04)	11.32	(2.31)
flip	**4.66**	(0.02)	4.89	(1.78)	6.49	(1.90)	9.14	(2.22)	5.34	(0.03)	11.26	(2.43)
Sonatas	11.52	(0.20)	13.11	(2.82)	39.35	(3.17)	16.11	(3.77)	16.01	(0.11)	21.54	(4.38)
clip	**10.49**	(0.19)	10.86	(3.74)	37.04	(4.54)	16.20	(3.73)	15.82	(0.17)	23.23	(4.41)
flip	10.58	(0.25)	11.06	(3.68)	37.04	(6.41)	16.29	(3.63)	15.07	(0.12)	21.91	(4.41)
Chromosom	3.93	(0.01)	2.93	(1.07)	5.48	(0.34)	8.31	(1.84)	7.14	(0.02)	38.51	(3.61)
clip	3.81	(0.01)	2.81	(1.14)	4.76	(0.00)	8.50	(1.71)	6.86	(0.02)	36.67	(3.63)
flip	3.79	(0.01)	**2.45**	(0.98)	4.29	(0.00)	8.38	(1.76)	7.52	(0.01)	34.76	(3.20)

Table 2: Misclassification error of different dissimilarity classifiers for benchmark data as reported in [6]. Standard deviations are given in parenthesis.

The results obtained on these data sets are reported in Tab. 2. For k-NN, the best result for $k \in \{1, 3, 5\}$ is reported.

Due to its almost Euclidean nature, preprocessing by clip and flip has hardly an effect for Amazon47, FaceRec, Sonatas, and Voting. For the data sets Patrol and Protein, flip and clip change the similarity severely, as can be spotted by the change of the k-NN error. Albeit all other data sets also display a considerable non-Euclideanity as can be seen by the spectrum, flip or clip do have a minor effect on these data only, resulting up to 3% change of the classification accuracy. Note that it depends very much on the data set and the used technique, which preprocessing yields best results. In general, SVM can show instabilities for non pdf data because some numeric schemes used for parameter optimization in SVM built on pdf similarity matrices. Unless data are Euclidean, where preprocessing using clip or flip has no effect, it is not clear a priori which technique is best, and it can happen that the best preprocessing also depends on the different learning algorithms (as can be seen for the Patrol data).

Interestingly, for all data sets, one or several of the kernel or relational LVQ techniques display a quality which is at least competitive to (if not better than) k-NN and SVM on the data set or an appropriate preprocessing. There are a few interesting outliers when comparing the different LVQ techniques: relational RSLVQ yields to a classification error for Chromosomes which is an order of magnitude higher than for the remaining techniques. This is possibly due to a sensitive choice of the bandwidth σ in this large data set. Similarly, kernel GLVQ yields more than 50% resp. close to 40% error for the FaceRec data or Sonatas, respectively, corresponding possibly to a local optimum in this case with large basin of attraction. Overall, both, relational GLVQ and kernel RSLVQ yield constantly good classification accuracy.

Computational complexity

Note that the computational complexity of LVQ for similarities or dissimilarities increases as compared to vectorial LVQ schemes: the space complexity for prototype storage becomes $\mathcal{O}(N)$, N being the number of data points, assuming a fixed number of prototypes m. The time complexity is dominated by a matrix multiplication in every adaptation step to compute the dissimilarity which is of order $\mathcal{O}(N^2)$. For SVM, depending on the implementation, space and time complexity are similar, the number of support vectors being usually a fraction of the training set, and training having worst case complexity $\mathcal{O}(N^3)$ unless speed-ups e.g. via SMO or core techniques are

used.

For the largest data set which we investigated (Chromosomes with 4.200 points), training one LVQ network for similarities or dissimilarities took approximately 15 minutes using a Matlab implementation on a powerful desktop computer (two six-core Xeon-processors, clocked at 3.5GHz). With current desktop computers, usually the storage of the distance matrix constitutes the main bottleneck concerning space (with approximately 30.000 data points just fitting into current standard memory of 8 GB) - albeit the final classifier requires linear space only, the matrix required to represent the training data is quadratic. Such data sets would require about a day training time.

Thus, one the limit lies at about 30.000 data points when training the proposes LVQ techniques with current desktop computers. Similar to approximation techniques for kernel methods, however, one can use approximation techniques such as the Nyström method to decrease the complexity to a linear size training set and training time, sparse approximation on the α-vectors to limit memory consumption and simplify the interpretation of the final prototypes, or by active learning strategies using the margin criterion to speed-up the learning for the online algorithms [12, 34, 35]. This can extend the size of feasible data sets by one to two orders of magnitude.

5 Discussion

We have discussed different possibilities to extend LVQ classifiers to similarity or dissimilarity data, thereby specifying different ways how to choose the cost function, how to include the data representation, how to train the mapping, and its mutual differences. This general treatment covers several existing approaches in the literature and suggests a couple of new combinations. By relying on pseudo-Euclidean embeddings, we have developed a general approach how to smoothly adapt prototypes in this setting, resulting in a well defined method for most of the techniques. Interestingly, while generalization bounds can be transferred to the pseudo-Euclidean setting, a probabilistic interpretation is lost in general. Similarly, gradients with respect to prototypes result in valid gradient techniques only in Euclidean settings, gradients with respect to coefficients can always be determined. We have evaluated the technique in a variety of benchmarks, showing surprisingly good classification results in several settings, albeit the underlying theory does not necessarily support this specific design choice as the one with most theoretical guarantees. A more specific investigation in how far

non-Euclidean data are present in such situations leading e.g. to possibly negative dissimilarities as regards prototypes will be the subject of future work.

Acknowledgement

This work has been supported by the DFG under grants number HA2719/6-1 and HA2719/7-1 and by the CITEC center of excellence. Further, this research and development project is funded by the German Federal Ministry of Education and Research (BMBF) within the Leading-Edge Cluster Competition and managed by the Project Management Agency Karlsruhe (PTKA). The authors are responsible for the contents of this publication.

References

[1] P. L. Bartlett, S. Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *Journal of Machine Learning Research*, 3:463–4982, 2002.

[2] M. Biehl, A. Ghosh, and B. Hammer. Dynamics and generalization ability of LVQ algorithms. *Journal of Machine Learning Research*, 8:323–360, 2007.

[3] M. Biehl, B. Hammer, M. Verleysen, and T. Villmann, editors. *Similarity Based Clustering.* Springer Lecture Notes Artificial Intelligence Vol. 5400/2009. Springer, 2009.

[4] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[5] R. Boulet, B. Jouve, F. Rossi, and N. Villa. Batch kernel SOM and related Laplacian methods for social network analysis. *Neurocomputing* 71(7-9):1257–1273. 2008.

[6] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. Similarity-based classification: Concepts and algorithms. *JMLR*, 10:747–776, June 2009.

[7] R. Cilibrasi and M. B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory* 51(4):1523–1545, 2005.

[8] M. Cottrell, B. Hammer, A. Hasenfuss, and T. Villmann. Batch and median neural gas. *Neural Networks*, 19:762–771, 2006.

[9] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE TNN*, 9(5):768–786, 1998.

[10] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[11] T. Gärtner. *Kernels for Structured Data*. PhD thesis, Univ. Bonn, 2005.

[12] A. Gisbrecht, B. Mokbel, F.-M. Schleif, X. Zhu, and B. Hammer. Linear Time Relational Prototype Based Learning. *Int. J. Neural Syst*, 22(5), 2012.

[13] B. Hammer and A. Hasenfuss. Topographic mapping of large dissimilarity datasets. *Neural Computation*, 22(9):2229–2284, 2010.

[14] B. Hammer, A. Micheli, and A. Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17:1109–1159, 2005.

[15] B. Hammer, B. Mokbel, F.-M. Schleif, and X. Zhu. Prototype based classification of dissimilarity data. *IDA*, 2011.

[16] T. Heskes. Energy Functions for Self-Organizing Maps. In Oja, Erkki; and Kaski, Samuel (Eds.), Kohonen Maps, Elsevier, 1999.

[17] D. Hofmann and B. Hammer. Sparse approximations for kernel learning vector quantization. *ESANN*, 2013.

[18] P. J. Ingram, M. P. H. Stumpf, and J. Stark. Network motifs: structure does not determine function. *BMC Genomics* 7, 108, 2006.

[19] T. Kohonen. *Self-Oganizing Maps*. Springer, 3rd edition, 2000.

[20] T. Kohonen and P. Somervuo. How to make large self-organizing maps for nonvectorial data. *Neural Networks* 15(8-9): 945–952. 2002.

[21] C. Lundsteen, J. Phillip, and E. Granum. Quantitative analysis of 6985 digitized trypsin g-banded human metaphase chromosomes. *Clinical Genetics* 18(5):355–370, 1980.

[22] T. Maier, S. Klebel, U. Renner, and M. Kostrzewa. Fast and reliable maldi-tof ms–based microorganism identification. *Nature Methods*, 3, 2006.

[23] T. Martinetz, S. Berkovich, and K. Schulten. "Neural-gas" Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE-Transactions on Neural Networks* 4(4):558–569, 1993.

[24] B. Mokbel, A. Hasenfuss, and B. Hammer Graph-Based Representation of Symbolic Musical Data. *GbRPR* 42–51, 2009.

[25] M. Neuhaus and H. Bunke. Edit distance based kernel functions for structural pattern classification. *Pattern Recognition* 39(10):1852–1863, 2006.

[26] M. Olteanu, N. Villa-Vialaneix, and M. Cottrell. On-line relational SOM for dissimilarity data. *CoRR* abs/1212.6316 (2012).

[27] E. Pekalska and R. P. Duin. *The Dissimilarity Representation for Pattern Recognition. Foundations and Applications*. World Scientific, 2005.

[28] O. Penner, P. Grassberger, and M. Paczuski. Sequence Alignment, Mutual Information, and Dissimilarity Measures for Constructing Phylogenies. *PLOS ONE* 6(1), 2011.

[29] A. K. Qin and P. N. Suganthan. Kernel neural gas algorithms with application to cluster analysis. In *Proc. of the 17th International Conference on Pattern Recognition*, (ICPR '04), 617–620, 2004.

[30] A. K. Qin and P. N. Suganthan. A novel kernel prototype-based learning algorithm. In *Proc. of the 17th International Conference on Pattern Recognition* (ICPR '04), 2004.

[31] F. Rossi and N. Villa-Vialaneix. Consistency of functional learning methods based on derivatives. *Pat. Rec. Letters*, 32(8):1197–1209, 2011.

[32] A. Sato and K. Yamada. Generalized Learning Vector Quantization. In *NIPS* 1995.

[33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. Computational capabilities of graph neural networks. *IEEE TNN*, 20(1):81–102, 2009.

[34] F.-M. Schleif, B. Hammer, and T. Villmann. Margin-based active learning for LVQ networks. *Neurocomputing* 70(7-9): 1215–1224, 2007.

[35] F.-M. Schleif, T. Villmann, B. Hammer, and P. Schneider. Efficient Kernelized Prototype Based Classification. *Int. J. Neural Syst.* 21(6): 443–457, 2011.

[36] P. Schneider, M. Biehl, and B. Hammer. Distance learning in discriminative vector quantization. *Neural Computation*, 21:2942–2969, 2009.

[37] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21:3532–3561, 2009.

[38] S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Comput.*, 15:1589–1604, 2003.

[39] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-SNE. *JMLR*, 9:2579–2605, 2008.

[40] A. Vellido, J.D. Martin-Guerroro, and P. Lisboa. Making machine learning models interpretable. In *ESANN'12*. 2012.

[41] Hujun Yin: On the equivalence between kernel self-organising maps and self-organising mixture density networks. Neural Networks 19(6-7): 780-784 (2006).