University of Groningen

Median Variants of Prototype Based Learning Vector Quantization

Nebel, David

DOI:
[10.33612/diss.135377546](https://doi.org/10.33612/diss.135377546)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2020

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):
Nebel, D. (2020). *Median Variants of Prototype Based Learning Vector Quantization: Methods for Classification of General Proximity Data*. [Thesis fully internal (DIV), University of Groningen]. University of Groningen. <https://doi.org/10.33612/diss.135377546>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Median Variants of Prototype Based Learning Vector Quantization - Methods for Classification of General Proximity Data

David Nebel

university of
 groningen

Median Variants of Prototype Based Learning Vector Quantization

Methods for Classification of General Proximity Data

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. E. Sterken
and in accordance by the College Deans.

This thesis will be defend in public on

Monday 19 October 2020 at 11.00 hours

by

David Nebel

born on 16 May 1980
in Karl-Marx-Stadt (now named Chemnitz), Germany

Supervisors

Prof. M. Biehl

Prof. T. Villmann

Assessment committee

Prof. N. Petkov

Prof. T. M. Heskes

Prof. M. Opper

Contents

Acknowledgments	5
Abbreviations	7
1. Introduction	11
1.1. Outline	13
2. Prototype Based Methods for Classification	17
2.1. Unsupervised Vector Quantization	17
2.2. Supervised Vector Quantization	19
2.2.1. Generalized Learning Vector Quantization	23
2.2.2. Robust Soft Learning Vector Quantization	26
2.2.3. Soft Nearest Prototype Classification	28
2.2.4. Support Vector Machines	30
3. Proximity Measures for Prototype Based Models	35
3.1. Types of Proximities	37
3.1.1. Similarities and Dissimilarities	37
3.1.2. Inner Products and Kernel Functions	42
3.2. Equivalence of Proximities	51
3.3. Handling of Proximity Data in nearest neighbor approaches	61
3.3.1. General Aspects and Motivating Example	61
3.3.2. Conversion of Proximity Matrices	63
3.3.3. Preprocessing for non-psd Kernels	67
3.4. Classification Algorithms for Proximity Data	71

4. Expectation Maximization Principle for Model Optimization Applicable for General Proximity Measures	77
4.1. Expectation Maximization for probabilistic models	77
4.1.1. The Gaussian Mixture Model (GMM)	77
4.1.2. Maximizing Mixture Models	85
4.2. Expectation Maximization for vector quantization	95
4.2.1. Jensen Inequality	95
4.2.2. Generalization of Expectation Maximization	96
4.2.3. Expectation Maximization for Neural Gas	99
5. Median Algorithms for Supervised Vector Quantization Using General Proximity Measures	111
5.1. The General Scheme for Supervised Vector Quantization Optimizing by Means of Generalized EM	111
5.2. Median Variants for cost function based LVQ Classifiers	114
5.2.1. Median Robust Soft LVQ (mRSLVQ)	114
5.2.2. Median Generalized LVQ (mGLVQ)	118
5.2.3. Median Nearest Prototype Classification (mNPC)	126
5.2.4. Experiments	130
5.3. Median Variants for Optimizing Statistical Classification Evaluation Measures behind Accuracy	134
5.4. Mixtures of Proximities	145
6. Summary and Conclusion	157
Nederlandse samenvatting	161
A. Appendix	165
A.1. Decomposition Proof	166
A.2. Proof for $\mathcal{R}(\mathcal{X}, \Theta) = 0$	167
A.3. Derivation for prototype update (Neural Gas)	168
Publications	169
Bibliography	174

Acknowledgments

The last few years, spend developing this work, have been very exciting. I have learned a lot about myself, pushed my own limits, improved my resistance to frustration and found many friends. I am grateful for the memorable times and all those people who have made this possible. Now, at this milestone of my journey, it is a good moment to look back and give a thanks to all who supported me. Thanks to:

-Angelika and Ronja-

for unrestricted support whenever I needed it.

-Prof. Thomas Villmann-

for the chance to do this work and for all the scientific fights.

-Prof. Michael Biehl-

for his patience and support including the survival of the Dutch
bureaucracy.

-Prof. Barbara Hammer and Andrej Gisbrecht-

for the helpful ideas and discussions especially at the beginning of my
work.

-Martin and Sven-

for being very close friends during our college years. Not only the day
of the defense will not be the same without Martin.

-Marika-

for hunting every single squirrel. I am sure it was not a waste of time.

-Tina, Mandy, Andrea, Lydia, Paul, Jensun, ...-

and the whole former and actual members of the CI Group for ideas,
discussion and having fun.

-my family and all my friends-

for believing in me when I did not. To list out all their names here
would double my printing fee.

-Michiel Straat-

for quick response and support with the summary translation.

Essential Abbreviations and frequently used Variables

$A(:, j)$		j-th column of Matrix A
$A(i, \cdot)$		i-th row of Matrix A
$ a $		Absolute value of a , if a is a real or complex number
$ A $		Number of elements in A , if A is a set
$ A $		Determinant of A , if A is a Matrix
c_i		Class label
$c(\cdot)$		Class label function
$\mathcal{C}(\mathcal{X}, \Theta)$		Objective or cost function
$d(\cdot, \cdot), \mathfrak{d}(\cdot, \cdot)$	...	Dissimilarity measure
$\mathbf{D}$		Dissimilarity matrix
$f_a(\cdot)$		Activation function in GLVQ
$g_j(x_i), g_{ij}$		formal probability score
$\hat{g}_{ij}$		Optimal value for the formal probability scores
$\mathcal{G}(x_i, \theta_j)$		Local cost function
$h(x)$		Hypothesis margin
$\mathbf{H}(\cdot)$		Heaviside function
$\mathfrak{K}(\cdot, \cdot)$		Kernel function
$\mathbf{K}$		Kernel matrix
$\mathcal{K}(g p)$		Kullback-Leibler-divergence

$\mathcal{L}(g), \mathcal{L}(\mathcal{X}, \Theta)$	Lower Bound (of the cost function decomposition)
n_p	Number of prototypes
n_d	Number of data points
n_c	Number of clusters/components
$\mathcal{N}(x \mu, \Sigma)$	Gaussian Distribution with mean μ and Covariance Σ
$\mathfrak{p}(\cdot, \cdot)$	General proximity measure
$\mathbf{P}$	Proximity matrix
$P(\mathcal{X})$	Data distribution
$\mathbf{R}$	Proximity rank matrix
$\mathbb{R}^D$	D dimensional real vector space
$\mathcal{R}(\mathcal{X}, \Theta)$	Remainder function (of the cost function decomposition)
$s(x)$	Index of the nearest prototype for data point x
$\text{sig}(\cdot)$	Sigmoid function
$s(\cdot, \cdot), \mathfrak{s}(\cdot, \cdot)$	Similarity measure
$\mathbf{S}$	Similarity matrix
x	Data points
$\mathcal{X}$	Data set
$\mathcal{Z}$	Matrix of hidden variables in EM
$\gamma_j(x_i), \gamma_{ij}$	Responsibilities
δ	Kronecker Delta

$\tilde{\vartheta}$	Absolute rank-equivalence measure
ϑ	Normalized rank-equivalence measure
θ	Prototypes
Θ	Prototype set
$\kappa(x)$	Classifier function in GLVQ
π_k	Mixing coefficient for mixture models
$\rho(\cdot, \cdot)$	Rank function
$\varrho(\Theta)$	Prototype distribution
$\langle \cdot, \cdot \rangle$	Inner product
$\stackrel{\cdot}{=}_{\mathcal{X}, \theta}$	Rank equivalence
ARE	Absolute Rank-equivalence Measure
ARP	Attraction-Repulsion-Principle
EM	Expectation Maximization
GLVQ	Generalized Learning Vector Quantization
GMM	Gaussian Mixture Model
iid	independent and identically distributed
IIP	Indefinite Inner Product
IIPS	Indefinite Inner Product Space
IP	Inner Product
k-ARE	k Absolute Rank-equivalence Measure

k-NN		k-Nearest Neighbor
LVQ		Learning Vector Quantization
ML		Machine Learning
NG		Neural Gas
NPC		Nearest Prototype Classification
NRE		Normalized Rank-equivalence Measure
pd		positive definite
PRM		Proximity Rank Matrix
psd		positive semidefinite
RSLVQ		Robust Soft Learning Vector Quantization
SGAL		Stochastic Gradient Ascent Learning
SGDL		Stochastic Gradient Descent Learning
SIP		Semi-Inner Product
SIPS		Semi-Inner Product Space
SNPC		Soft Nearest Prototype Classification
SOM		Self Organizing Maps
SVM		Support Vector Machines
VQ		Vector Quantization

1. Introduction

The amount of digital data increases every year dramatically. For 2025 a data volume of 175 zettabyte ($175 \cdot 10^{21}$ byte) is predicted [1]. The processing of these data requires improved strategies, methods and algorithms for data compression, data visualization and general data processing. Among others the goals of these methods is to condense the information inside, to extract knowledge and to store data efficiently. Beside traditional statistical methods and approaches in pattern recognition more and more machine learning applications are used for those data processing tasks. Particularly, their ability for adaptive data processing, for knowledge extraction and their generalization ability are highly appreciated features in context of big data. Yet, although frequently taken as alternatives to statistical data analysis, machine learning approaches include statistical ideas and methodologies. Further, probabilistic approaches in machine learning can be seen as advanced statistical tools. Their ability to adapt distinguish them from traditional probabilistic methods.

Generally, machine learning algorithms can be partitioned into supervised learning, unsupervised learning and learning with delayed rewards. Supervised approaches are used for classification and regression prob-

lems whereas unsupervised methods comprise algorithms for data clustering/compression and visualization. Learning with delayed rewards is learning by minimal information and, therefore, between supervised and unsupervised learning, frequently applied in control problems.

Famous ML-methods for clustering and data compression are (Fuzzy-) C-means and variants thereof [2], [3], neural vector quantizers [4], affinity propagation [5] and Gaussian mixture models [6]. For visualization t-SNE [7] and SOM [8] gained a lot of attraction. Supervised learning is nowadays dominated by (deep) multilayer perceptrons. Other well known approaches are support vector machines and Bayes networks.

Although, these methods often deliver promising results, most of them are handled as black box tools because they are difficult to interpret. A paradigm to avoid or to reduce at least this black box behavior is the prototype principle. Roughly speaking those methods distribute references (prototypes) to estimate and approximate the data distribution. Prototype based methods were developed for both, supervised and unsupervised learning. Well known examples are the above mentioned vector quantizers for data compression and Kohonens learning vector quantizers [9] for classification learning. Additionally to their easy interpretability prototype based methods seem to be robust according to small variations in data [10]. An important ingredient for prototype based methods is that the data as well as the prototypes are compared in terms of a dissimilarity or similarity measure in contradiction to SVM [11] and MLP [12] which are based on inner products determining the decision hyperplanes. Variants of learning vector quantizers for classification learning are the

main focus of this thesis. Particularly we will address several problems related to them:

- First, we consider (dis)-similarities in detail and provide a general taxonomy of those measures in accordance to their mathematical properties. Subsequently we introduce a tool for the comparison of those measures with respect to their behavior in particular classification problems. For a given task, it provides a recommendation regarding a suitable (dis)-similarity measure.
- We generalize the expectation maximization approach for the probabilistic Gaussian mixture models using an analytical interpretation of the model. This description is transferred to non-probabilistic (deterministic) models resulting in a generalized EM scheme.
- The developed gEM methodology is used to extend the probabilistic robust soft learning vector quantizer for the use case of discrete data. Thereafter several variants of this scheme are considered which differ according to their specific classification performance evaluation as well as their learning paradigms.

1.1. Outline

Chapter 2 introduces prototype based methods. Particularly, supervised algorithms are in the focus.

Chapter 3 includes the introduction of the general concept of proximities and a discussion with focus on their usage for vector quantization.

For this purpose, we clarify the taxonomy of dissimilarities, similarities and inner products regarding their properties as an extension of Duin&Pekalskas work [13]. Furthermore we introduce a comparison concept which helps us to measure the differences of various (dis)-similarities and inner products regarding their topological behavior, which allows to construct a respective tool for comparison.

Chapter 4 deals with the EM-principle applied to GMM and LVQ-models for vector quantization. Particularly, we introduce a modification of the well known EM-principle to tackle median variants of LVQ for discrete data. This scheme is denoted as gEM. For motivation purpose, we first concentrate on the well known Gaussian mixture model in relation to RSLVQ. As an interesting application example a batch variant of neural gas based on gEM-optimization is derived.

Chapter 5 transfers the concepts developed in chapter 4 to LVQ-based classifier models. Additionally we show, how those models can be extended by task dependent (dis)-similarity learning taking into account the developed evaluation measure for (dis)-similarity comparison.

This thesis is mainly based on

- [1] D. Nebel, B. Hammer, K. Frohberg and T. Villmann. Median variants of learning vector quantization for learning of dissimilarity data. *Neurocomputing*, 169, (2015). pp. 295-305.

-
- [2] D. Nebel, M. Kaden, A. Villmann and T. Villmann. Types of (dis)-similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, 268 (2017): pp. 42-54.
 - [3] D. Nebel, B. Hammer and T. Villmann. A median variant of generalized learning vector quantization. *International Conference on Neural Information Processing*. Springer, Berlin, Heidelberg, (2013). pp. 19-26.
 - [4] D. Nebel, B. Hammer and T. Villmann. Supervised Generative Models for Learning Dissimilarity Data. *ESANN, 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2014). pp. 35-40.
 - [5] D. Nebel and T. Villmann. Median-LVQ for classification of dissimilarity data based on ROC-optimization. *ESANN, 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2015). pp. 1-6.
 - [6] D. Nebel and T. Villmann. Optimization of Statistical Evaluation Measures for Classification by Median Learning Vector Quantization. *Advances in Self-Organizing Maps and Learning Vector Quantization*. Springer, Cham, (2016). pp. 281-291.

- [7] M. Kaden, D. Nebel and T. Villmann. Adaptive dissimilarity weighting for prototype-based classification optimizing mixtures of dissimilarities. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2016). pp. 135-140

- [8] T. Villmann, M. Kaden, D. Nebel and A. Bohnsack. Similarities, dissimilarities and types of inner products for data analysis in the context of machine learning. *International Conference on Artificial Intelligence and Soft Computing*. Springer, Cham, (2016). pp. 125-130

- [9] M. Kaden, D. Nebel, F. Melchert, A. Backhaus, U. Seiffert and T. Villmann. Data dependent evaluation of dissimilarities in nearest prototype vector quantizers regarding their discriminating abilities. *Advances in Self-Organizing Maps and Learning Vector Quantization*. Springer, Cham, (2017). pp. 1-7.

2. Prototype Based Methods for Classification

Roughly speaking prototype based methods can be divided into two general classes. The first class, the unsupervised learning paradigm, is often referred as Vector Quantization (VQ) where the second one, the supervised learning paradigm, is often referred as Learning Vector Quantization (LVQ). The main difference between both ideas regarding to the given data is that in the first case the data is unlabeled whereby in the second case each data point is assigned to a class label out of a finite set of classes. There also exist a third class of methods which are referred to semi-supervised VQ which can be seen as mixture of the first two types. In this case, we have labeled as well as unlabeled data as input. The unsupervised scenario will play only a minor role in this thesis. Therefore, we give only a rough description of the idea. For the supervised case, we give a more detailed description because this focus will play a key role in this thesis.

2.1. Unsupervised Vector Quantization

Assume that we have given n_d unlabeled data points $x_i \in \mathcal{X} \subseteq \mathbb{R}^D$ ($i = 1, \dots, n_d$) from a D-dimensional real vector space whereby the data

points are realizations of an unknown underlying data distribution $P(\mathcal{X})$. Beside other goals the main idea of VQ is to describe the data points by a set of n_p prototypes $\theta_j \in \Theta \subseteq \mathbb{R}^D$ ($j = 1, \dots, n_p$) such that $n_p \ll n_d$ holds and the prototype distribution $\varrho(\Theta)$ approximates the data distribution $P(\mathcal{X}) \approx \varrho(\Theta)$, i.e. we want to find a discrete approximation of the data distribution by a set of prototypes. The principle can be obtained by minimizing the reconstruction (quantization) error

$$E = \int P(x) d(x, \theta_{s(x)}) dx$$

where $d(\cdot, \cdot)$ is a dissimilarity measure, in the easiest case the squared Euclidean distance, and $\theta_{s(x)}$ is the winning prototype. The winning prototype for a data point x is the closest prototype

$$s(x) = \arg \min_j d(x, \theta_j). \quad (2.1)$$

The more general case is the generalized distortion error

$$E_\gamma = \int P(x) (d(x, \theta_{s(x)}))^\gamma dx.$$

Taking $d(x, \theta_{s(x)})$ as Euclidean distance and optimizing the general error we get for the achieved prototype distribution that $P(\mathcal{X}) \approx \varrho(\Theta)^\alpha$ holds with $\alpha = \frac{d}{d+\gamma}$. Thereby d is the intrinsic (Hausdorff) dimension of the data [14], [15].

There are several VQ methods like k-Means [2], Self Organizing Maps (SOM) [8], [16] and Neural Gas (NG) [17] which differs mainly in the cooperation of prototypes during the learning process.

The outcome of a VQ model i.e. the set of adjusted prototypes can be

used for several tasks and is often a preprocessing step in data analysis. One obvious usage is data compression because we can describe the given data set by a smaller set of prototypes. VQ models also are used for visualization of data [18] and for clustering [19], [20]. The principle of clustering is to partition the input space $\mathbb{R}^D$ into several disjoint subsets such that the union of all subsets is the input space itself and the intersection of each pair of subsets has zero measure. Each cluster is represented by one prototype. The data points $x_i \in \mathbb{R}^D$ belonging to a cluster are those data points for which the prototype is the winning prototype. These data form the Voronoi cell [21]. In context of neural systems (neural vector quantizers), the Voronoi cell is also denoted as receptive fields [22].

2.2. Supervised Vector Quantization

As mentioned in the last subsection the main difference of supervised VQ to unsupervised algorithms is that we have, additionally to the given data, class labels for every train data point available. The goal is to create a classifier function from the given training data, which can assign a label to new incoming unlabeled data points and minimizes the classification error for the given training data.

A simple, but very intuitive way, to tackle this problem is the k-Nearest Neighbor rule (k-NN) [23], [24]: Let x_i ($i = 1, \dots, n_d$) be the given training data points and

$$c : \mathcal{X} \rightarrow \{c_1, \dots, c_l\}$$

a formal class label function which returns the class labels for all training data points where l is the number of classes. For a new sample $\tilde{x}$

we search for the k nearest training data points, also called reference examples, $S_k = \{x_{S(1)}, \dots, x_{S(k)}\}$ regarding to some given dissimilarity measure $d(\cdot, \cdot)$ such that

$$d(x_{S(1)}, \tilde{x}) \leq \dots \leq d(x_{S(k)}, \tilde{x}) \leq d(x_j, \tilde{x}), \forall j \notin \{S(1), \dots, S(k)\}$$

holds. The usage of some similarity measure $s(\cdot, \cdot)$ instead of a dissimilarity measure is straight forward

$$s(x_{S(1)}, \tilde{x}) \geq \dots \geq s(x_{S(k)}, \tilde{x}) \geq s(x_j, \tilde{x}), \forall j \notin \{S(1), \dots, S(k)\}.$$

The label assignment, in the easiest case, is done by majority vote over these k nearest examples such that

$$c(\hat{x}) = c_i$$

with

$$i = \arg \max_j |\{x \in S_k | c(x) = c_j\}|.$$

The advantage of the approach is that we avoid any training process because of the fact that we use the given training data directly as reference examples. The only parameter for the model is the choice of the size k of the neighborhood which can be selected using cross validation. In addition, we have the possibility to use very general dissimilarity or similarity measures, which does not have to fulfill for example metric assumptions. One major drawback is that the size of the resulting classifier goes along with the number of given data points. Especially for large data sets this results in a huge computational effort for the out of sample classification

beside the necessity of data storage (see e.g. [25] for comparison). A further problem is that the resulting classifier tends to overfit and additionally is sensitive to outliers [25]. There exist selection procedures for the reference exemplars to relax these problems but it would advantageous to have a small set of reference exemplars, like in Vector Quantization, where prototypes describe the class specific data distribution in a proper way. We will discuss this topic later in detail.

One important special case of the k-NN rule is the Nearest Prototype Classifier (NPC), which is resulted for $k = 1$. In this case k-NN is related to K-Means. Yet, k-Means training is unsupervised, as mentioned before. Therefore, Kohonen suggested to modify NPC-based prototype learning, incorporating the class information of the training data into prototype learning. Having the NPC as goal in mind and using the ideas of Vector Quantization, Kohonen introduced the first idea for a Learning Vector Quantization (LVQ) [9]. The basic idea behind this class of learning algorithms is to describe the class wise distribution of the data by a small set of prototypes, which are also labeled according to the given classes whereby we assume at least one prototype per class. On the other hand, the resulting (NPC) classifier should approximate the decision boundary between the different classes in an accurate way.

The first version of Kohonens LVQ is LVQ 1. For this learning rule we assume several prototypes $\theta_1, \dots, \theta_{n_p}$ which also have labels $c(\theta_i), \dots, c(\theta_{n_p}) \in \{c_1, \dots, c_l\}$. In the beginning of the training, the prototypes are placed randomly or with some smart guess into the data space. During the training process we randomly select a data point x_i and determine the closest prototype $\theta_{s(x_i)}$ to x_i regarding equation (2.1). Thereafter we compare the label of the data point x_i with the label of the winning prototype and

determine the new position of the prototype $\theta_{s(x_i)}$ in the following way

$$\theta_{s(x_i)} = \begin{cases} \theta_{s(x_i)} + \alpha(x_i - \theta_{s(x_i)}) & \text{if } c(x_i) = c(\theta_{s(x_i)}) \\ \theta_{s(x_i)} - \alpha(x_i - \theta_{s(x_i)}) & \text{if } c(x_i) \neq c(\theta_{s(x_i)}) \end{cases} \quad (2.2)$$

with a decreasing learning rate α which satisfies $0 < \alpha \ll 1$. This process is repeated until the system is converged meaning that we have no more change in the prototype positions. The update rule in (2.2) and the so chosen learning rate realizing a vector shift which is an attraction if the labels of the data point and the prototype coincide and an repelling if they are different. The choice of $\alpha \ll 1$ ensures that we move the prototypes just a little way towards to the data point or away from the data point. We will call this principle therefore Attraction-Repulsing-Principle (ARP). Obviously, the prototypes tend to be class typical, in a converged state, because of the attraction term in the updates. Furthermore, the algorithm tries to reduce the number of wrongly classified data points if we use the learned prototypes in a NPC scenario. However, this goal is not explicitly addressed in the algorithm. The learning is only a heuristically motivated approximation of the Bayesian decision [26] driven by the attraction and repulsion. The convergence of the algorithm is ensured by a decreasing learning rate α . In fact, we do not know to which state the system converges regarding to the false classification of the given data points or any other quality measure for classification evaluation. To tackle this problem several cost function based methods were introduced [27]–[29] and will be discussed in more detail in the following.

2.2.1. Generalized Learning Vector Quantization

The Generalized Learning Vector Quantization (GLVQ) was introduced by Sato&Yamata in 1996 [27]. The authors constructed a cost function, which is an approximation of the accuracy keeping the ARP. For that, let x be any arbitrary data point. The prototype θ^+ which satisfies $c(x) = c(\theta^+)$ and $d(x, \theta^+) \leq d(x, \theta)$, $\forall \theta \in \Theta : c(x) = c(\theta)$ is called best matching correct prototype. Analogous θ^- with $c(x) \neq c(\theta^-)$ and $d(x, \theta^-) \leq d(x, \theta)$, $\forall \theta \in \Theta : c(x) \neq c(\theta)$ is the best matching incorrect one. Then the classifier function $\kappa(x)$ is defined via

$$\begin{aligned} \kappa(x) &= \frac{d(x, \theta^+) - d(x, \theta^-)}{d(x, \theta^+) + d(x, \theta^-)} \\ &= \frac{h(x)}{d(x, \theta^+) + d(x, \theta^-)} \end{aligned} \quad (2.3)$$

where $h(x) = d(x, \theta^+) - d(x, \theta^-)$ is called the hypothesis margin [30]. The classifier function is bounded to the interval $[-1, 1]$ by the denominator and is negative iff the data point is correctly classified and positive iff the data point is misclassified. Resulting from those observations, we want to minimize

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} f_a(\kappa(x_i)) \quad (2.4)$$

where Sato&Yamata introduced the choice of the activation function f_a as any arbitrary monotonic increasing function. An extension of the class of the potential activation functions can be found, for example, in [31]. However, the most common choices for the function f_a are the identity

$$\mathcal{ID}(z) = z$$

and the sigmoid function

$$\text{sig}(z) = \frac{1}{1 + \exp(-\frac{z}{\sigma})}. \quad (2.5)$$

Here the parameter σ has a high influence on the learning behavior of the classifier. For large values of σ the sigmoid function approximates a linear function and for very small values the sigmoid function approximates the Heaviside function

$$\text{H}(\xi) = \begin{cases} 1 & \text{if } \xi > 0 \\ 0 & \text{else .} \end{cases} \quad (2.6)$$

(see Figure 2.1). This results to different behavior of the learning process. In the case of large σ we optimize the adjusted hypothesis margin $h(x)$. For small values of σ we minimize the number of false classified data points. The second scenario is also known as border sensitive learning [32]. The optimization of the cost function (2.4) takes place by stochastic gradient descent learning (SGDL). The corresponding gradients regarding a given data point x_i are

$$\nabla \theta^\pm = \frac{\partial f_a(\kappa(x_i))}{\partial \kappa(x_i)} \frac{\partial \kappa(x_i)}{\partial d(x_i, \theta^\pm)} \frac{\partial d(x_i, \theta^\pm)}{\partial \theta^\pm}. \quad (2.7)$$

In the training phase of the classifier we choose randomly a data point out of the training set and update the best matching prototypes $\theta^\pm$ according to

$$\theta^\pm = \theta^\pm - \alpha_t \nabla \theta^\pm$$

realizing an SGDL. The learning rate α has to be chosen from $\alpha \in (0, 1)$ and should be decreased during training. To be more precise the learning

Figure 2.1.: Properties of the activation function $\text{sig}(z)$ depending on the parameter σ .

rate α should be time depended in such way that the learning rate in the t -th step of the training α_t is chosen such that $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$ to ensure the convergence to a global optimum [33], [34]. One choice of such an learning rate could be $\alpha_t = \frac{1}{t}$ which corresponds to the harmonic series. The common choice for the dissimilarity measure in equation (2.4) is the Euclidean distance. In this case the gradient (2.7) is a vector shift known from LVQ, but with a data dependent learning factor realizing the ARP. Yet, the GLVQ scheme offers a greater flexibility with respect to a given application case compared to standard LVQ. E.g., it is possible to use any differentiable distance/dissimilarity measure instead of the Euclidean distance. This ensures that we can adjust the GLVQ on a wide range of data problems by using different measures like distances based on kernels, divergences and many more [35]. Depending on the used distance measure the interpretation of the gradient as vector shift

is lost.

To sum up GLVQ is a flexible training algorithm for NPC classifiers with a wide usage in classification. The main important restrictions are that the dissimilarity measure has to be differentiable with respect to the prototypes and the data have to be given in vectorial form.

2.2.2. Robust Soft Learning Vector Quantization

Robust Soft Learning Vector Quantization (RSLVQ) as introduced in [29] is a probabilistic variant of LVQ. RSLVQ can be seen as special case of the more general PLVQ introduced in [36]. The respective model probabilities are estimated in terms of a (Gaussian) mixture model for the class-dependent data densities. Thereby, the main assumption is that each mixture component is responsible to generate samples of exactly one class. Further we suppose that the mixture component j is parametrized by a vector θ_j . The respective class responsibility is indicated by the class label $c(\theta_j)$ as before. We explicitly remark that several mixture components can contribute to a certain class. Further, for the choice of Gaussian for the mixture components the ARP should be kept. The model density of a data point regarding a given class c_k is defined via

$$p(x_i, c_k | \Theta) = \sum_{j=1}^{n_p} \delta_{c_k}^{c(\theta_j)} p(\theta_j) p(x_i | \theta_j)$$

with the Kronecker delta

$$\delta_{c_k}^{c(\theta_j)} = \begin{cases} 1 & \text{if } c_k = c(\theta_j) \\ 0 & \text{else} \end{cases}.$$

Typically, the prior probability $p(\theta_j)$ is chosen as constant and equal for all mixture components [29]. The conditional probability $p(x_i|\theta_j)$ is defined in RSLVQ via a simplified Gaussian density

$$p(x_i|\theta_j) = \exp\left(-\frac{1}{2}\left(\frac{d_E(x_i, \theta_j)}{\sigma}\right)^2\right).$$

Here $d_E(x_i, \theta_j)$ is the Euclidean distance. Generalizations via changing the dissimilarity measure are possible in a similar way as for GLVQ see e.g. [37] but in this case we do not have a Gaussian mixture model anymore. The Gaussian assumption allows a geometric interpretation. In this setting θ_j are the centers of the Gaussians, and therefore, play the role of prototypes like in LVQ/GLVQ. Now, the probability that a certain data point has label c_k can be formalized with respect to this model as

$$p(c_k|x_i, \Theta) = \frac{p(x_i, c_k|\Theta)}{\sum_j p(x_i, c_j|\Theta)} \quad (2.8)$$

which can be seen as likelihood ratio of the probability that x_i belongs to class c_k and the probability that x_i fits into the whole data model. The goal for the optimization is to maximize the probability for the correct classification. Using the log likelihood ratio this objective can be reformulated as maximization of the log-likelihood function

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \log(p(c(x_i)|x_i, \Theta)) \quad (2.9)$$

which can be optimized by stochastic gradient ascent learning (SGAL) (see e.g. [29]). For a given data point x_i the gradients for prototype

learning become

$$\nabla\theta_k = \begin{cases} \frac{\exp\left(-\frac{1}{2}\left(\frac{d(x_i, \theta_k)}{\sigma^2}\right)^2\right)}{\sum_{j=1}^{n_p} \delta_{c(x_i)}^{c(\theta_j)} \exp\left(-\frac{1}{2}\left(\frac{d(x_i, \theta_j)}{\sigma^2}\right)^2\right)} (x_i - \theta_k) & \text{if } c(x_i) = c(\theta_k) \\ -\frac{\exp\left(-\frac{1}{2}\left(\frac{d(x_i, \theta_k)}{\sigma^2}\right)^2\right)}{\sum_{j=1}^{n_p} (1 - \delta_{c(x_i)}^{c(\theta_j)}) \exp\left(-\frac{1}{2}\left(\frac{d(x_i, \theta_j)}{\sigma^2}\right)^2\right)} (x_i - \theta_k) & \text{if } c(x_i) \neq c(\theta_k) \end{cases}$$

dependent on class agreement between the data label and the prototype label under the assumption of the Euclidean distance. The resulting update

$$\theta_k = \theta_k + \alpha \nabla\theta_k$$

is again equipped with a learning rate α , requiring the same conditions as for GLVQ. The update interpretation as vector shift with data dependent factor remains unchanged. Furthermore, we have an attraction of the prototypes in case of label agreement and a repulsion otherwise.

The labeling of new unlabeled data takes place by using equation (2.8) applying a Bayes-like probability maximum principle. Obviously, this is not a nearest neighbor decision because all prototypes with the same label are involved in the decision. Only in the special case $\sigma \rightarrow 0$ or in the case where only one prototype per class is used NPC will be achieved. Similar to GLVQ, the data have to be given in vectorial form and the distance measure is required to be differentiable.

2.2.3. Soft Nearest Prototype Classification

Soft Nearest Prototype Classification (SNPC) was first introduced in [28] and model a relaxed version of the classification error as cost function

keeping the NPC principle which is the main difference to RSLVQ. The misclassification for given training data x_i ($i = 1, \dots, n_d$) and a set of prototypes θ_j ($j = 1, \dots, n_p$) is evaluated by

$$\mathcal{C}(\mathcal{X}, \Theta) = \frac{1}{n_d} \sum_{i=1}^{n_d} ls_i \quad (2.10)$$

whereby $ls_i = \sum_{j=1}^{n_p} p(\theta_j|x_i) \left(1 - \delta_{c(x_i)}^{c(\theta_j)}\right)$ is called the local costs. In case of crisp classification like in NPC classifiers the assignment probability $p(\theta_j|x_i)$ can be expressed as winner takes all rule

$$p(\theta_j|x_i) = \delta_{\theta_j}^{\theta_{s(x_i)}} \quad (2.11)$$

whereby $\theta_{s(x_i)}$ is the nearest prototype to data point x_i as known from LVQ. Obviously, $\mathcal{C}(\mathcal{X}, \Theta)$ is not differentiable in case of crisp assignments (2.11) and hence, SGDL would not be applicable. Therefore, the authors in [28] replace (2.11) by

$$p(\theta_j|x_i) = \frac{\exp\left(-\frac{1}{2}\left(\frac{d_E(x_i, \theta_j)}{\sigma}\right)^2\right)}{\sum_{k=1}^{n_p} \exp\left(-\frac{1}{2}\left(\frac{d_E(x_i, \theta_k)}{\sigma}\right)^2\right)} \quad (2.12)$$

where $d_E(\theta_j, x_i)$ is the Euclidean distance. For any given data point x_i , the SGDL scheme with respect to a prototype θ_k results

$$\theta_k = \theta_k - \alpha \nabla \theta_k$$

with

$$\nabla\theta_k = \begin{cases} -p(\theta_k|x_i)ls_i(x_i - \theta_k) & \text{if } c(x_i) = c(\theta_k) \\ p(\theta_k|x_i)(1 - ls_i)(x_i - \theta_k) & \text{if } c(x_i) \neq c(\theta_k). \end{cases}$$

In consequence, the ARP scheme is kept.

The classification for any unlabeled out of sample data point can be done with a simple nearest neighbor decision. In analogy to the two LVQ variants discussed above, it is assumed that the training data is available in vectorial form and the dissimilarity measure has to be differentiable regarding the prototypes.

2.2.4. Support Vector Machines

Although it is not a prototype based method in the sense of VQ we also introduce the Support Vector Machines (SVM). Further, SVM are used in this thesis as a widely applied classifier for comparison with results obtained by new approaches. Thus, we only give a brief introduction to the basic ideas and properties of SVM, which are relevant for the comparison. A detailed description can be found, for example, in [11].

The first difference to the previous VQ-classifiers is that SVM's are originally defined only for two-class problems. These classes are handled as positive class (+1) and negative class (−1). Moreover the class labels are handled as numerical class labels, meaning $c(x) \in \{-1, 1\} \forall x$. In case of multi-class problems we have to use heuristics like one-versus-all or one-versus-one to make the SVM usable (see eg. [38]).

For the moment, we assume that we have given a binary classification problem where the classes can be separated by a hyperplane, i.e. the problem is linearly separable. The idea of SVM is to find that hyper-

plane, which divides the data space in such a way that all data points with the same numerical label are located on the same side of the hyperplane and the distance between the hyperplane and the nearest data point of every class is maximized (maximization of the separation margin). The hyperplane itself can easily be described by the set of all points $x \in \mathbb{R}^D$ for which $\mathbf{w} \cdot x - \mathbf{b} = 0$ holds, whereby $\mathbf{w}$ is the normal vector of the hyperplane and $\mathbf{b}$ is a bias which encodes the distance of the hyperplane from the origin.

The two goals of maximizing the separation margin on the one hand and classifying all training samples correctly at the other hand can be expressed as an optimization problem with constraints in the following form (see e.g. [11])

$$\min_{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}} \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} \text{ subject to } c(x_i) \cdot (\langle \mathbf{w}, x_i \rangle + \mathbf{b}) \geq 1, \forall i = 1, \dots, n_d$$

where $\langle \cdot, \cdot \rangle$ is the standard Euclidean inner product. This convex optimization problem can be further reformulated as the Wolfe dual problem

$$\max_{\alpha_i} \left(\sum_{i=1}^{n_d} \alpha_i - \sum_{i=1}^{n_d} \sum_{j=1}^{n_d} c(x_i) c(x_j) \alpha_i \alpha_j \langle x_i, x_j \rangle \right) \quad (2.13)$$

$$\text{subject to } \sum_{i=1}^{n_d} \alpha_i c(x_i) = 0 \wedge \alpha_i \geq 0, \forall i = 1, \dots, n_d \quad (2.14)$$

using the Lagrange formulation of constraint optimization with the Lagrangian multipliers α_i . The problem (2.13) under the constraints (2.14) can be solved by, e.g. quadratic programming, in an efficient way. The optimal decision hyperplane can be reconstructed using the Lagrangian

multipliers

$$\begin{aligned}\mathfrak{w} &= \sum_{i=1}^{n_d} \alpha_i c(x_i) x_i \\ \mathfrak{b} &= \frac{1}{n_d} \sum_{j=1}^{n_d} \left(c(x_j) - \sum_{i=1}^{n_d} \alpha_i c(x_i) \langle x_i, x_j \rangle \right)\end{aligned}$$

and the decision function $c(x_k)$ for the class label can be formulated as

$$c(x_k) = \text{sgn} \left(\sum_{i=1}^{n_d} \alpha_i c(x_i) \langle x_k, x_i \rangle + \mathfrak{b} \right). \quad (2.15)$$

The detailed derivations of the formulas can be found in [11].

The essential differences between SVM and LVQ methods result from the formulas (2.13),(2.14),(2.15): First, one can see that we do not need the data points in explicit form for the description and solution of the optimization problem as well as for the decision function. Hence, it is sufficient that the pairwise inner products of all training data are available. This leads to the first conclusion that non-vectorial data can be used for SVM as long as an inner product can be defined between them. Second, the model complexity depends directly on the number of Lagrangian multipliers different from zero. Thus, in comparison to LVQ methods the model complexity cannot be controlled directly or specified in advance.

So far we assumed that the problem is linearly separable. This would be a hard restriction to the applicability of the SVM in practice. There are several ways to overcome this restriction. One is to use the so-called kernel trick. Thereby, the data points are implicitly mapped into a high-

dimensional Hilbert vector space in which the data should be linearly separable with high probability, as known from the XOR problem which is linearly separable in 3d but not in 2d. Instead of doing this mapping explicitly, a so-called kernel function is used which allows to calculate the inner product in the mapping space in terms of the given data in the original space. Thus, an explicit evaluation of the mapping is not necessary. For more details, see e.g. [11].

The second way to avoid the problem of linear separability is to soften the hard constraints. For this purpose, so-called slack variables $\zeta_i \geq 0$ are introduced

$$c(x_i) \cdot (\langle \mathbf{w}, x_i \rangle + \mathbf{b}) \geq 1 - \zeta_i, \forall i = 1, \dots, n_d.$$

To ensure that the constraints are not violated in an arbitrary manner, a penalty term is added to the main condition

$$\min_{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}} \frac{\langle \mathbf{w}, \mathbf{w} \rangle}{2} + \mathbf{c} \sum_{i=1}^{n_d} \zeta_i$$

whereby $\mathbf{c}$ controls the trade-off between correct classification of the training samples and maximization of the separation margin. Of course, both ideas can be used at the same time and make Support Vector Machine a powerful tool for classification with the mentioned limitations.

3. Proximity Measures for Prototype Based Models

Comparing data in terms of similarity or dissimilarity measures is one of the key ingredients of machine learning. For the large variety of data and algorithms, a wide range of respective measures has been developed. The term proximity measure can be used as a generic term for this diverse variety of measurements with different characteristics. Generally, we understand every measure fulfilling the next definition as proximity.

Def. 1 *Let $\mathcal{X}$ be a set of arbitrary data objects x . A proximity measure, denoted as $\mathbf{p}$, is a function that assign a real value to each arbitrary pair of data objects*

$$\mathbf{p} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \quad (3.1)$$

Casually speaking, any quantity, measuring the similarity or dissimilarity between objects can be understood as proximity measure. Also inner products and kernel functions are types of proximities. We will see that we have to distinguish between (dis-)similarities on the one hand and inner products and kernels on the other hand. Proximity measures could be only exemplary of a mathematical nature, such as distances, correlations or divergences. Also questionnaire values, counting variables or set

distances [39]–[42] are used in machine learning.

The description of general proximity measures is very different in the literature and depends on the community. As a result, several measures with different properties are often listed under the same name. As an example, the term similarity is often used in the context of SVM, where inner products or general kernel functions are in use [42]. However, if the term similarity is used in connection with cluster algorithms such as affinity propagation [5], it is often meant as a similarity in the sense that two data items are more similar if they sharing more properties. Therefore, it is essential to know the exact properties of proximity measures in order to analyze them within appropriate algorithms.

In this chapter we introduce a taxonomy of proximity measures based on the previous work of Tversky and Gati [43] and the first taxonomy attempt given in [13]. Furthermore, we introduce a measure to evaluate the differences between given proximities when used in nearest neighbor based classifiers. This measure also serves to quantify how different two similarity/dissimilarity measures for a classification task or vector quantization behave. Further, it can be used to evaluate whether the information content of a proximity measure is changed by data preprocessing, necessary for certain algorithms like kernel methods. This problem is addressed explicitly in the last part of this chapter, when handling of non-vectorial data for classification problems is considered.

This chapter of the thesis is mainly based on our work [44].

3.1. Types of Proximities

3.1.1. Similarities and Dissimilarities

Now we consider basic requirements of proximities, which should be valid for a similarity measure. One of the most obvious properties is that for every object, no other object is more similar to this object than this one to itself. More precisely, if $\mathcal{X}$ is an object space with arbitrary objects then a similarity measure $\mathfrak{s} : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ should fulfill the following property [43]

$$[\mathfrak{s}(x_i, x_i) \geq \mathfrak{s}(x_i, x_j)] \wedge [\mathfrak{s}(x_i, x_i) \geq \mathfrak{s}(x_j, x_i)] \quad \forall x_j, x_i \in \mathcal{X} \quad (3.2)$$

which is called the maximum or dominance principle (MAX as abbreviation). Likewise, this principle can also be adapted to a measure of dissimilarity $\mathfrak{d} : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$

$$[\mathfrak{d}(x_i, x_i) \leq \mathfrak{d}(x_i, x_j)] \wedge [\mathfrak{d}(x_i, x_i) \leq \mathfrak{d}(x_j, x_i)] \quad \forall x_j, x_i \in \mathcal{X}$$

where it is called consequently minimum principle (MIN as abbreviation). We will call a (dis)-similarity fulfilling the maximum/minimum principle as basic (dis)-similarity. One interesting example, which does not fulfill the dominance principle, is introduced by the authors in [45] and is used as similarity in [42]. The data set with the name patrol data was collected in the following form: All members from seven different patrol units were asked to name five members of their own unit. The similarity between pairwise objects is then defined $\mathfrak{s}(x_i, x_j) = \frac{N(x_j, x_i) + N(x_i, x_j)}{2}$ where $N(x_i, x_j)$ counts how often x_i names x_j . Thus for the similarity follows $\mathfrak{s}(x_i, x_j) \in \{0, 0.5, 1\}$. Of course, most people usually do not

name themselves, so in this case the similarity $\mathfrak{s}(x_i, x_i)$ will be zero. On the other hand the person has to name five others such that there are at least five objects which have at minimum the similarity of 0.5 to this person. This already shows that a measurement based on a simple counting variable can violate the basic condition of a similarity, meaning it is only a general proximity. However, in this example the maximum principle can be easily saved by setting the similarity of an object to itself to one, which does not change the information content of the data.

Furthermore, the second frequently demanded property for similarities and dissimilarities is the non-negativity (NN as abbreviation)

$$\begin{aligned} \mathfrak{s}(x_i, x_j) &\geq 0 & \forall x_i, x_j \\ \mathfrak{d}(x_i, x_j) &\geq 0 & \forall x_i, x_j \end{aligned} \tag{3.3}$$

where the regarding (dis)-similarities fulfilling (3.2) and (3.3) are called primitive (dis)-similarities. Following the categorization in [43], the maximum principle and the non-negativity are often coupled with a bound for self-similarity $\mathfrak{s}(x_i, x_i) = \mathfrak{r}_{\mathfrak{s}}$ with $\mathfrak{r}_{\mathfrak{s}} = 1 \ \forall x_i \in \mathcal{X}$ as further constrain which we refer as normalized consistency (nC as abbreviation). The more gradual definition of that boundary is taken $\mathfrak{r}_{\mathfrak{s}} \in \mathbb{R}$ as arbitrary constant (strong consistency: sC) or more weak as data dependent boundary $\mathfrak{r}_{\mathfrak{s}}(x_i) \in \mathbb{R} \ \forall x_i \in \mathcal{X}$ (weak consistency: wC). Similarly, there might be a weak consistency $\mathfrak{r}_{\mathfrak{d}}(x_i) \in \mathbb{R} \ \forall x_i \in \mathcal{X}$ and a strong consistency $\mathfrak{r}_{\mathfrak{d}} \in \mathbb{R}$ for dissimilarity measures. The reflexivity is the comparable property (R as abbreviation) for dissimilarities $\mathfrak{d}(x_i, x_i) = 0 \ \forall x_i \in \mathcal{X}$ which is of particular interest if the measure should be interpreted geometrically: For mathematical distances $\mathfrak{d}(x_i, x_j)$ between data points $x_i, x_j \in \mathbb{R}^n$,

$\mathfrak{d}(x_i, x_i) = 0$ is obviously demanded.

Here, we emphasize that inner products or kernel functions cannot be interpreted as a similarity in the introduced sense, in general. A simple counter example shows this statement: Let us consider the two vectors $x_i = (1, 1)^T$ and $x_j = (2, 2)^T$. The Euclidean inner products are $\langle x_i, x_i \rangle_E = 2$ and $\langle x_i, x_j \rangle_E = 4$ and, hence, the maximum principle for similarities is violated. Depending on the given vectors, the Euclidean inner product can also become negative and, thus, violates the principle of non-negativity. Since the Euclidean inner product is a special case of the polynomial kernel, it is a good example why the interpretation of kernel functions as similarity can be misleading. A consistent similarity measure (cosine similarity) based on the Euclidean inner product is

$$\mathfrak{s}_c(x_i, x_j) = \frac{\langle x_i, x_j \rangle_E}{\|x_i\|_E \|x_j\|_E}$$

which is consistent with the Euclidean distance in the geometric sense [46]. Yet, it is only a basic similarity because it does not fulfill the non-negativity property.

Many algorithms require symmetry (S as abbreviation) for proximities

$$\begin{aligned} \mathfrak{s}(x_i, x_j) &= \mathfrak{s}(x_j, x_i) & \forall x_i, x_j \in \mathcal{X} \\ \mathfrak{d}(x_i, x_j) &= \mathfrak{d}(x_j, x_i) & \forall x_i, x_j \in \mathcal{X} \end{aligned} \quad (3.4)$$

like kernel/relational LVQ methods [47]. Representatives of non-symmetric proximities are measures based on counting variables like the above mentioned patrol problem. Yet, many mathematically defined dissimilarity measures like divergences are generally not symmetric (see e.g. [48]).

The next issue to be considered is the property of definiteness or non-degeneration (D as abbreviation), which is closely related to the MIN/MAX property. It simply says, if an object x_j has a similarity to object x_i , which is equal to the self similarity, than the object x_j should be not distinguishable from x_i .

$$\begin{aligned}\mathfrak{s}(x_i, x_j) = \mathfrak{s}(x_i, x_i) \vee \mathfrak{s}(x_i, x_j) = \mathfrak{s}(x_j, x_j) &\implies x_i = x_j \\ \mathfrak{d}(x_i, x_j) = \mathfrak{d}(x_i, x_i) \vee \mathfrak{d}(x_i, x_j) = \mathfrak{d}(x_j, x_j) &\implies x_i = x_j\end{aligned}\quad (3.5)$$

Finally, we extend the list of properties by the triangle inequality (T or rT as abbreviation)

$$\begin{aligned}\mathfrak{s}(x_i, x_j) + \mathfrak{s}(x_j, x_k) &\leq \mathfrak{s}(x_i, x_k) & \forall x_i, x_j, x_k \in \mathcal{X} \\ \mathfrak{d}(x_i, x_j) + \mathfrak{d}(x_j, x_k) &\geq \mathfrak{d}(x_i, x_k) & \forall x_i, x_j, x_k \in \mathcal{X}\end{aligned}\quad (3.6)$$

which leads to a metric or Minkowsky-like similarity if all the properties minimum/maximum principle, non-negativity, normalized consistency (reflectivity) and symmetry are also fulfilled. In the even stronger formulation as ultra metric inequality (UM or rUM as abbreviation)

$$\max \{\mathfrak{s}(x_i, x_j), \mathfrak{s}(x_j, x_k)\} \leq \mathfrak{s}(x_i, x_k) \quad (3.7)$$

$$\max \{\mathfrak{d}(x_i, x_j), \mathfrak{d}(x_j, x_k)\} \geq \mathfrak{d}(x_i, x_k) \quad (3.8)$$

we obtain ultra metrics/ultra similarities.

All introduced properties and kinds of proximities are summarized in Table 3.1, Table 3.2 and Table 3.3.

	$\mathfrak{d}(\mathbf{x}_i, \mathbf{x}_j)$	$\mathfrak{s}(\mathbf{x}_i, \mathbf{x}_j)$	
MIN	minimum principle $\mathfrak{d}(x_i, x_i) \leq \mathfrak{d}(x_i, x_j) \wedge$ $\mathfrak{d}(x_j, x_j) \leq \mathfrak{d}(x_i, x_j)$	maximum principle $\mathfrak{s}(x_i, x_i) \geq \mathfrak{s}(x_i, x_j) \wedge$ $\mathfrak{s}(x_j, x_j) \geq \mathfrak{s}(x_i, x_j)$	MAX
NN	non-negativity $\mathfrak{d}(x_i, x_j) \geq 0$	$\mathfrak{s}(x_i, x_j) \geq 0$	NN
wC	weak consistency $\mathfrak{d}(x_i, x_i) = \mathfrak{r}_\mathfrak{d}(x_i)$ $(\mathfrak{r}_\mathfrak{d}(x_i), \mathfrak{r}_\mathfrak{s}(x_i) : X \rightarrow \mathbb{R})$	$\mathfrak{s}(x_i, x_i) = \mathfrak{r}_\mathfrak{s}(x_i)$	wC
sC	strong consistency $\mathfrak{d}(x_i, x_i) = \mathfrak{r}_\mathfrak{d}$	$\mathfrak{s}(x_i, x_i) = \mathfrak{r}_\mathfrak{s}$ $c_d, c_s \in \mathbb{R}$	sC
R	reflexivity $\mathfrak{d}(x_i, x_i) = 0$	normalized sC $\mathfrak{s}(x_i, x_i) = 1$	nC
S	symmetry $\mathfrak{d}(x_i, x_j) = \mathfrak{d}(x_j, x_i)$	$\mathfrak{s}(x_i, x_j) = \mathfrak{s}(x_j, x_i)$	S
D	definiteness (non-degeneration) $\mathfrak{d}(x_i, x_j) = \mathfrak{d}(x_i, x_i) \vee$ $\mathfrak{d}(x_i, x_j) = \mathfrak{d}(x_j, x_j) \Rightarrow x_i = x_j$	$\mathfrak{s}(x_i, x_j) = \mathfrak{s}(x_i, x_i) \vee$ $\mathfrak{s}(x_i, x_j) = \mathfrak{s}(x_j, x_j) \Rightarrow x_i = x_j$	D
T	triangle inequality $\mathfrak{d}(x_i, x_j) + \mathfrak{d}(x_j, x_k) \geq \mathfrak{d}(x_i, x_k)$	reverse triangle inequality $\mathfrak{s}(x_i, x_j) + \mathfrak{s}(x_j, x_k) \leq \mathfrak{s}(x_i, x_k)$	rT
UM	ultra-metric inequality $\max\{\mathfrak{d}(x_i, x_j), \mathfrak{d}(x_j, x_k)\} \geq$ $\mathfrak{d}(x_i, x_k)$	reverse ultra-sim. inequality $\min\{\mathfrak{s}(x_i, x_j), \mathfrak{s}(x_j, x_k)\} \leq$ $\mathfrak{s}(x_i, x_k)$	rUS

Table 3.1.: Types of dissimilarity and similarity measures based on complementary mathematical properties. The categorization of properties are motivated by cognitive-psychological deliberations as well as geometrical thoughts.

dissimilarities/properties	MIN	NN	consistency			S	D	inequalities	
			wC	sC	R			T	UM
basic dis.	x								
primitive dis.	x	x							
weakly-consistent dis.	x	x	x						
strongly-consistent dis.	x	x		x					
general dis. (hollow metric)	x	x			x				
pre dis. (pre-metric)	x	x			x	x			
usual dis. (quasi-metric)	x	x			x	x	x		
semi-metric	x	x			x	x		x	
distance (metric)	x	x			x	x	x	x	
ultra metric	x	x			x	x	x		x

Table 3.2.: Types of dissimilarities regarding the properties identified in Table 3.1.

3.1.2. Inner Products and Kernel Functions

We have introduced similarities and dissimilarities as special proximities and categorized them according to their properties. We have also already mentioned that e.g. inner products or kernels are generally not similarities. In order to highlight the interrelationships and differences more precisely, we will consider inner products and generalizations thereof in more detail. For the further discussion let us assume that the data set is coupled with a vector summation ($+$: $\mathcal{X} \times \mathcal{X} \rightarrow \mathcal{X}$) and scalar multiplication ($\cdot$: $\mathbb{R} \times \mathcal{X} \rightarrow \mathcal{X}$) such that $(\mathcal{X}, +, \cdot)$ is a (real-valued) vector space. As long as there is no risk of mistaking, we continue to shorten

similarities/properties	MAX NN		consistency			S	D	inequalities	
			wC	sC	nC			rT	rUS
basic sim.	x								
primitive sim.	x	x							
weakly-consistent sim.	x	x	x						
weak sim.	x		x						
strongly-consistent sim.	x	x		x					
general sim. (hollow sim.)	x	x			x				
pre-sim.	x	x			x	x			
usual sim. (quasi-sim.)	x	x			x	x	x		
semi-sim.	x	x			x	x		x	
(Minkowsky-like) sim.	x	x			x	x	x	x	
ultra sim.	x	x			x	x	x		x

Table 3.3.: Types of similarities regarding the properties identified in Tab.3.1.

$(\mathcal{X}, +, \cdot)$ simply as $\mathcal{X}$. From a formal point of view, every real-valued¹ inner product (IP) is a proximity with the properties

$$1) \langle \alpha x_i + \beta x_j, x_k \rangle = \alpha \langle x_i, x_k \rangle + \beta \langle x_j, x_k \rangle \quad (3.9)$$

$$2) \langle x_i, x_j \rangle = \langle x_j, x_i \rangle \quad (3.10)$$

$$3) \langle x_i, x_i \rangle \geq 0, \langle x_i, x_i \rangle = 0 \Leftrightarrow x_i = \mathbf{0} \quad \forall x_i, x_j, x_k \in \mathcal{X} \quad \forall \alpha, \beta \in \mathbb{R}. \quad (3.11)$$

As already mentioned inner products do not have to be positive and the minimum principle is maybe violated. Hence, inner products generally are not a measure of similarity, even not a basic similarity, and thus in

¹Usually inner products are defined as mappings into the complex numbers, but we restrict ourself to the case of real numbers, which is the most frequently used scenario in machine learning.

general a particular type of proximity. A vector space equipped with an inner product is called a pre-Hilbert space. The inner product induces a norm [13]

$$\|x_i\| = \sqrt{\langle x_i, x_i \rangle} \quad (3.12)$$

and therefore $\mathcal{X}$ becomes a normed vector space. If the vector space is complete regarding the norm (3.12) we refer the vector space as Banach space ($\mathcal{B}$) corresponding to the norm and as a Hilbert space ($\mathcal{H}$), if $\mathcal{X}$ is equipped with an inner product. Yet, inner products are always related to distances in such a way that they generate a distance via

$$\mathfrak{d}(x_i, x_j) = \|x_i - x_j\| = \sqrt{\langle x_i - x_j, x_i - x_j \rangle} \quad (3.13)$$

such that $\mathcal{X}$ is a metric space [13]. In general, however, the reversal, finding a related inner product or even a norm to a given metric, is not possible [13]. A metric $\mathfrak{d}$ induces a norm $\|x\| = \mathfrak{d}(x, 0)$ iff $\mathfrak{d}$ is translation invariant and homogeneous. Further there exist a related inner product iff this norm fulfills the parallelogram equation

$$2(\|x_i\|^2 + \|x_j\|^2) = \|x_i + x_j\|^2 + \|x_i - x_j\|^2.$$

Inner products can be generalized and we give here only a very rough overview over the most important variations. Detailed explanations can be found e.g. in [49], [50]. Two possible types of generalization play a major role in machine learning.

First we mention $\langle \cdot, \cdot \rangle_{id}$, the so-called indefinite inner products (IIP) [13], [51]. Those are generalizations in the sense that the property of positive definiteness (3.11) is dropped whereas the properties (3.9) and (3.10) are

kept². A vector space equipped with $\langle \cdot, \cdot \rangle_{id}$ is called an indefinite inner product space (IIPS). Formally, we can classify the vectors of the vector space into three categories regarding the IIP. An element $x \in \mathcal{X}$ is called positive if $\langle x, x \rangle_{id} > 0$ holds, negative if $\langle x, x \rangle_{id} < 0$ holds and neutral $\langle x, x \rangle_{id} = 0$ holds. A subset $\hat{\mathcal{X}} \subset \mathcal{X}$ is called positive/negative/neutral if all $x \in \hat{\mathcal{X}}$ are positive/negative/neutral. The respective subsets³ are indicated by $\mathcal{X}^+, \mathcal{X}^-$ and $\mathcal{X}^0$. An orthogonal decomposition of the IIPS in the form

$$\mathcal{X} = \mathcal{X}^+ \oplus \mathcal{X}^- \oplus \mathcal{X}^0 \quad (3.14)$$

is called a fundamental decomposition and $\mathcal{X}$ is called decomposable. The direct sum (3.14) means that every $x \in \mathcal{X}$ can be uniquely decomposed in $x = x^+ + x^- + x^0$ with $x^+ \in \mathcal{X}^+$, $x^- \in \mathcal{X}^-$ and $x^0 \in \mathcal{X}^0$. The term orthogonal decomposition implies that x^+ , x^- and x^0 are pairwise orthogonal, meaning explicitly $\langle x^+, x^- \rangle_{id} = 0$. Indefinite inner product spaces which are decomposable are referred as Kreĭn spaces ($\mathcal{K}$). Not every IIPS is a Kreĭn space, but finite-dimensional spaces are so [13]. One interesting special case of Kreĭn spaces is the pseudo-Euclidean space $\mathbb{R}^{(p,q)}$ which is a real valued vector space with the orthogonal decomposition $\mathbb{R}^{(p,q)} = \mathbb{R}^p \oplus \mathbb{R}^q \oplus \mathbb{R}^0$ [51]. The IIP in a pseudo-Euclidean space

²It is also possible to introduce another intermediate stage. For semi-indefinite inner products $\langle x_i, x_i \rangle \geq 0$ holds but $\langle x_i, x_i \rangle = 0 \Leftrightarrow x_i = \mathbf{0}$ is not necessarily fulfilled [52].

³If we complete the sets $\mathcal{X}^+$ and $\mathcal{X}^-$ with the zero vector they fulfilling all properties to be subspaces [51]

can be expressed via the standard Euclidean inner product

$$\begin{aligned}\langle x_i, x_j \rangle_{p,q} &= \sum_{k=1}^p x_i(k)x_j(k) - \sum_{l=p+1}^{p+q} x_i(l)x_j(l) \\ &= \langle x_i^+, x_j^+ \rangle_E - \langle x_i^-, x_j^- \rangle_E.\end{aligned}$$

Corresponding to the indefinite inner product, the indefinite norm becomes $\|x\|_{p,q}^2 = \langle x, x \rangle_{p,q}$ [13] which can have any sign. Related to this, the indefinite distance is defined

$$\mathfrak{d}_{p,q}^2(x_i, x_j) = \|x_i - x_j\|_{p,q} = \langle x_i - x_j, x_i - x_j \rangle_{p,q} \quad (3.15)$$

which is refereed as pseudo-Euclidean square distance [13]. The proximity defined in this way is, however, only a very basic one regarding to the taxonomy presented. In particular, the measure is not necessarily positive for any pairs of objects. This has to be taken into account in the corresponding applications. Respective application examples can be found in [47], for LVQ models, or in [42] and [53], among others for SVM.

A second class of generalizations are the so-called semi-inner-products [49], [50]. Compared to the inner products, linearity (3.9) is preserved as a property but symmetry is not required. More precise, every mapping⁴

⁴We restrict ourselves again to the most common case in machine learning of real valued semi-inner-product.

$[\cdot, \cdot] \rightarrow \mathbb{R}$ which fulfills

$$1) [\alpha x_i + \beta x_j, x_k] = \alpha [x_i, x_k] + \beta [x_j, x_k]$$

$$2) [x_i, x_i] > 0 \text{ if } x \neq \mathbf{0}$$

$$3) |[x_i, x_j]|^2 \leq [x_i, x_i] [x_j, x_j]$$

is a semi-inner-product (SIP). The respective vector space is then called a SIP space (SIPS). Every SIPS is a normed vector space regarding the norm $\|x\|_{\text{sip}} = \sqrt{[x, x]}$ and, therefore a Banach space, if it is a complete vector space regarding this norm. Moreover every normed space can be represented as SIPS [50]. Following these results every SIPS is a metric space with the metric induced by that norm

$$\mathfrak{d}_{\text{sip}}(x_i, x_j) = \sqrt{\|x_i - x_j\|_{\text{sip}}} \quad (3.16)$$

$$= \sqrt{[x_i - x_j, x_i - x_j]} \quad (3.17)$$

Possible use cases in machine learning are the applications in special LVQ variants [54] or generalized PCA (principle component analysis) variants [55], [56].

Finally, we shortly consider kernels as one of the most important proximity concepts in machine learning. Without claiming to be complete, we will discuss the concept of kernels, which can also be understood as a special variant of inner products.

For kernels we assume the existence of a function $\Phi_{\mathcal{H}} : \mathcal{X} \rightarrow \mathcal{H}$ which maps the data into a high-dimensional (maybe indefinite) Hilbert space which is related to the inner product

$\langle \Phi_{\mathcal{H}}(x_i), \Phi_{\mathcal{H}}(x_j) \rangle_{\mathcal{H}}$. A function $\mathfrak{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a kernel⁵ related to $\Phi_{\mathcal{H}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ if

$$\mathfrak{K}(x_i, x_j) = \langle \Phi_{\mathcal{H}}(x_i), \Phi_{\mathcal{H}}(x_j) \rangle_{\mathcal{H}} \quad (3.18)$$

hold for all $x_i, x_j \in \mathcal{X}$. In an analog way we can define kernel functions for SIP spaces [57], related to semi-inner products, and for IIP spaces [53] related to indefinite inner products. In the following, we only consider kernel functions related to inner products.

In case of a given feature map $\Phi_{\mathcal{H}}$ and feature space $\mathcal{H}$ associated with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ it is possible to specify the kernel function explicitly using equation (3.18). One advantage of kernels, however, lies in the fact that it is possible to formulate conditions for the function $\mathfrak{K}$ which, even without knowledge of $\Phi_{\mathcal{H}}$ and $\mathcal{H}$, ensure that $\mathfrak{K}$ is a kernel in the above sense:

Let us start with a property that is easy to check, especially for finite data. A Kernel function is called positive semidefinite (psd)⁶ if $\forall n \in \mathbb{N}, \forall \alpha_1, \dots, \alpha_n \in \mathbb{R}, \forall x_1, \dots, x_n \in \mathcal{X}$

$$\sum_{i,j} \alpha_i \alpha_j \mathfrak{K}(x_i, x_j) \geq 0 \quad (3.19)$$

⁵We restrict ourselves again to the most common case in machine learning by defining real valued Kernel function instead complex valued.

⁶We follow here the typical mathematical notation for positive definite and semidefinite property. In machine learning literature positive semidefinite kernels are typically referred as positive kernel or positive definite kernel. Further, positive definite kernels (in our notation) are referred as strictly positive (see e.g. [11]).

is valid, and positive definite (pd) if for mutually distinct $x_1, \dots, x_n$

$$\sum_{i,j} \alpha_i \alpha_j \mathfrak{K}(x_i, x_j) \geq 0 \wedge \sum_{i,j} \alpha_i \alpha_j \mathfrak{K}(x_i, x_j) = 0 \Leftrightarrow \alpha_i = \alpha_j = 0 \quad (3.20)$$

holds. Equation (3.19) looks impractical but for finite data the use of the Gram matrix

$$\mathbf{K} = \mathfrak{K}(x_i, x_j), x_i, x_j \in \mathcal{X} \quad (3.21)$$

results in a simplification to the statement that the kernel function is psd if the corresponding Gram matrix is psd. The same holds for pd kernel functions. Hence, any psd matrix can be seen as a finite Gram matrix of a psd kernel function. This observation we will use later.

Testing the Gram matrix to be psd can be easily done by checking whether the eigenvalues of the matrix are nonnegative or positive in the pd case because the Gram matrix is symmetric.

A further relevant attribute for kernel functions is the reproducing property. For this, let $\mathbb{R}^{\mathcal{X}} : \{f : \mathcal{X} \rightarrow \mathbb{R}\}$ be the space of continuous functions and $\Phi_{\mathcal{H}} : \mathcal{X} \rightarrow \mathbb{R}^{\mathcal{X}}$. The kernel $\mathfrak{K}$ is called reproducing if

$$\mathfrak{K}(\cdot, x) \in \mathbb{R}^{\mathcal{X}}, \forall x \in \mathcal{X} \quad (3.22)$$

and

$$f(x) = \langle f, \mathfrak{K}(\cdot, x) \rangle_{\mathbb{R}^{\mathcal{X}}}, \forall f \in \mathbb{R}^{\mathcal{X}} \text{ and } x \in \mathcal{X} \quad (3.23)$$

are both valid. Moreover, it can be stated that a kernel is reproducing iff it is a psd kernel [58], [59]. The reproducing property (3.22) results in two special characteristics for the kernel $\mathfrak{K}$ and the kernel mapping $\Phi_{\mathcal{H}}$.

First, the kernel mapping can be represented as follows

$$\Phi_{\mathcal{H}} : x \rightarrow \mathfrak{K}(\cdot, x) \quad (3.24)$$

which means that $\mathfrak{K}(\cdot, x)$ is the corresponding function for x under the mapping $\Phi_{\mathcal{H}}$ i.e. the respective reproducing kernel Hilbert space $\mathcal{H}$ is the space of linear functions due the linearity of inner products. Secondly, for two corresponding functions $\mathfrak{K}(\cdot, x_i)$ and $\mathfrak{K}(\cdot, x_j)$, we have

$$\langle \mathfrak{K}(\cdot, x_i), \mathfrak{K}(\cdot, x_j) \rangle_{\mathcal{H}} = \mathfrak{K}(x_i, x_j),$$

realizing (3.18).

For kernel functions associated with SIPs one can find in an analogous way so-called reproducing kernel Banach spaces [57].

Formally, kernels are generalizations of inner products (SIPs, IIPs) in the sense that every inner product (SIP, IIP) can be seen as a kernel where the mapping $\Phi_{\mathcal{H}}$ is the identity. The fact that kernel functions implicitly calculate inner products (SIP, IIP) in a possibly unknown Hilbert space (SIP space, IIP space) does not lead to the fact that all properties of the inner product (SIP, IIP) being transferred to the kernel. Generally kernel functions are not linear and, hence, violate the linear property of IP, SIP, IIP because the map $\Phi_{\mathcal{H}}$ is usually non-linear.

Furthermore, the dissimilarity obtained by

$$\begin{aligned} \mathfrak{d}_{\Phi_{\mathcal{H}}}(x_i, x_j) &= \sqrt{\langle \Phi_{\mathcal{H}}(x_i) - \Phi_{\mathcal{H}}(x_j), \Phi_{\mathcal{H}}(x_i) - \Phi_{\mathcal{H}}(x_j) \rangle_{\mathcal{H}}} \quad (3.25) \\ &= \sqrt{\mathfrak{K}(x_i, x_i) - 2\mathfrak{K}(x_i, x_j) + \mathfrak{K}(x_j, x_j)} \end{aligned}$$

applying general kernels ((3.18)) is only a semi-metric in general. However, for psd kernel functions it is a metric [59].

Possible applications for kernels and dissimilarities based on kernels are SVM [11], kernelized LVQ methods [35], [47], [60], [61] for classification learning and kernel PCA [11].

3.2. Equivalence of Proximities

In the last section, we have introduced proximities in general. We discussed various concepts of proximities and their relation to each other. The question that arises is: What are the criteria that given proximities are equivalent in the context of machine learning or for a given particular algorithm. Obviously, it is clear that proximities are (mathematically) equivalent if they result in identical values for the same arguments. For nearest prototype methods this equivalence seems to be too strict due to the discrete character of nearest prototype principle. Implicitly, a ranking takes place for these methods. We will illustrate the problem by the following example: we consider the dissimilarity

$$\mathfrak{d}_{\text{Gauss}}(x_i, x_j) = \sqrt{2 - 2\mathfrak{K}_{\text{Gauss}}(x_i, x_j)} \quad (3.26)$$

based on the Gaussian kernel

$$\mathfrak{K}_{\text{Gauss}}(x_i, x_j) = \exp\left(-\frac{\mathfrak{d}_E^2(x_i, x_j)}{2\sigma^2}\right) \quad (3.27)$$

and compare with the Euclidean distance. We illustrate the behavior of both dissimilarities in the $\mathbb{R}^2$ in (Figure 3.1), whereby x_i is fixed in the origin and the x_j values vary in the plane. For each point x_j we visualize

the corresponding dissimilarity to x_i by color coding. Obviously, both dissimilarities are not equivalent, i.e. $\mathfrak{d}_E(x_i, x_j) \neq \mathfrak{d}_{\text{Gauss}}(x_i, x_j)$ holds for almost all data points x_j .

But there is a structural relation between both in the following senses: If we take a finite number of arbitrary data points x_j and sort them in such a way that $x_1, x_2, \dots, x_k$ implies $\mathfrak{d}_E(x_i, x_1) \leq \mathfrak{d}_E(x_i, x_2) \leq \dots \leq \mathfrak{d}_E(x_i, x_k)$ then it follows immediately that $\mathfrak{d}_{\text{Gauss}}(x_i, x_1) \leq \mathfrak{d}_{\text{Gauss}}(x_i, x_2) \leq \dots \leq \mathfrak{d}_{\text{Gauss}}(x_i, x_k)$ holds. Thus the ranking order is preserved. This can easily be verified looking at the following relation between the two dissimilarities

$$\mathfrak{d}_{\text{Gauss}}(x_i, x_l) = f(\mathfrak{d}_E(x_i, x_j)) \quad (3.28)$$

$$\text{with } f(\xi) = \sqrt{2 - 2 \exp\left(-\frac{\xi^2}{2\sigma^2}\right)} \quad (3.29)$$

where $\sigma > 0$ holds. The function $f(\xi)$ is monotonically increasing (Figure 3.2) for ξ and, hence, the stated ordering preservation holds. This example leads to the following general considerations: Sorting of data points regarding a (dis-)similarity in the described way is based on the ranking of these data points. Formally, we can assign a rank value to each data point according to the following rule (introduced in [17])

$$\rho_{\mathfrak{d}}(x_i, x_j) = \sum_{l=1}^k \mathbf{H}(\mathfrak{d}(x_i, x_j) - \mathfrak{d}(x_i, x_l)) \quad (3.30)$$

where $\mathbf{H}$ is the Heaviside function (2.6). Let us assume the sorting of the data points $x_1, x_2, \dots, x_k$ is strictly in the sense that $\mathfrak{d}(x_i, x_1) < \mathfrak{d}(x_i, x_2) < \dots < \mathfrak{d}(x_i, x_k)$ holds than $\rho_{\mathfrak{d}}(x_i, x_1) = 1$ and, therefore, x_1

Figure 3.1.: Visualization of the dissimilarity value between the origin and all other points of the (restricted) plane. Left: regarding to the Euclidean distance, Right: regarding to the Gaussian kernel dissimilarity.

Figure 3.2.: The function $f(\xi) = \sqrt{(2 - 2 \exp(-\frac{\xi^2}{2\sigma^2}))}$.

is the nearest data point to x_i . The data point x_2 has rank value two according to $\rho_{\mathfrak{d}}(x_i, x_2) = 2$ and is the second nearest data point to x_i , and so on.

For monotonic functions, the ranking is preserved. Hence, if ML methods are based on a discretization together with monotonic functions, those functions can be seen as equivalent. We explain this idea more detailed for similarity and dissimilarity measures:

Def. 2 Rank equivalence for similarities and dissimilarities

Let M be a set of data items and $\mathcal{X} \subseteq M, \Theta \subseteq M$ non empty subsets of M . Two dissimilarities $\mathfrak{d}$ and $\hat{\mathfrak{d}}$ in M are said to be rank-equivalent for the data set X with respect to the set Θ if $\forall x_i, x_j \in \mathcal{X}$ and $\forall \theta_k, \theta_l \in \Theta$ the following relation hold

$$\begin{aligned} \mathfrak{d}(x_i, \theta_k) < \mathfrak{d}(x_j, \theta_l) & \quad \text{iff} \quad \hat{\mathfrak{d}}(x_i, \theta_k) < \hat{\mathfrak{d}}(x_j, \theta_l) \\ \mathfrak{d}(x_i, \theta_k) = \mathfrak{d}(x_j, \theta_l) & \quad \text{iff} \quad \hat{\mathfrak{d}}(x_i, \theta_k) = \hat{\mathfrak{d}}(x_j, \theta_l). \end{aligned}$$

As short hand notations we use for the rank-equivalence $\mathfrak{d} \stackrel{\cdot}{=}_{\mathcal{X}, \Theta} \hat{\mathfrak{d}}$. Two similarities are said to be rank-equivalent ($\mathfrak{s} \stackrel{\cdot}{=}_{\mathcal{X}, \Theta} \hat{\mathfrak{s}}$) if the following relation holds

$$\begin{aligned} \mathfrak{s}(x_i, \theta_k) < \mathfrak{s}(x_j, \theta_l) & \quad \text{iff} \quad \hat{\mathfrak{s}}(x_i, \theta_k) < \hat{\mathfrak{s}}(x_j, \theta_l) \\ \mathfrak{s}(x_i, \theta_k) = \mathfrak{s}(x_j, \theta_l) & \quad \text{iff} \quad \hat{\mathfrak{s}}(x_i, \theta_k) = \hat{\mathfrak{s}}(x_j, \theta_l). \end{aligned}$$

A dissimilarity and a similarity are said to be rank-equivalent ($\mathfrak{d} \stackrel{\cdot}{=}_{\mathcal{X}, \Theta} \mathfrak{s}$) if

$$\begin{aligned} \mathfrak{s}(x_i, \theta_k) < \mathfrak{s}(x_j, \theta_l) & \quad \text{iff} \quad \mathfrak{d}(x_i, \theta_k) > \mathfrak{d}(x_j, \theta_l) \\ \mathfrak{s}(x_i, \theta_k) = \mathfrak{s}(x_j, \theta_l) & \quad \text{iff} \quad \mathfrak{d}(x_i, \theta_k) = \mathfrak{d}(x_j, \theta_l). \end{aligned}$$

holds.

Remark 1 *Formally speaking, the rank-equivalence induces an equivalence relation on the set of all similarities and dissimilarities. Let Ξ be the set of all similarities $\mathfrak{s}$ and dissimilarities $\mathfrak{d}$. For the relation $\stackrel{\cdot}{\sim}_{\mathcal{X},\theta}$ the following properties are true*

$$\begin{aligned} \text{reflexivity} \quad & d \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} d, \forall d \in \Xi \\ \text{symmetry} \quad & d \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} \hat{d} \Leftrightarrow \hat{d} \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} d, \forall d, \hat{d} \in \Xi \\ \text{transitivity} \quad & d \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} \hat{d} \wedge \hat{d} \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} \bar{d} \Rightarrow d \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} \bar{d}, \forall d, \hat{d}, \bar{d} \in \Xi. \end{aligned}$$

Furthermore, we can use the equivalence class: $[\mathfrak{d}] = \{d \in \Xi \mid \mathfrak{d} \stackrel{\cdot}{\sim}_{\mathcal{X},\theta} d\}$ as a short notation for all (dis)-similarities which are rank-equivalent to the given dissimilarity $\mathfrak{d}$. The equivalence class for a given similarity is defined in the same way.

Remark 2 *Two dissimilarities $\mathfrak{d}, \hat{\mathfrak{d}}$ are rank-equivalent iff there exist a monotonously increasing function f , such that $\hat{\mathfrak{d}} = f(\mathfrak{d})$ holds.*

Analogously, two similarities $\mathfrak{s}, \hat{\mathfrak{s}}$ are said to be rank-equivalent iff there exist a monotonously increasing function f , such that $\hat{\mathfrak{s}} = f(\mathfrak{s})$ holds.

A given dissimilarity $\mathfrak{d}$ and a similarity $\mathfrak{s}$ are rank-equivalent iff there exist a monotonously decreasing function f , such that $\mathfrak{d} = f(\mathfrak{s})$ or $\mathfrak{s} = f(\mathfrak{d})$ holds.

We have already considered an example of such an equivalence: The Euclidean distance $\mathfrak{d}_E$ and the distance $\mathfrak{d}_{\text{Gauss}}$ based on the Gaussian kernel (3.26) are rank-equivalent because of the monotonic function (3.28). Fur-

thermore, even the Gaussian kernel $\mathfrak{K}_{\text{Gauss}}$, considered as quasi-similarity⁷, is rank-equivalent to the Euclidean distance $\mathfrak{d}_E$ according to the monotonic function

$$f(\xi) = \exp\left(-\frac{\xi^2}{2\sigma^2}\right). \quad (3.31)$$

In this sense, $\mathfrak{d}_{\text{Gauss}}$ and $\mathfrak{K}_{\text{Gauss}}$ are both in the equivalence class of the Euclidean distance $[\mathfrak{d}_E]$. The squared Euclidean distance $\mathfrak{d}_E^2$ is also in this equivalence class, since the square function is also monotonously increasing for positive arguments.

The fact that two dissimilarities are in the same equivalence class does not mean that they share the same properties (regarding the taxonomy given in section 3.1). Examples for that are the Euclidean distance and the square Euclidean distance: The former is a metric and the latter one is only a quasi-metric because the triangle inequality is violated. Moreover, if we think of general monotonically increasing functions for f also other properties like reflexivity, normalized consistency or non-negativity could be different. The introduced equivalence class has, therefore, only to be seen as an equality in the sense of ranks but not in the sense of the characteristics of (dis)-similarity measures.

A direct consequence of the two remarks can be drawn for several ML algorithms: There is no difference in classification behavior for the k-NN if used with rank-equivalent (dis)-similarities because k-NN has a only rank based decision. Specifically, there is no benefit if we replace the Euclidean distance by the dissimilarity based on the Gaussian ker-

⁷In general, kernels are not similarities as already discussed. However, one can easily proof that the Gaussian kernel is a quasi-similarity.

nel or the Gaussian kernel as a similarity measure. The same holds for every nearest neighbor algorithm (like LVQ algorithms) or unsupervised VQ (like Self Organizing maps, Neural Gas,...) in the recall phase after training. Meaning, if we have fixed prototypes after training and want to classify a new unlabeled data point, we are only interested in finding that prototype which is the nearest one to this data point. For fixed prototypes, those systems does not benefit from changing the dissimilarity to any other from the same equivalence class because the ranking of the prototypes regarding a given data point remains unchanged. However, the vector quantizer depends on the dissimilarity measures (also from the same equivalence class) during the training. In unsupervised vector quantization the prototypes are distributed regarding the magnification law according to the data density [14] in dependence on the data distribution. Generally the magnification of the quantizer depends on the used dissimilarity measure [15]. Thus, the choice of the dissimilarity measure triggers the placing of the prototypes during the training. However, we can replace the dissimilarity measure after training by any other rank-equivalent measure with lower computational costs.

If we want to compare proximities in terms of their rank-equivalence, a criterion would be helpful, which measures the the degree of rank-inequivalence between them in a suitable way. For this purpose, we first introduce an abbreviation, which we will always use in the following.

Def. 3 *Let $\mathcal{X}$ be a finite set of of arbitrary data objects x_k and $\mathbf{p}(x_i, x_j)$ an corresponding proximity measure. The matrix $\mathbf{P}$ with the entries $\mathbf{P}(i, j) = \mathbf{p}(x_i, x_j)$ is denoted as proximity matrix. We will call a data set **proximity data** if the information about the data is only given in terms of the proximity matrix.*

Regarding the last definition we will denote a given similarity, dissimilarity or kernel matrix as $\mathbf{S}$, $\mathbf{D}$ and $\mathbf{K}$, respectively.

Def. 4 Suppose $\mathcal{X}, \Theta \subseteq M$ to be finite and nonempty subsets of the set of data items M . The respective cardinalities are $N_{\mathcal{X}} = |\mathcal{X}|$ and $N_{\Theta} = |\Theta|$.

Further let $\mathbf{p}$ be a proximity measure on M .

The matrix $\mathbf{R}_{\mathbf{p}}$ with the entries $\mathbf{R}_{\mathbf{p}}(i, j) = \rho_{\mathbf{p}}(x_i, \theta_j)$ is denoted as **proximity rank matrix (PRM)** regarding $\mathbf{p}$ whereby the ranks $\rho_{\mathbf{p}}$ are evaluated as

$$\rho_{\mathbf{p}}(x_i, \theta_j) = \begin{cases} \sum_{l=1}^{N_{\Theta}} \mathbf{H}(\mathfrak{d}(x_i, \theta_j) - \mathfrak{d}(x_i, \theta_l)) & \text{iff } \mathbf{p} \text{ is a dissimilarity} \\ \sum_{l=1}^{N_{\Theta}} \mathbf{H}(\mathfrak{s}(x_i, \theta_l) - \mathfrak{s}(x_i, \theta_j)) & \text{iff } \mathbf{p} \text{ is a similarity.} \end{cases}$$

Each element $\mathbf{R}_{\mathbf{p}}(i, j)$ of the proximity rank matrix is simply the number of objects $\theta \in \Theta$ which have a higher similarity (lower dissimilarity) to the object x_i as θ_j .

Def. 5 Let $\mathbf{R}_{\mathbf{p}}$ and $\mathbf{R}_{\hat{\mathbf{p}}}$ be PRMs regarding two proximity measures $\mathbf{p}$ and $\hat{\mathbf{p}}$. The **absolute rank-equivalence measure (ARE)** is defined as

$$\tilde{v}_{\mathcal{X}, \Theta}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{i=1}^{N_{\mathcal{X}}} \sum_{j=1}^{N_{\Theta}} |\mathbf{R}_{\mathbf{p}}(i, j) - \mathbf{R}_{\hat{\mathbf{p}}}(i, j)|. \quad (3.32)$$

The absolute rank-equivalence measure quantifies the amount of differences in the regarding rank matrices. If $\mathcal{X} = \Theta$ is valid we can simply denote the absolute rank-equivalence measure as $\tilde{v}_{\mathcal{X}}(\mathbf{p}, \hat{\mathbf{p}})$.

Def. 6 Given an absolute rank-equivalence measure $\tilde{v}_{\mathcal{X}}(\mathbf{p}, \hat{\mathbf{p}})$, the **nor-**

malized rank-equivalence measure (NRE) is defined as

$$\vartheta_{\mathcal{X}}(\mathbf{p}, \hat{\mathbf{p}}) = \frac{\tilde{\vartheta}_{\mathcal{X}}(\mathbf{p}, \hat{\mathbf{p}})}{c_{\mathcal{X}}} \quad (3.33)$$

where

$$c_{\mathcal{X}} = \begin{cases} \frac{N_{\mathcal{X}}^2}{2} & , \text{ if } N_{\mathcal{X}} \text{ is even} \\ \frac{(N_{\mathcal{X}}-1)(N_{\mathcal{X}}+1)}{2} & , \text{ if } N_{\mathcal{X}} \text{ is odd} \end{cases}$$

is a normalization constant⁸.

With the last definition, $\vartheta_{\mathcal{X}}$ becomes independent from $N_{\mathcal{X}}$, which allows a better comparison of the quantity for different sets $\mathcal{X}$.

Remark 3

1) In prototype based algorithms, thereof Θ is identified with the prototype set whereas $\mathcal{X}$ is the set of data samples. In this sense, the rank-equivalence measure describes the difference of a given nearest prototype based classifier for different (dis)-similarities.

2) The case $\mathcal{X} = \Theta$ is of interest if we want to determine the effect of the (dis)-similarities on a data set without any classifier.

Remark 4 The choice of the measure to judge rank-equivalence is, of course, not unique. One could also use the Kendall- τ -distance [62], which is a measure for differences in ranking lists based on the Kendall- τ -correlation. However, we will use the measure defined in Definition 5 because useful extensions can be directly derived from it.

In Definition 5 we consider the ranks of all data points of $\mathcal{X}$ to each of the prototypes in Θ . This might be too restrictive in the sense that the

⁸The normalization constant can be verified by taking the worst case where the ranks are in the exact opposite order.

preservation of ranks in a local data area could be sufficient for preserving the behavior of a classifier or a vector quantizer. A well-known example is the k -NN which performs equivalently if the k next neighbors do not change. Therefore, we introduce a k -ARE measure, which only involves the first k ranks.

Def. 7 Let $\mathbf{R}_{\mathbf{p}}$ and $\mathbf{R}_{\hat{\mathbf{p}}}$ be two PRMs regarding two proximity measures $\mathbf{p}$ and $\hat{\mathbf{p}}$. The **k absolute rank-equivalence measure (k -ARE)** is defined via

$$\tilde{\vartheta}_{\mathcal{X},k}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{i=1}^{N_{\mathcal{X}}} \sum_{j=1}^{N_{\mathcal{X}}} \varsigma_k(i, j)$$

with

$$\varsigma_k(i, j) = \begin{cases} |\mathbf{R}_{\mathbf{p}}(i, j) - \mathbf{R}_{\hat{\mathbf{p}}}(i, j)| & \text{if } \mathbf{R}_{\mathbf{p}}(i, j) \leq k \\ 0 & \text{else} \end{cases}$$

The corresponding normalized k -ARE is

$$\vartheta_{\mathcal{X},k}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{i=1}^{N_{\mathcal{X}}} \sum_{j=1}^{N_{\mathcal{X}}} \frac{\varsigma_k(i, j)}{k(N_{\mathcal{X}} - k)} \text{ for } k < \frac{N_{\mathcal{X}}}{2}$$

Unlike the ARE, the k -ARE measurement is not longer symmetrical. Thus, if we consider different proximities, one proximity $\mathbf{p}$ has to serve as a reference proximity for the comparison with all other proximities $\hat{\mathbf{p}}$ to guarantee comparability.

3.3. Handling of Proximity Data in nearest neighbor approaches

In subsection 3.2 we introduced the term proximity data. For the following we restrict ourselves to this type of data. Further we assume that the only information available about the data are the pairwise proximities of the data samples, i.e. the proximity data is given in form of a (dis)-similarity or a kernel matrix. These (dis)similarities may represent structure kernels for graphs, trees, alignment distances, string distances or other, such that also non-vectorial data can be compared (see e.g.: [13], [42], [47], [63]). In the following we want to discuss briefly, which of the (prototype based) algorithms introduced in chapter 2 can handle proximity matrices and what properties thereof are important for this task.

3.3.1. General Aspects and Motivating Example

First, we start considering the relations between the matrix and an the underlying measure. Let $\mathbf{P}$ be the $n \times n$ proximity matrix for the finite set $\{x_1, \dots, x_n\}$ of data examples with the entries $\mathbf{P}(i, j) = \mathbf{p}(x_i, x_j)$ as introduced above. Obviously, the matrix properties result from the properties of the (assumed) underlying proximity measure (see Table 3.1). For example, the proximity matrix is symmetric if the underlying proximity measure is symmetric. Thus we can formulate the properties of Table 3.1 also in terms of matrix properties. Moreover, we can classify the resulting proximity matrices in the same way as in Table 3.3. However, we must be careful with the statements because a proximity matrix is always created only for a finite set of data. The only evaluation of

the proximity matrix can lead to conclusions, which contradict the assumed but frequently unknown general data proximity measure. To see this, let us take the following simple example. We take the set of data $\mathcal{X} = \{x_1 = (1, 1)^T, x_2 = (1, 3)^T, x_3 = (2, 2)^T\} \subseteq \mathbb{R}^2$ and suppose the squared Euclidean distance $\mathfrak{d}_E^2$. The regarding dissimilarity matrix $\mathbf{D}$ is found to be

$$\mathbf{D} = \begin{pmatrix} 0 & 4 & 2 \\ 4 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}.$$

This dissimilarity matrix fulfills, among other properties, the triangle inequality

$$\begin{aligned} \mathbf{D}(1, 2) + \mathbf{D}(2, 3) &\geq \mathbf{D}(1, 3) \\ 6 &\geq 2 \\ \mathbf{D}(1, 3) + \mathbf{D}(3, 2) &\geq \mathbf{D}(1, 2) \\ 4 &\geq 4 \\ \mathbf{D}(2, 1) + \mathbf{D}(1, 3) &\geq \mathbf{D}(2, 3) \\ 6 &\geq 2 \end{aligned}$$

Of course, the triangle inequality is violated in general by the *squared* Euclidean distance. Thus, the only consideration of the matrix would be misleading. However, if we extend the data set by $x_4 = (1, 5)^T$, the

resulting dissimilarity matrix

$$\hat{\mathbf{D}} = \begin{pmatrix} 0 & 4 & 2 & 16 \\ 4 & 0 & 2 & 4 \\ 2 & 2 & 0 & 10 \\ 16 & 4 & 10 & 0 \end{pmatrix}$$

violates the triangle inequality too, because of

$$\begin{aligned} \mathbf{D}(1,2) + \mathbf{D}(2,4) &\not\geq \mathbf{D}(1,4) \\ 8 &\not\geq 16. \end{aligned}$$

As mentioned before, the underlying data dissimilarity is not known or cannot be assumed to fulfill certain mathematical properties. For example, if the proximity values arising from questionnaires (see as example the introduced patrol data from section 3.1), mathematical properties are at least difficult to derive. For those cases, we can only make statements regarding the dissimilarity matrix obtained from a finite number of data samples and have to be prepared that new data could destroy these properties. This is particularly interesting for the post-training phase, when new data arrive.

3.3.2. Conversion of Proximity Matrices

(Dis)-Similarity matrices In the context of finite data, the transformation between dissimilarity and similarity matrices is quite simple. Let $\mathbf{D}$ be a finite dissimilarity matrix and $d_{\max} \geq \max_{i,j} \mathbf{D}(i,j)$. A rank-

equivalent similarity matrix can be found via

$$\mathbf{S} = d_{\max} \mathbf{1} - \mathbf{D}$$

where $\mathbf{1}$ is the matrix of the same size as $\mathbf{D}$ filled with ones. The other way around we can find a rank-equivalent dissimilarity matrix $\mathbf{D}$ for a given similarity matrix $\mathbf{S}$ by

$$\mathbf{D} = s_{\max} \mathbf{1} - \mathbf{S}$$

with $s_{\max} \geq \max_{i,j} \mathbf{S}(i, j)$. These transformations lead to rank-equivalent proximities and keep the properties according to Table 3.1. For example, a basic dissimilarity becomes a basic similarity and a metric become a Minkowsky-like similarity.

Kernel matrices The transformation from a psd kernel matrix $\mathbf{K}$ into a dissimilarity matrix $\mathbf{D}$ follows directly from equation (3.25)

$$\begin{aligned} \Upsilon : \mathbf{K} &\rightarrow \mathbf{D} \\ \text{such that } \mathbf{D}(i, j) &= \sqrt{\mathbf{K}(i, i) - 2\mathbf{K}(i, j) + \mathbf{K}(j, j)}. \end{aligned} \tag{3.34}$$

Correspondingly, a dissimilarity $\mathbf{D}$ is obtained from a kernel by double centering (see e.g. [13]), i.e.

$$\begin{aligned} \Psi : \mathbf{D} &\rightarrow \mathbf{K} \\ \text{such that } \mathbf{K}(i, j) &= -\frac{1}{2} (\mathbf{D}(i, j)^2 - D(:, j) - D(i, :) + D(:, :)) \end{aligned} \tag{3.35}$$

$$\begin{aligned}
\text{and where: } D(:, j) &= \frac{1}{n_d} \sum_{i=1}^{n_d} \mathbf{D}(i, j)^2 \\
D(i, :) &= \frac{1}{n_d} \sum_{j=1}^{n_d} \mathbf{D}(i, j)^2 \\
D(:, :) &= \frac{1}{n_d^2} \sum_{j=1}^{n_d} \sum_{i=1}^{n_d} \mathbf{D}(i, j)^2
\end{aligned}$$

For the composition of Υ and Ψ the equality $\Upsilon \circ \Psi = I$ holds, whereby I is the identical mapping. The reverse composition $\Psi \circ \Upsilon$ yields the identical mapping iff the data is centered⁹. By equation (3.35) we can calculate a corresponding kernel matrix for an arbitrary proximity matrix generated by a metric. The other way around we can find a corresponding proximity matrix (being a metric) to a given psd kernel matrix applying (3.34) under the centering restriction. In literature a non-psd proximity matrix is often revered as indefinite kernel (see e.g. [53]). This notation is, however, somewhat misleading: First of all, given only the proximity matrix $\mathbf{P}$ without knowledge of the underlying data and proximity measure, we cannot relate $\mathbf{P}$ to the kernel equation (3.18). Yet, a psd kernel corresponds allways to a psd Gram-matrix and, vice versa, a psd Gram-matrix can be always interpreted as realization of a psd kernel for finite data. However, a finite non-psd proximity matrix can result from a wide range of proximity measures including indefinite inner products and dissimilarities. For illustration, let us consider an extreme example for this circumstance: We randomly sample 10 data points $x_i \in \mathbb{R}^2$ and calculate the proximity matrix $\mathbf{P}$ by the Euclidean distance (left Figure in 3.3). We further assume that someone does not know, how this proximity matrix was created. This person might evaluate the signature of the proximity matrix, which is $\Sigma_{\mathbf{P}} = (1, 9, 0)$ (middle Figure in 3.3) and, therefore, treat the proximity as indefinite kernel matrix

⁹Assume the data to be vectorial then the data is called centered if the mean of all data points is the origin.

K. Further this person calculates a corresponding dissimilarity matrix **D** via equation (3.36) which is a proper way under the assumption that **K** is a indefinite kernel. The resulting dissimilarity **D** (right Figure in 3.3) is obviously far away from the original used Euclidean distance. Not only that almost all dissimilarities are negative, also the resulting dissimilarity is not even a basic dissimilarity, because the minimum principle is heavily violated.

Of course, this toy example is an extreme case, which might be not happen in

Figure 3.3.: Toy example for visualizing the effect treating a dissimilarity as indefinite kernel might be unacceptable. The left figure shows the Euclidean distance for 10 randomly selected data points. The middle figure shows the regarding eigenvalues of the dissimilarity matrix. The right figure shows the corresponding dissimilarity treating the Euclidean distance as non psd kernel.

practice. Anyway, we should always be careful by treating an arbitrary non-psd matrix as an indefinite kernel and should have in mind that it could also be interpreted as a dissimilarity matrix. However, in the end, if **K** is an indefinite kernel matrix then we can generalize the equations (3.34),(3.35) such that

$$\mathbf{D}(i, j) = \mathbf{K}(i, i) - 2\mathbf{K}(i, j) + \mathbf{K}(j, j) \quad (3.36)$$

is a corresponding dissimilarity matrix to the indefinite kernel. The other way around, for a given general dissimilarity matrix

$$\begin{aligned}\mathbf{K}(i, j) &= -\frac{1}{2} (\mathbf{D}(i, j) - D(:, j) - D(i, :) + D(:, :)) \\ \text{with: } D(:, j) &= \frac{1}{n_d} \sum_{i=1}^{n_d} \mathbf{D}(i, j) \\ D(i, :) &= \frac{1}{n_d} \sum_{j=1}^{n_d} \mathbf{D}(i, j) \\ D(:, :) &= \frac{1}{n_d^2} \sum_{j=1}^{n_d} \sum_{i=1}^{n_d} \mathbf{D}(i, j)\end{aligned}$$

is a corresponding indefinite kernel [13], [47].

3.3.3. Preprocessing for non-psd Kernels

For kernels and inner products, we follow the usual characterization proposed in [13] for respective proximity matrices: Let $\mathbf{P}$ be a arbitrary symmetric proximity matrix. The signature of $\mathbf{P}$ is given by $\Sigma_{\mathbf{P}} = (p, q, z)$ whereby p is the number of positive eigenvalues of $\mathbf{P}$, q is the number of negative eigenvalues and z the number of zero eigenvalues. Hence, a symmetric proximity matrix is psd if $\Sigma_{\mathbf{P}} = (p, 0, z)$. Assuming that the underlying proximity measure is an inner product or psd kernel, the resulting proximity matrix is always psd. Further, given a proximity matrix, we can no longer distinguish between an inner product and a kernel function because the concept of linearity does not make sense on finite data. In this sense, we refer finite matrices as kernel matrices only.

The other way around, we can treat every psd proximity matrix as a valid psd kernel for a fixed finite set of data samples but keeping in mind that out of sample extensions may violate the psd property. Otherwise, we can always say that if the matrix is not psd then the underlying measure cannot be a psd kernel

function or an inner product.

If the proximity has to be a psd kernel matrix because the ML-algorithm requires this kind of data, there are several ways to preprocess the data to ensure respective properties [42], [53]. All those preprocessing methods have in common that they change the eigenvalue spectrum of the proximity matrix. More precisely let $\mathbf{P}$ be a proximity matrix and $\mathbf{P} = U\Lambda U^T$ the decomposition, where Λ is the diagonal matrix with all eigenvalues on the diagonal and U is the matrix containing the corresponding eigenvectors as columns. In case of negative eigenvalues, the following corrections are suggested:

Clipping sets all the negative eigenvalues to zero. This leads to the closest psd matrix to $\mathbf{P}$ regarding the Frobenius norm [64]. If we have only a few negative eigenvalues, which are relatively small, then the clipping can be seen as denoising step [65]. Let Λ_{clip} be the matrix where all the negative eigenvalues are set to zero, then the clipped proximity $\mathbf{P}_{\text{clip}}$ is given by

$$\mathbf{P}_{\text{clip}} = U\Lambda_{\text{clip}}U$$

as shown in [53] or, regarding the original proximity matrix, as

$$\mathbf{P}_{\text{clip}} = \mathbf{P}V_{\text{clip}}V_{\text{clip}}^T\mathbf{P} \quad (3.37)$$

where V_{clip} is

$$V_{\text{clip}} = U|\Lambda|^{-\frac{1}{2}}I_{\lambda>0}$$

with the diagonal matrix

$$I_{\lambda>0}(i, j) = \begin{cases} 1 & \text{if } i=j \text{ and } \lambda_i > 0 \\ 0 & \text{else} \end{cases}.$$

Equation (3.37) allows to apply the data preprocessing also to new data: Let $\mathbf{P}_{\text{clip}}$ be the preprocessed proximity matrix stemming from the data examples

$\{x_1, x_2, \dots, x_k\}$ with the corresponding transformation matrix V_{clip} . Further, let x_{k+1} be a new data point and $\tilde{\mathbf{P}}(1, j) = \mathbf{p}(x_{k+1}, x_j)$ the 1-by- k vector with proximities to all other data objects. Then we can simply evaluate $\tilde{\mathbf{P}}_{\text{clip}} = \tilde{\mathbf{P}} V_{\text{clip}} V_{\text{clip}}^T \mathbf{P}$. In this sense we are consistent because we can also pre-process new data in the same way as the original given samples. However, this pre-processing does not guaranty that the extended proximity matrix is still psd although the original matrix is psd as discussed before.

Flipping takes the absolute value of all eigenvalues instead of the eigenvalues itself. In the same way as before, we can describe the flipping as [53]

$$\mathbf{P}_{\text{flip}} = \mathbf{P} V_{\text{flip}} V_{\text{flip}}^T \mathbf{P} \quad (3.38)$$

where V_{flip} is

$$V_{\text{flip}} = U |\Lambda|^{-\frac{1}{2}}.$$

If the negative eigenvalues are not caused by noise, they may contain relevant information about the data [42]. In this case it is obviously a bad idea to ignore all the negative eigenvalues by clipping. Taking the absolute values instead preserves this information partially. In particular, flipping the eigenvalues corresponds to a transformation of data from an pseudo Euclidean space into a Euclidean space [13]. Equation (3.38) offers the same possibility for out of sample extension as for clipping.

Shifting changes the eigenvalues in such a way that the absolute value of the smallest eigenvalue is added to each eigenvalue. The shifting operation can be described via [53]

$$\mathbf{P}_{\text{shift}} = \mathbf{P} V_{\text{shift}} V_{\text{shift}}^T \mathbf{P} \quad (3.39)$$

where V_{shift} is

$$V_{\text{shift}} = U |\Lambda|^{-1} (\Lambda - \nu \mathbb{I}).$$

and $\mathbb{I}$ is the identity Matrix with $\nu = \min(\min_i(\lambda_i), 0)$. The shifting operator only changes the self-proximities $\mathbf{p}(x_i, x_i)$ by enhancing them. All other proximity values remain unchanged. As for clipping and shifting the out of sample extension is straight forward via equation (3.39).

Square of the eigenvalues takes λ_i^2 instead of λ_i . This operation can be described via [53]

$$\mathbf{P}_{\text{squared}} = \mathbf{P} V_{\text{squared}} V_{\text{squared}}^T \mathbf{P} \quad (3.40)$$

where V_{squared} is

$$V_{\text{squared}} = U(\Lambda\Lambda)^{-\frac{1}{2}}.$$

The new proximity matrix $\mathbf{P}_{\text{squared}}$ can also be calculated simply as

$$\mathbf{P}_{\text{squared}} = \mathbf{P}\mathbf{P}^T$$

which leads to the trivial observation that we can obviously construct a kernel matrix out of any (maybe non symmetric) proximity matrix. Moreover if we assume $\mathbf{P}$ to be a symmetric matrix and taking $\mathbf{P}(\cdot, i)$ as a feature vector for the i -th data object then $\mathbf{P}_{\text{squared}}$ is the Euclidean inner product regarding this feature representation, meaning: $\mathbf{P}_{\text{squared}}(i, j) = \langle \mathbf{P}(\cdot, i), \mathbf{P}(\cdot, j) \rangle_E$ [42]. As for the last cases the out of sample extension is straight forward via equation (3.40). In summary, it can be stated that there are several possibilities to adapt a given proximity to be psd if this is necessary for an algorithm. However, this may result in significant changes in the data structure relations, which, in the worst case, may reduce the performance of classification or vector quantization. One way to explore this change in the data information is to use the introduced rank measure.

For demonstration of the idea behind, we constructed a simple toy example. We artificially generated a similarity matrix $\mathbf{S}$ for 10 hypothetical data objects (see Figure 3.4 a). The respective proximity rank matrix (Figure 3.4 b) and the eigenvalues of $\mathbf{S}$ (Figure 3.4 c) are also shown. We calculate the signature of $\mathbf{S}$

as $\Sigma_{\mathbf{S}} = (8, 2, 0)$, i.e. $\mathbf{S}$ is not psd. The second row of the Figure 3.4 shows $\mathbf{S}$ after clipping (Figure 3.4 d), the regarding proximity rank matrix (Figure 3.4 e) and the rank differences regarding the original similarity (Figure 3.4 f). In the same way, the third row shows the results after flipping (Figure 3.4 g-i). For comparison we distorted the similarity $\mathbf{S}$ by some small random noise according to $\mathcal{N}(0, 0.05)$ for each similarity value in $\mathbf{S}$. The resulting similarity matrix, proximity rank matrix and rank differences are shown in the last row of Figure 3.4. We can calculate the normalized rank difference measure for all three cases: Comparing the original similarity $\mathbf{S}$ with the clipped similarity $\mathbf{S}_c$ we result in $\vartheta(\mathbf{S}, \mathbf{S}_c) = 0.154$. In the case of the flipped similarity we get $\vartheta(\mathbf{S}, \mathbf{S}_f) = 0.073$ and for the noise case we obtain $\vartheta(\mathbf{S}, \mathbf{S}_n) = 0.043$. As expected from the discussion about the data preprocessing via eigenvalue correction, the effect for clipping is larger than for flipping. Hence, zeroing the negative eigenvalues seems to be more restrictive than flipping their sign.

However, if a psd matrix is required, a respective preprocessing is mandatory. Yet, we should keep in mind that we always change the relations between the respective data by these kinds of preprocessing, as already discussed.

3.4. Classification Algorithms for Proximity Data

To close up this chapter we want to discuss shortly, which prototype methods introduced in chapter 2 can handle proximity data in general and which kind of proximity data is required in detail.

Support Vector Machines Using proximity data for SVMs is straight forward because for solving the dual optimization problem (2.14) we need only the pairwise inner products or kernel values. Comparable to k-NN we need neither a vectorial representation of the data exemplars nor we need an explicit formulation of the proximity measure. SVMs are restricted to work on inner products or psd kernels [11] such that the proximity matrix has to be psd. For

Figure 3.4.: Toy example for the influence of clipping, flipping and random noise to the data structure:

first row: **a)** original similarity matrix $\mathbf{S}$, **b)** rank matrix $\mathbf{R}_\mathbf{S}$, **c)** eigenvalue structure of $\mathbf{S}$;

second row: **d)** clipped similarity matrix $\mathbf{S}_c$, **e)** rank matrix $\mathbf{R}_{\mathbf{S}_c}$, **f)** rank difference matrix $\mathbf{R}_\mathbf{S} - \mathbf{R}_{\mathbf{S}_c}$;

third row: **g)** flipped similarity matrix $\mathbf{S}_f$, **h)** rank matrix $\mathbf{R}_{\mathbf{S}_f}$, **i)** rank difference matrix $\mathbf{R}_\mathbf{S} - \mathbf{R}_{\mathbf{S}_f}$;

fourth row: **j)** original similarity matrix with noise $\mathbf{S}_n$, **k)** rank matrix $\mathbf{R}_{\mathbf{S}_n}$, **l)** rank difference matrix $\mathbf{R}_\mathbf{S} - \mathbf{R}_{\mathbf{S}_n}$

non psd proximities a preprocessing via clipping, flipping,... is required [42], [53].

k-Nearest Neighbor The k-NN method naturally can handle proximity data. We need neither the data examples nor the proximity measure explicitly. For the classification of an arbitrary data point we only require the pairwise proximities for all training samples. Yet, the k-NN is nearest neighbor based such that the proximity matrix should be a (dis-)similarity. If the given proximity is a psd kernel or at least a indefinite kernel, we have to generate a proper dissimilarity regarding equation (3.34) or equation (3.36).

LVQ methods LVQ methods as introduced in chapter 2 rely on vectorial data such that a direct use of proximity data is not possible. Mainly there are two ways to overcome this problem: The first is vectorizing of the given proximity data. The second way is to generalize the LVQ methods in such a way that we do not need explicit formulation of data points and prototypes. This is frequently realized by relational and median variants.

We briefly consider vectorization approaches and relational variants of LVQ here. The median variants are investigated in detail in chapter 5 as one of the key topics of this thesis.

a) Vectorization If the given proximity matrix is a kernel matrix we can perform an Euclidean embedding [13]. For this purpose let $\mathbf{K}$ be the kernel matrix and $\mathbf{K} = U\Lambda U^T$ the decomposition regarding the eigenvalue matrix Λ and the eigenvector matrix U . The embedding of an arbitrary data object is given by

$$x_i = \sqrt{\Lambda(i, i)}U(., i)$$

whereby $U(., i)$ is the i-th column of the eigenvector matrix. For two embedded objects the relation $\langle x_i, x_j \rangle_E = \mathbf{K}(i, j)$ holds, which means that the embedding preserves the proximity values. Moreover, in the embedded space the corre-

sponding distance is the Euclidean distance. This embedding idea is generalizable to indefinite inner products, which leads to pseudo Euclidean embeddings [13]. Let here $\mathbf{K}$ be a indefinite kernel matrix with the signature $\Sigma = (p, q, z)$. Than we can decompose the matrix into $\mathbf{K} = U|\Lambda|^{\frac{1}{2}}I_{p,q}|\Lambda|^{\frac{1}{2}}U^T$ where $I_{p,q}$ is a diagonal matrix with the first p entries are 1 and the next q entries are -1 and the last z entries are zero. We can embedd an arbitrary data object by

$$x_i = \sqrt{|\Lambda(i, i)|}U(., i).$$

Furthermore, we can equip them with the indefinite inner product

$$\langle x_i, x_j \rangle_{p,q} = x_i I_{p,q} x_j^T.$$

Also in this indefinite case the equation $\langle x_i, x_j \rangle_{p,q} = \mathbf{K}(i, j)$ holds for arbitrary data objects.

Corresponding to the indefinite inner product we can define a dissimilarity via $\mathfrak{D}_{p,q} = \langle x_i - x_j, x_i - x_j \rangle_{p,q}$, which is in general only a very basic dissimilarity as discussed in section 3.1.2. Having both, the embedded data and the dissimilarity measure, we can use every preferred LVQ method we want.

Another way of vectorization is to treat the columns of a psd kernel matrix $\mathbf{K}(., i)$ as vectorial representation for the i -th data object x_i . This can be motivated by the fact that every psd kernel is a reproducing kernel (see discussion in section 3.1). In this sense $\mathbf{K}(., i)$ can be seen as a finite approximation of the function $\Phi_{\mathcal{H}}(x_i)$ in the Hilbert space $\mathcal{H}$.

For both outlined vectorization approaches, the representation of the data samples is very rich, i.e. the dimension of the embedding spaces goes along with the number of given data examples. Alternatives to that kind of embedding, which can lead to a lower dimensional representation, could be principle component analysis [66], multidimensional scaling [67] or t-stochastic neighbor embedding [7].

b) Relational Variants Working without an explicit embedding leads to the kernel, relational or median variants for LVQ. The basic idea for kernel and relational methods is to do the (pseudo)-Euclidean embedding only implicitly as described in [47]. According to this idea we assume the prototypes θ_j to be convex linear combination of data examples $\theta_j = \sum_{i=1}^{n_d} \lambda_{j,i} x_i$ with $\lambda_{j,i} \in [0, 1]$ and $\sum_{i=1}^{n_d} \lambda_{j,i} = 1$. Now the trick is that we can do the distance calculation for $\mathfrak{d}(x_i, \theta_j)$ only using the pairwise proximities. The formulas can be found using equation (3.15) after some calculations as

$$\mathfrak{d}(x_i, \theta_j) = \mathbf{K}(i, i) - 2 \sum_{l=1}^{n_d} \lambda_{j,l} \mathbf{K}(i, l) + \sum_{l=1}^{n_d} \sum_{k=1}^{n_d} \lambda_{j,l} \lambda_{j,k} \mathbf{K}(l, k)$$

if the proximity is a (indefinite)-kernel matrix and as

$$\mathfrak{d}(x_i, \theta_j) = \sum_{l=1}^{n_d} \lambda_{j,l} \mathbf{D}(i, l) - \frac{1}{2} \sum_{l=1}^{n_d} \sum_{k=1}^{n_d} \lambda_{j,l} \lambda_{j,k} \mathbf{D}(l, k)$$

if the proximity is a dissimilarity matrix. In the case that the proximity is a psd kernel matrix or a matrix stemming from a metric the so called *kernel* LVQ is obtained. Otherwise, if the proximity is only an indefinite kernel or a general non-metric dissimilarity, this results in the so-called *relational* methods [47].

The obvious advantage of this relational/kernel approaches is that we can create the classifier without having to calculate an explicit embedding. The prototypes can be trained indirectly via learning of the weights $\lambda_{j,i}$. However, one disadvantage is that the model is usually not sparse. In the worst case all coefficients $\lambda_{j,i}$ are greater than zero, which means that we need all data samples to represent the prototypes. Moreover, in this case, the interpretability of the prototypes as data samples is lost, which is one of the main advantages of LVQ methods [68]. A discussion on sparsity constraints to force better interpretability can be found, for example, in [69].

As mentioned previously, median variants of LVQ provide another alternative. To discuss them in detail, we first provide some mathematical preliminaries re-

garding the Expectation Maximization principle for use in Vector Quantization (Chapter 4) and thereafter turnover to the description and analysis of median LVQ (Chapter 5).

4. Expectation Maximization

Principle for Model Optimization Applicable for General Proximity Measures

The following chapter is intended to motivate the Expectation Maximization algorithm (EM) as optimization method for vector quantization. We give a short introduction to the commonly used ideas of EM, exemplarily shown for a well known example (Gaussian Mixture Models). Afterwards, we generalize this idea to develop a variant of the algorithm, which can handle non-probabilistic cost functions for vector quantization. To demonstrate the possibilities of this generalization, we end up with an example, where we use the EM to obtain a batch variant of the well known Neural Gas algorithm.

4.1. Expectation Maximization for probabilistic models

4.1.1. The Gaussian Mixture Model (GMM)

A frequently used method to model data sets is probabilistic modeling. There exist a huge number of ML-methods and ideas for several problems, like classification, data compression and others. Here, we focus on the problem that

we want to model the distribution of a given data set supposing that our data points are realizations of a random variable.

Under the assumption that the given data were generated by a single but unknown distribution we can make a hypothesis on this data distribution and fit the corresponding density function regarding to the given data. This hypothesis can be evaluated by statistical methods, for instance by the likelihood method [38], [70]. More formally, we suppose a finite data set $\mathcal{X} = \{x_1, x_2, \dots, x_{n_d}\}$ with $x_i \in \mathbb{R}^D$. Under the assumption that the given points are realizations of a random variable, which is iid (independent and identical distributed) the goal is to find the underlying probability density function $p_d(x)$ considering that $p_d(x)$ belongs to a family of parameterized densities $\{p_d(x|\phi)\}$ with the parameter vector ϕ . If the data is iid, we can specify a joint density function for the observations in the following form

$$p_j(x_1, x_2, \dots, x_{n_d}|\phi) = p_d(x_1|\phi) \cdot p_d(x_2|\phi) \cdot \dots \cdot p_d(x_{n_d}|\phi) .$$

From the perspective that we have a finite and fixed data set $\mathcal{X}$, the joint density is transferred into the so called likelihood function

$$\ell(\phi; \mathcal{X}) = \prod_{i=1}^{n_d} p_d(x_i|\phi) \quad (4.1)$$

which only depends on the parameter vector ϕ . Assuming that our data is the most likely observation regarding to the unknown distribution we search for a certain parameter setting $\hat{\phi}$, which maximizes the likelihood $\ell(\phi; \mathcal{X})$ i.e.

$$\hat{\phi} = \arg \max_{\phi \in \Phi} (\ell(\phi; \mathcal{X})) . \quad (4.2)$$

Often, it is easier to maximize the logarithmic likelihood function instead of (4.1). Especially, for distributions from the exponential family, this results to a closed solution for the unknown parameter estimation [38]. The corresponding

optimization problem

$$\hat{\phi} = \arg \max_{\phi \in \Phi} (\ell_{log}(\phi; \mathcal{X})) \quad (4.3)$$

with $\ell_{log}(\phi; \mathcal{X}) = \log(\ell(\phi; \mathcal{X}))$ has the same optimal solution regarding to the parameter vector θ as (4.2). This follows directly from the monotony properties of the logarithm.

One prominent example for log-likelihood methods is fitting of data points under the Gaussian assumption. There exist several motivations, why the Gaussian distribution is so important for data description. One is the central limit theorem, which states that a sum of a large number of iid random variables with finite mean and variance is approximately Gaussian distributed. This means in practice that if we assume that our given data comes from a superposition of a sufficiently high amount of random factors with unknown underlying distribution, a Gaussian model is the best choice (see e.g. [38]). The second main motivation is due to information theory. If we assume a random variable with finite mean and variance, then the Gaussian distribution achieves the maximum entropy over all distributions with known mean and variance (see e.g. [71]). This fact is known as the second Gibbs theorem.

The Gaussian (or Normal) density is given in general form as

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right) \quad (4.4)$$

where μ is the mean, Σ denotes the covariance matrix and D is the dimension of the data. Setting the joint distribution as

$$p_j(x_1, x_2, \dots, x_{n_d}|\phi) = \prod_{i=1}^{n_d} \mathcal{N}(x_i|\mu, \Sigma),$$

the log likelihood (with the natural logarithm: $\ln(\cdot)$) arises to

$$\ell_{\log}(\mu, \Sigma; \mathcal{X}) = -\frac{n_d D}{2} \ln(2\pi) - \frac{n_d}{2} \ln(|\Sigma|) - \frac{1}{2} \sum_{i=1}^{n_d} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) .$$

We can find the maximum likelihood estimation of the unknown mean vector and covariance matrix considering zero derivatives

$$\begin{aligned} \frac{\partial}{\partial \mu} \ell_{\log}(\mu, \Sigma; \mathcal{X}) &= 0 \\ \frac{\partial}{\partial \Sigma} \ell_{\log}(\mu, \Sigma; \mathcal{X}) &= 0 \end{aligned}$$

with the closed form solution

$$\begin{aligned} \mu_{ML} &= \frac{1}{n} \sum_{i=1}^{n_d} x_i \\ \Sigma_{ML} &= \frac{1}{n} \sum_{i=1}^{n_d} (x_i - \mu_{ML})(x_i - \mu_{ML})^T . \end{aligned}$$

Up to this point, we have an easy model with a closed form solution, which can describe data in a very intuitive way. But this model is restricted to data sets, which consist of only one cluster according to the iid condition. If the data form different dominant clusters like in Figure 4.1, this assumption is obviously violated and, hence, the modeling as single Gaussian density does not make sense.

One well known solution to handle such problems is the use of Gaussian mixture models (GMM) (see e.g. [6], [38]). The Gaussian Mixture Model (GMM) is a superposition of normal densities. Taking n_c single Gaussian into account, the resulting linear combination takes the form

$$p_m(x) = \sum_{k=1}^{n_c} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k) \quad (4.5)$$

Figure 4.1.: Artificial Gaussian. The data is drawn from four different Gaussian distributions with different prior probabilities.

where each mixture component has its own mean μ_k and covariance Σ_k . For the mixing coefficients π_k we require positivity ($\pi_k \geq 0$) and $\sum_{k=1}^{n_c} \pi_k = 1$ i.e. we suppose a convex linear combination. The main advantage of this approach is that for a reasonable number of Gaussians and proper choices of π_k , μ_k and Σ_k we can approximate many continuous densities with sufficient accuracy (see e.g. [38]). A second benefit of GMM is that $p_m(x)$ is still a density. To illustrate this in more detail, we suppose the following assumptions: We interpret

$$p(k) = \pi_k \quad (4.6)$$

as the prior probability to choose the k -th component of the mixture model. Further, we set the density

$$p(x_i|k) = \mathcal{N}(x_i|\mu_k, \Sigma_k) \quad (4.7)$$

as conditional probability of x given the k -th component. Then it follows directly from the properties of $p(x_i|k)$, $p(k)$ and the sum- and product rule for probabilities that

$$p_m(x_i) = \sum_{k=1}^{n_c} p(k)p(x_i|k) \quad (4.8)$$

Figure 4.2.: Distribution of Artificial Gaussian. The upper left figure shows data points sampled from the four single Gaussian densities. We have a red, a blue, a green and a black cluster. The upper right figure shows the contour plot of the marginal density $p_m(x)$. The lower shows the surface plot of the marginal density $p_m(x)$.

is a marginal density (Figure 4.2).

In summary, the GMM is a useful generalization for probabilistic modeling of data under Gaussian assumption. Moreover, the Gaussian mixture model can be interpreted as clustering technique (see e.g. [6], [38]). For this reason, we look at the posterior probability $p(k|x_i)$, which is the probability that a given data point x_i is assigned to the k -th component of the mixture. The posterior,

also known as responsibility $\gamma_k(x_i)$, follows directly from the Bayes Rule

$$\begin{aligned}\gamma_k(x_i) &\equiv p(k|x_i) \\ &= \frac{p(k)p(x_i|k)}{\sum_{l=1}^{n_c} p(l)p(x_i|l)} .\end{aligned}\tag{4.9}$$

If we interpret every single mixture component k as a single cluster generator with μ_k as center then the responsibility $\gamma_k(x_i)$ for a given data point x_i is the probability that data point x_i belongs to cluster k . This fact is exemplary illustrated in Figure 4.3. There we can see that the responsibilities for data points, which are far away from the mixture component, are approximately zero. Responsibilities for data points, which are inside the cluster component but far away from any other, are approximately one and, responsibilities for data points, which are located in the intersection of components have values in between zero and one, depending also on the prior probability of the component.

As for single distributed data we have to find the unknown parameters (π_k, μ_k, Σ_k) of the Gaussian mixture model with respect to the given data. One way to find them is to use a log-likelihood in analogy to (4.3). For short hand description we use the notations $\boldsymbol{\pi} = \{\pi_1, \dots, \pi_{n_c}\}$, $\boldsymbol{\mu} = \{\mu_1, \dots, \mu_{n_c}\}$ and $\boldsymbol{\Sigma} = \{\Sigma_1, \dots, \Sigma_{n_c}\}$. Starting with the density of the GMM (4.8), the log-likelihood function becomes

$$\begin{aligned}\ell_{\log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X}) &= \ln(p(\mathcal{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})) \\ &= \ln\left(\prod_{i=1}^{n_d} \sum_{k=1}^{n_c} p(k)p(x_i|k)\right) \\ &= \sum_{i=1}^{n_d} \ln\left(\sum_{k=1}^{n_c} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\right) .\end{aligned}\tag{4.10}$$

Compared to the likelihood for single Gaussian, this result is more complicate: Since the sum over the mixture components appears inside the logarithm, there does not exist a closed form solution for the parameter estimation (see e.g. [38]).

Figure 4.3.: Visualization of the responsibilities $\gamma_k(x_i)$ for Mixture of Gaussian for an artificial data set containing four Gaussian cluster. The upper left figure shows the responsibility values for the blue cluster, the upper right for the green cluster, the lower left for the red cluster and the lower right for the black cluster (compared to figure 4.2).

Nevertheless, there exist several numerical methods like stochastic gradient ascend [72] and other iterative techniques [73] to optimize respective functions. Instead of these numerical methods, here we focus on a different access to solve this problem, a framework which is called *Expectation Maximization*. This optimization scheme is probabilistically motivated and will be considered in more detail in the next subsection.

4.1.2. Maximizing Mixture Models

There are several views of the Expectation Maximization Algorithm (EM) realizing different strategies. The most common one makes use of the probabilistic nature of mixture models (see e.g. [38], [74]) introducing so called hidden variables. A second approach that we want to discuss takes a deterministic view, which gives the possibility to generalize the EM to non-probabilistic problems.

Probabilistic View on EM The starting point for the probabilistic formulation is the introduction of a matrix $\mathcal{Z}$ of hidden variables into the mixture model with the aim to simplify the optimization problem of the log-likelihood (4.10). Therefore, the new local latent variables $\mathcal{Z}(i, \cdot)$ are n_c -dimensional binary random vectors with components $\mathcal{Z}(i, k) \in \{0, 1\}$ for each data point x_i and the additional restriction $\sum_{k=1}^{n_c} \mathcal{Z}(i, k) = 1, \forall i$. Now, the goal is to reformulate the log-likelihood (4.10) in terms of marginal distribution $p(\mathcal{Z}(i, \cdot))$ with the reasonable constraint $p(\mathcal{Z}(i, k) = 1) = \pi_k$ and the conditional distribution $p(x_i | \mathcal{Z}(i, \cdot))$, such that $p(x_i | \mathcal{Z}(i, k) = 1) = \mathcal{N}(x_i | \mu_k, \Sigma_k)$ holds. Particularly we have

$$p(\mathcal{Z}(i, \cdot)) = \prod_{k=1}^{n_c} \pi_k^{\mathcal{Z}(i, k)} \quad (4.11)$$

$$p(x_i | \mathcal{Z}(i, \cdot)) = \prod_{k=1}^{n_c} \mathcal{N}(x_i | \mu_k, \Sigma_k)^{\mathcal{Z}(i, k)} \quad (4.12)$$

Hence, having the joint distribution $p(x_i, \mathcal{Z}(i, \cdot))$ we can describe the mixture model (4.5) as marginalization over all possible states $\tilde{\mathcal{Z}}_i$ of $\mathcal{Z}(i, \cdot)$

$$\begin{aligned} p_m(x_i) &= \sum_{\tilde{\mathcal{Z}}_i} p(x_i, \tilde{\mathcal{Z}}_i) \\ &= \sum_{\tilde{\mathcal{Z}}_i} p(\tilde{\mathcal{Z}}_i) p(x_i | \tilde{\mathcal{Z}}_i). \end{aligned}$$

The set $\{\mathcal{X}, \mathcal{Z}\}$ is denoted as the *complete* data set, where the data points $x \in \mathcal{X}$ are the observed data and the latent variables $\mathcal{Z}$ are the unobserved data. Consequently, $\mathcal{X}$ without $\mathcal{Z}$ is called the *incomplete* data. The likelihood of the complete data now becomes

$$\begin{aligned} p(\mathcal{X}, \mathcal{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= \prod_{i=1}^{n_d} \prod_{j=1}^{n_c} p(\mathcal{Z}(i, j)) p(x_i | \mathcal{Z}(i, j)) \\ &= \prod_{i=1}^{n_d} \prod_{j=1}^{n_c} \pi_j^{\mathcal{Z}(i, j)} \mathcal{N}(x_i | \mu_j, \Sigma_j)^{\mathcal{Z}(i, j)} \end{aligned}$$

and the log-likelihood, after simplification, results to

$$\ln(p(\mathcal{X}, \mathcal{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} \mathcal{Z}(i, j) [\ln(\pi_j) + \ln(\mathcal{N}(x_i | \mu_j, \Sigma_j))]. \quad (4.13)$$

We detect that the log-likelihood (4.13) for the complete data is simpler than the log-likelihood (4.10) for the incomplete data, because the summation takes place outside of the logarithm.

Starting from this complete likelihood the EM algorithm can be derived as an alternating optimization method. To do so, the first step is the determination of the expectation of the variables $\mathcal{Z}$ regarding to the posterior distribution $p(\mathcal{Z} | \mathcal{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$

$$\mathbb{E}[\mathcal{Z}(i, j)] = \frac{\pi_j \mathcal{N}(x_i | \mu_j, \Sigma_j)}{\sum_{k=1}^{n_c} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k)} \quad (4.14)$$

$$= \gamma_j(x_i) \quad (4.15)$$

with $\gamma_j(x_i)$ known from (4.9). For simplification purposes we will use the short hand notation γ_{ij} instead of $\gamma_j(x_i)$ if there is no risk of confusion. The expectation of the complete likelihood, regarding the hidden variables, is then given

by

$$\mathbb{E}_{\mathcal{Z}} [\ln (p(\mathcal{X}, \mathcal{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}))] = \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} \gamma_{ij} \{ \ln \pi_j + \ln [\mathcal{N}(x_i | \mu_j, \Sigma_j)] \}$$

and should be maximized in the second step. Therefore, we set the derivatives of $\mathbb{E}_{\mathcal{Z}} [\ln (p(\mathcal{X}, \mathcal{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}))]$, with respect to the parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, to zero and solve the resulting equations for the corresponding variables. For the mixture coefficients π_j we have additionally to ensure that they are sum up to one. Both, the expectation step (E-step) and the maximization step (M-step), are applied iteratively in a alternating manner. The detailed derivations can be found in [38]. The resulting *alternating* EM algorithm is summarized in Figure 4.4.

As mentioned at the beginning of this section, the illustrated way is only one possibility to derive the EM-algorithm for Gaussian mixtures. In the following, another idea will be outlined.

Deterministic View on EM The main difference to the previous approach is that the probabilistic behavior of the mixture model will set aside. For the derivation we use only analytical properties of the log-likelihood (4.10). The motivation behind is that this approach can be generalized to other objective functions, which do not have a probabilistic background. It turns out that this strategy will be the basic optimization method for median algorithms, which are one main focus of this thesis. This approach is denoted as generalized EM algorithm (gEM).

Of course, essential idea is similar as before to keep an alternating optimization scheme like in the EM but without use of the complete data $\{\mathcal{X}, \mathcal{Z}\}$.

For this purpose we introduce the formal probability scores $g_j(x_i) \geq 0$ with

$$\sum_{j=1}^{n_c} g_j(x_i) = 1 \tag{4.16}$$

EM algorithm for Gaussian Mixtures

1. **Initialization:** randomly initialize all free parameters (π_j, μ_j, Σ_j) of the model and set the stopping criteria $\varepsilon > 0$
2. **Expectation (E-Step):** evaluate the responsibilities

$$\gamma_{ij} = \frac{\pi_j \mathcal{N}(x_i | \mu_j, \Sigma_j)}{\sum_{k=1}^{n_c} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k)}$$

3. **Maximization (M-Step):** evaluate the new values for the parameters

$$\begin{aligned} \pi_j^{\text{new}} &= \frac{\sum_{k=1}^{n_d} \gamma_{kj}}{n_d} \\ \mu_j^{\text{new}} &= \frac{1}{\sum_{k=1}^{n_d} \gamma_{kj}} \sum_{i=1}^{n_d} \gamma_{ij} x_i \\ \Sigma_j^{\text{new}} &= \frac{1}{\sum_{k=1}^{n_d} \gamma_{kj}} \sum_{i=1}^{n_d} \gamma_{ij} (x_i - \mu_j^{\text{new}})(x_i - \mu_j^{\text{new}})^T \end{aligned}$$

4. **Convergence check:** evaluate the likelihood and check for convergence. If

$$p(\mathcal{X} | \pi^{\text{new}}, \mu^{\text{new}}, \Sigma^{\text{new}}) - p(\mathcal{X} | \pi^{\text{old}}, \mu^{\text{old}}, \Sigma^{\text{old}}) \leq \varepsilon$$

then stop. Otherwise, proceed with step 2.

Figure 4.4.: EM algorithm for Gaussian Mixtures. Note that always $p(\mathcal{X} | \pi^{\text{new}}, \mu^{\text{new}}, \Sigma^{\text{new}}) \geq p(\mathcal{X} | \pi^{\text{old}}, \mu^{\text{old}}, \Sigma^{\text{old}})$ is valid because of the algorithmic restrictions (see text)

and with the formal Shannon Entropys

$$H_g^i = - \sum_{j=1}^{n_c} g_j(x_i) \ln(g_j(x_i)). \quad (4.17)$$

We can decompose the log-likelihood $\ell_{log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X})$ (4.10) into

$$\ell_{log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X}) = \sum_{i=1}^{n_d} [\mathcal{L}_i(g) + \mathcal{K}_i(g||p)]. \quad (4.18)$$

Here $\mathcal{L}(g)$ is

$$\begin{aligned} \mathcal{L}(g) &= \sum_{i=1}^{n_d} \mathcal{L}_i(g) \\ &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j)p(x_i|j)}{g_j(x_i)} \right] \end{aligned} \quad (4.19)$$

and $\mathcal{K}(g||p)$ is

$$\begin{aligned} \mathcal{K}(g||p) &= \sum_{i=1}^{n_d} \mathcal{K}_i(g||p) \\ &= - \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j|x_i)}{g_j(x_i)} \right] \end{aligned} \quad (4.20)$$

and $\mathcal{K}_i$ is the Kullback-Leibler-divergence (KLD) between $g = (g_1(x_i), \dots, g_{n_c}(x_i))$ and $p = (p(j=1|x_i), \dots, p(j=n_c|x_i))$. The proof for the decomposition (4.18) can be found in Appendix A.1 Here, the prior probability $p(j)$, the conditional probability $p(x_i|j)$ and the posterior probability $p(j|x_i)$ are given as before (Equations (4.6),(4.7),(4.9)). Further, we explicitly remark that the decomposition is independent from the concrete realization of $g_j(x_i)$.

Because the $\mathcal{K}_i(g||p)$ are divergences we have

$$\begin{aligned}\mathcal{K}(g||p) &\geq 0 \\ \mathcal{K}(g||p) &= 0 \Leftrightarrow g_j(x_i) = p(j|x_i), \forall j, i\end{aligned}$$

for every valid choice of the formal probability scores $g_j(x_i)$. Hence, $\mathcal{L}(g)$ is a lower bound of the log-likelihood and, especially, $\mathcal{L}(g)$ is equal to the log-likelihood for the choice $g_j(x) = p(j|x_i)$. Using these properties of $\mathcal{K}$ and $\mathcal{L}$, we

Figure 4.5.: Illustration of the basic steps in the gEM Algorithm. The blue bars represent the values of the log-likelihood $\ell_{\log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X})$, the green bars represent the values of the lower bound $\mathcal{L}(g)$ and the red bars represent the values of the Kullback-Leibler-Divergence $\mathcal{K}(g||p)$

can derive an EM like algorithm in the following way: After the initialization, the lower bound $\mathcal{L}$ and the Kullback-Leibler-Divergence $\mathcal{K}$ sum up to the value

of the log-likelihood. In the E-step, we set the variables as $g_j(x_i) \leftarrow p(j|x_i)$ such that $\mathcal{K}(g|p) = 0$ is valid and, therefore, $\ell_{log} = \mathcal{L}(g)$ holds afterwards. Thus, ℓ_{log} remains unchanged during the E-step according to the remark regarding the independence on g of the decomposition stated above. In the M-step, we maximize the lower bound $\mathcal{L}$ with respect to the free parameters $(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ while fixing the free variables $g_j(x_i)$. After application of the M-step the value of $\mathcal{L}$ is either increased or remains unchanged, if no better parameter configuration is found. Yet, a change of the parameters $(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ causes a recalculation of the posterior probabilities $p(k|x_i)$ and, hence, also the function $\mathcal{K}$ gets a new value. However, due to the non-negativity property of KLD, $\mathcal{K} > 0$ is valid such that the resulting new value ℓ_{log} is greater than before. Now, an E-step can be applied again to optimize the formal probability scores $g_j(x_i)$. In the following we show that the E-step is comparable to the E-step of the EM algorithm (see Figure 4.4) if we identify the probability scores $g_j(x_i)$ of gEM with the responsibilities γ_{ij} in the EM algorithm. We start rewriting the lower bound as

$$\begin{aligned}
 \mathcal{L}(g) &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j)p(x_i|j)}{g_j(x_i)} \right] \\
 &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [\pi_j p(x_i|j)] + \sum_{i=1}^{n_d} H_q^i \\
 &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [\pi_j] + \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [\mathcal{N}(x_i|\mu_j, \Sigma_j)] + \sum_{i=1}^{n_d} H_q^i.
 \end{aligned}$$

using $p(j) = \pi_j$, the Gaussian assumption for $p(x_i|j)$ and the formal Shannon entropy (4.17).

The entropies H_q^i are independent of π_j , μ_j , Σ_j and, therefore, can be ignored

for the maximization. Taking the necessary conditions

$$\begin{aligned}\frac{\partial}{\partial \pi_j} \mathcal{L}(g) &= 0, \forall j \\ \frac{\partial}{\partial \mu_j} \mathcal{L}(g) &= 0, \forall j \\ \frac{\partial}{\partial \Sigma_j} \mathcal{L}(g) &= 0, \forall j\end{aligned}$$

for an maximum of the lower bound into account, and paying attention to the restriction that $\sum_{j=1}^{n_c} \pi_j = 1$, the Lagrange method for constrained optimization can be applied and we obtain

$$\begin{aligned}0 &= \frac{\partial}{\partial \pi_k} \left[\sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [\pi_j] + \lambda \left(\sum_{j=1}^{n_c} \pi_j - 1 \right) \right] \\ &= \frac{1}{\pi_k} \sum_{i=1}^{n_d} g_k(x_i) + \lambda\end{aligned} \quad (4.21)$$

and

$$\begin{aligned}0 &= \frac{\partial}{\partial \lambda} \left[\sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [\pi_j] + \lambda \left(\sum_{j=1}^{n_c} \pi_j - 1 \right) \right] \\ &= \sum_{j=1}^{n_c} \pi_j - 1\end{aligned} \quad (4.22)$$

with the Lagrangian multiplier λ . For the first equation we get

$$\pi_k = -\frac{1}{\lambda} \sum_{i=1}^{n_d} g_k(x_i), \forall k.$$

Summing up all mixture components k we have

$$\underbrace{\sum_{k=1}^{n_c} \pi_k}_{=1} = -\frac{1}{\lambda} \sum_{i=1}^{n_d} \sum_{k=1}^{n_c} g_k(x_i)$$

and get for the Lagrangian multiplier the relation

$$\begin{aligned} \lambda &= - \sum_{i=1}^{n_d} \underbrace{\sum_{k=1}^{n_c} g_k(x_i)}_{=1} \\ &= -n_d. \end{aligned}$$

Plugging this result into (4.21), we obtain

$$\frac{1}{\pi_k} \sum_{i=1}^{n_c} g_k(x_i) - n_d = 0$$

and we end up with the result

$$\pi_k = \frac{\sum_{i=1}^{n_d} g_k(x_i)}{n_d}.$$

as updated values for π_k .

To derive the update for the mean values μ_k we consider

$$\begin{aligned} 0 &= \frac{\partial}{\partial \mu_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln(\mathcal{N}(x_i | \mu_j, \Sigma_j)) \\ &= \frac{\partial}{\partial \mu_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma^{-1} (x_i - \mu_j) \right) \end{aligned}$$

whereas for the covariance matrices Σ_k we have to analyze

$$\begin{aligned}
 0 &= \frac{\partial}{\partial \Sigma_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln (\mathcal{N}(x_i | \mu_j, \Sigma_j)) \\
 &= \frac{\partial}{\partial \Sigma_k} \left[\sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma^{-1} (x_i - \mu_j) \right) \right. \\
 &\quad \left. - \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left(\sqrt{(2\pi)^D |\Sigma_k|} \right) \right]
 \end{aligned}$$

accordingly. The solution for both updates is obtained as

$$\begin{aligned}
 \mu_k^{\text{new}} &= \frac{1}{\sum_{i=1}^{n_d} g_k(x_i)} \sum_{i=1}^{n_d} g_k(x_i) x_i \\
 \Sigma_k^{\text{new}} &= \frac{1}{\sum_{i=1}^{n_d} g_k(x_i)} \sum_{i=1}^{n_d} g_k(x_i) (x_i - \mu_k^{\text{new}})(x_i - \mu_k^{\text{new}})^T,
 \end{aligned}$$

respectively. Obviously, these update rules for π_k , μ_k and Σ_k are equivalent to those in the EM scheme (Figure 4.4) considering that the EM Parameter $g_j(x_i)$ and the responsibilities $\gamma_j(x_i)$ are in correspondence.

To summarize this subsection, we have shown how it is possible to use the EM Algorithm to optimize log-likelihood functions. Moreover, we demonstrated for the Gaussian mixture model that several approaches exist to tackle the optimization problem and lead to the same solution.

4.2. Expectation Maximization for vector quantization

4.2.1. Jensen Inequality

One further possibility to motivate the generalized EM Algorithm is the well known Jensen Inequality [75]. Therefore, we give a reminder regarding the inequality and a particular consequence, which is important for this thesis. The Jensen inequality for concave functions is formulated as the following theorem:

Theorem 1 *For any concave real valued function f , arbitrary numbers x_k ($k = 1, \dots, K$) and positive λ_k with $\sum_{k=1}^K \lambda_k = 1$ the following inequality holds*

$$f\left(\sum_{k=1}^K \lambda_k x_k\right) \geq \sum_{k=1}^K \lambda_k f(x_k).$$

Thus, basically the theorem gives an upper bound for an concave function applied to a convex linear combination as argument. This theorem implies the following corollary:

Corollary 1 *Suppose $c_k > 0$ ($k = 1, \dots, K$) are arbitrary numbers and λ_k are defined as above. Then it follows immediately that*

$$f\left(\sum_{k=1}^K c_k\right) = f\left(\sum_{k=1}^K \lambda_k \frac{c_k}{\lambda_k}\right) \geq \sum_{k=1}^K \lambda_k f\left(\frac{c_k}{\lambda_k}\right).$$

As an example of this corollary, we can see the derivation of the lower bound in the last section. Starting from the log-likelihood (4.10) we get

$$\begin{aligned}
 \ell_{log} &= \sum_{i=1}^{n_d} \ln \left[\sum_{j=1}^{n_c} p(j) p(x_i|j) \right] \\
 &= \sum_{i=1}^{n_d} \ln \left[\sum_{j=1}^{n_c} g_j(x_i) \frac{p(j) p(x_i|j)}{g_j(x_i)} \right] \\
 &\geq \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j) p(x_i|j)}{g_j(x_i)} \right] \\
 &= \mathcal{L}(g).
 \end{aligned}$$

Therefore, it is clear from the corollary that the derived formula (4.19) is a lower bound of the log-likelihood. This idea will be the main point for the derivation of the EM-algorithm in a more general version, which is explained in the next subsection.

4.2.2. Generalization of Expectation Maximization

The goal of this subsection is to derive an EM-algorithm for more general versions of objective functions than probabilistic like in the Gaussian mixture model. For this purpose, we start with an abstract view and give afterwards a more detailed example how to use this for vector quantization (VQ). As mentioned in section 2.1 the goal of VQ models is to distribute prototypes θ_j in the input space such that a regarding cost function will be optimized. Generally this cost function depends on prototype set $\Theta = \{\theta_1, \dots, \theta_{n_p}\}$ and the given input data $\mathcal{X} = \{x_1, \dots, x_{n_d}\}$. In a formal way, we can describe the structure of those cost functions as

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \mathcal{G}(x_i, \theta_j). \quad (4.23)$$

Without concentrating on a particular example, we now want to discuss the general maximization problem

$$\sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \mathcal{G}(x_i, \theta_j) \xrightarrow{\Theta} \max$$

under the assumption that $\mathcal{G}(x_i, \theta_j)$ is a positive and bounded function for all possible inputs x_i and θ_j . The reason for having this abstract view is that we can take all the derivations in the following sections as special cases of this general discussion.

Now, let f be a monotonically increasing function and $\hat{\Theta}$ a particular set of prototypes, which maximizes the cost function $\mathcal{C}(\mathcal{X}, \Theta)$. Then this parameter set $\hat{\Theta}$ also maximizes the function $f(\mathcal{C}(\mathcal{X}, \Theta))$. This means in particular that it is irrelevant for the maximization problem, whether we use the cost function $\mathcal{C}(\mathcal{X}, \Theta)$ or the transformed cost function

$$\mathcal{C}_f(\mathcal{X}, \Theta) = f(\mathcal{C}(\mathcal{X}, \Theta)) \quad (4.24)$$

if this helps to solve the problem easier. Let f be monotonically increasing, arbitrarily but fixed, e.g. the natural logarithm $f(x) = \ln(x)$ or the square root $f(x) = \sqrt{x}$. Further, we introduce the formal probability score $g(x_i, \theta_j) = g_{ij}$ such that

$$\begin{aligned} g_{ij} &\geq 0 \quad \forall i, j \\ \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} &= 1 \end{aligned} \quad (4.25)$$

is valid. Then it follows directly from the Corollary 1 that

$$\mathcal{C}_f(\mathcal{X}, \Theta) \geq \sum_{i,j} g_{ij} f\left(\frac{\mathcal{G}(x_i, \theta_j)}{g_{ij}}\right)$$

holds. Moreover, we have with

$$\mathcal{L}(\mathcal{X}, \Theta) = \sum_{i,j} g_{ij} f\left(\frac{\mathcal{G}(x_i, \theta_j)}{g_{ij}}\right) \quad (4.26)$$

a lower bound for the cost function $\mathcal{C}_f(\mathcal{X}, \Theta)$ for all valid choices of the formal probability scores g_{ij} . Doing so, the cost function $\mathcal{C}_f$ can be decomposed into

$$\mathcal{C}_f(\mathcal{X}, \Theta) = \mathcal{L}(\mathcal{X}, \Theta) + \mathcal{R}(\mathcal{X}, \Theta) \quad (4.27)$$

which allows to apply an EM-like algorithm for the optimization.

Remark 5 *It can be shown that $\mathcal{R}(\mathcal{X}, \Theta)$ is always non-negative, because $\mathcal{L}(\mathcal{X}, \Theta)$ is a lower bound of $\mathcal{C}_f(\mathcal{X}, \Theta)$ according the Jensen inequality (Corollary 1). Further, as shown in the Appendix A.2, $\mathcal{R}(\mathcal{X}, \Theta)$ becomes zero in the expectation step, if the optimum choice*

$$\hat{g}_{ij} = \frac{\mathcal{G}(x_i, \theta_j)}{\sum_{(x_k, \theta_l)} \mathcal{G}(x_k, \theta_l)}. \quad (4.28)$$

is taken. Otherwise, during the maximization step $\mathcal{R}(\mathcal{X}, \Theta)$ could increase to a certain positive value but never becomes negative according to the firstly mentioned property.

At this point, we explicitly remark that this kind of decomposition is the basis for all later considered decompositions of cost functions for the several algorithms. Therefore, we will always refer to this general scheme if necessary.

Taking the previous settings, we can formally describe the EM-algorithm as depicted in Figure 4.6.

Additionally we use a result from [74], which states that it is also sufficient to make only an improvement of the lower bound in the maximization step instead of searching for the overall maximum. This type of EM is known as generalized Expectation Maximization Algorithm. The benefit of this generalization is that, if a closed solution for the maximum in the maximization step does

not exist, we can also use some approximation which ensures a non-decreasing behavior of the lower bound. Especially when the EM-algorithm is applied for median methods, we will make use of this generalization.

The Algorithm is described in a general manner. Yet, the expectation step is obvious for every particular choice of cost functions, whereas the maximization step has to be specified precisely. In the next section, we will consider an example from unsupervised learning to underpin the possible usage of the generalized EM.

4.2.3. Expectation Maximization for Neural Gas

The Neural Gas Algorithm is an unsupervised algorithm for vector quantization and was introduced by T. Martinez and K. Schulten [4], [17]. The task in unsupervised VQ is to distribute the prototypes $\Theta = \{\theta_1, \dots, \theta_{n_p}\}$ in the data space, where the data distribution P is given only implicitly by a training data set $\mathcal{X}$. The idea of NG is to consider the prototypes as particles of a gas with viscosity, realizing particle cooperativeness during learning in dependence of a formal temperature $T = 1/\lambda$. The cost function to be minimized is the energy of the gas

$$\mathcal{C}(\mathcal{X}, \Theta) = \frac{1}{C(\lambda)} \sum_{j=1}^{n_p} \int h_\lambda(k_j(x, \Theta)) d^2(x, \theta_j) P(x) dx$$

where $d^2(x, \theta_j)$ is again squared Euclidean distance. Further $k_j(x, \Theta)$ is the rank function

$$k_j(x, \Theta) = |\{\theta_i | d^2(x, \theta_i) < d^2(x, \theta_j)\}| \quad (4.29)$$

for the prototypes, with $k_j(x, \Theta) = 0$ for the best matching prototype. The neighborhood function $h_\lambda(t)$ is defined as exponential

$$h_\lambda(t) = \exp(-tT)$$

generalized EM algorithm for general cost functions

1. **Initialization:** randomly initialize the free parameters (the prototypes Θ) of the model and set the stopping criteria $\varepsilon > 0$

2. **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_{ij} = \frac{\mathcal{G}(x_i, \theta_j)}{\sum_{k,l} \mathcal{G}(x_k, \theta_l)}$$

3. **Maximization (M-Step):** evaluate the new values for the parameters Θ such that

$$\mathcal{L}(\mathcal{X}, \Theta)^{(\text{new})} \geq \mathcal{L}(\mathcal{X}, \Theta)^{(\text{old})}$$

holds for fixed probability scores g_{ij}

4. **Convergence check:** evaluate the cost function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) - \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})}) \leq \varepsilon$$

then stop. Else start again with step 2.

Figure 4.6.: generalized EM-algorithm for general cost functions. Note that always $\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) \geq \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$ is valid because of the algorithmic restrictions.

with the positive formal temperature T . The normalization constant $C(\lambda)$ is

$$C(\lambda) = \sum_{j=1}^{n_p} h_\lambda(k_j(x, \Theta)) = \sum_{k=0}^{n-1} h_\lambda(k).$$

In [4] it is shown how to optimize this cost function by stochastic gradient descent learning which yields an online algorithm.

Here we focus on the batch scenario for Neural Gas which was derived in [76]. Accordingly, the cost function is reformulated for finite data sets $\mathcal{X} = \{x_1, \dots, x_{n_d}\}$ as

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} h_\lambda(k_j(x_i, \Theta)) d^2(x_i, \theta_j). \quad (4.30)$$

to be minimized by alternating steps: First, the variables $k_j(x_i, \Theta)$ are determined according to Equation (4.29) while fixing the prototypes. Afterwards, the optimal prototypes are found according to

$$\begin{aligned} \hat{\theta}_j &= \sum_{i=1}^{n_d} \beta_i x_i \\ \text{with} \quad \hat{\beta}_i &= \frac{h_\lambda(k_j(x_i, \Theta))}{\sum_{l=1}^{n_d} h_\lambda(k_j(x_l, \Theta))} \\ &= \hat{\beta}(x_i) \end{aligned} \quad (4.31)$$

for fixed $k_j(x_i, \Theta)$.

In the following, we give an alternative formulation for batch optimization in terms of gEM. We will show later that the calculation of the probability score $g(x_i, \theta_j)$ is not necessary here. For this approach we keep in mind that the Gaussian kernel (3.27) is a rank-equivalent similarity to the squared Euclidean distance. Hence, minimizing (4.30) is topological equivalent¹ to maximizing

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} h_\lambda(k_j(x_i, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_i, \theta_j) \quad (4.32)$$

¹The costfunctions are not equivalent in a strict mathematical sense. Meaning the minimum/maximum does not appear necessarily for exact the same prototype setting.

whereby the rank function (4.29) formally becomes

$$k_j(x_i, \Theta) = |\{\theta_k | \mathfrak{R}_{\text{Gauss}}(x_i, \theta_k) > \mathfrak{R}_{\text{Gauss}}(x_i, \theta_j)\}|.$$

The change of the rank function is just a formalism because from the discussion in subsection 3.2 it follows directly that

$$|\{\theta_j | \mathfrak{R}_{\text{Gauss}}(x, \theta_j) > \mathfrak{R}_{\text{Gauss}}(x, \theta_i)\}| = |\{\theta_j | d^2(x, \theta_j) < d^2(x, \theta_i)\}|$$

holds.

Thus, the cost function (4.32) has the structure required in Section 4.2.2 for gEM and so we can apply the generalized Expectation Maximization scheme for the optimization. Taking the natural logarithm for the function f (Equation (4.24)) from subsection 4.2.2 we obtain

$$\mathcal{C}_f(\mathcal{X}, \Theta) = \ln \left(\sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \mathcal{G}(x_i, \theta_j, \Theta) \right)$$

with

$$\mathcal{G}(x_i, \theta_j, \Theta) = h_\lambda(k_j(x_i, \Theta)) \mathfrak{R}_{\text{Gauss}}(x_i, \theta_j).$$

Thus, the lower bound regarding equation (4.26) becomes

$$\begin{aligned} \mathcal{L}(\mathcal{X}, \Theta) &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \ln \left[\frac{h_\lambda(k_j(x_i, \Theta)) \mathfrak{R}_{\text{Gauss}}(x_i, \theta_j)}{g_{ij}} \right] \\ &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \left(-\frac{k_j(x_i, \Theta)}{\lambda} \right) + \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \left(-\frac{d^2(x_i, \theta_j)}{2\sigma^2} \right) \\ &\quad - \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \ln(g_{ij}) \end{aligned}$$

with the formal probability scores $g_{ij} = g(x_i, \theta_j)$ which can be chosen regarding equations (4.25). This lower bound should be increased in the maximization step. Moreover, for that case it is always possible to find a maximum regarding the prototype set Θ for the lower bound: The necessary condition for the maximum is

$$\frac{\partial}{\partial \theta_j} \mathcal{L}(\mathcal{X}, \Theta) = 0 \quad (4.33)$$

where the optimal but fixed probability scores $\hat{g}_{ij}$ are obtained according to (4.28) as

$$\begin{aligned} \hat{g}_{ij} &= \hat{g}(x_i, \theta_j) \\ &= \frac{\mathcal{G}(x_i, \theta_j, \Theta)}{\sum_{l=1}^{n_d} \sum_{m=1}^{n_p} \mathcal{G}(x_l, \theta_m, \Theta)} \\ &= \frac{h_\lambda(k_j(x_i, \Theta)) \mathfrak{R}_{\text{Gauss}}(x_i, \theta_j)}{\sum_{l=1}^{n_d} \sum_{m=1}^{n_p} k_m(x_l, \Theta) \mathfrak{R}_{\text{Gauss}}(x_l, \theta_m)}. \end{aligned} \quad (4.34)$$

Thus, the solution of (4.33) for optimal prototypes $\hat{\theta}$ with fixed ranks $k_j(x_i, \Theta)$ is

$$\hat{\theta}_j = \sum_{i=1}^{n_d} \hat{\beta}_i x_i \quad (4.35)$$

with

$$\begin{aligned} \hat{\beta}_i &= \frac{\hat{g}_{ij}}{\sum_{l=1}^{n_d} \hat{g}_{lj}} \\ &= \hat{\beta}(x_i) \end{aligned}$$

The detailed derivation for that result can be found in Appendix A.3.

For comparison of the resulting equation (4.35) with equation (4.31) we take a closer look to the updates for the prototypes $\hat{\theta}_j$. Substituting (4.34) into

equation (4.35) we get

$$\begin{aligned}
\theta_j &= \frac{\sum_{i=1}^{n_d} \hat{g}_{ij} x_i}{\sum_{r=1}^{n_d} \hat{g}_{rj}} \\
&= \frac{\sum_{i=1}^{n_d} \frac{h_\lambda(k_j(x_i, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_i, \theta_j)}{\sum_{l=1}^{n_d} \sum_{m=1}^{n_p} k_m(x_l, \Theta) \mathfrak{K}_{\text{Gauss}}(x_l, \theta_m)} x_i}{\sum_{r=1}^{n_d} \frac{h_\lambda(k_j(x_r, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_r, \theta_j)}{\sum_{l=1}^{n_d} \sum_{m=1}^{n_p} k_m(x_l, \Theta) \mathfrak{K}_{\text{Gauss}}(x_l, \theta_m)}} \\
&= \frac{\sum_{i=1}^{n_d} h_\lambda(k_j(x_i, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_i, \theta_j) x_i}{\sum_{r=1}^{n_d} h_\lambda(k_j(x_r, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_r, \theta_j)}.
\end{aligned}$$

From that it follows immediately that we do not need to calculate the formal probability scores $\hat{g}(x_i, \theta_j) = \hat{g}_{ij}$ explicitly in the expectation step. It is sufficient to evaluate and fix the neighborhood function $h_\lambda(k_j(x_i, \Theta))$ as well as the prototypes. This results in the algorithm, which is shown in Figure 4.10.

Further, we can directly compare the result with the alternating scheme derived in [76]. In the first step of both algorithms we evaluate the neighborhood function $h_\lambda(k_j(x_i, \Theta))$ which depends only on the ranks $k_j(x_i, \Theta)$. Although we use different similarity/dissimilarity measures in both cases (squared Euclidean distance for the classical Batch NG and Gaussian Kernel for the EM Variant) the ranks are equal because of the rank equivalent behavior of both proximity measures. Thus, the first step is equivalent for both variants. Looking closer on the second step Equation (4.31) vs. Equation (4.35)) we find that both update rules are structurally similar: The update step constitutes in each case a weighted average of the given input data. In the classical batch variant (Equation (4.31)) we weight this average only by the values of the neighborhood function. Especially for the case that the neighborhood range decreases ($\lambda \rightarrow 0$), this update converges to the simple mean of all those data points, for which the considered prototype is best fitting. In case of the EM update (Equation (4.35)) the weighting includes the neighborhood function as well as the Gaussian similarity $\mathfrak{K}_{\text{Gauss}}(x_i, \theta_j)$. Assuming again a decreasing neighborhood range ($\lambda \rightarrow 0$) we end up approximately with an update step which is a

similarity weighted mean of the data for the considered prototype. Following from this it turns out that the update of the EM variant strictly depends on the chosen range σ of the Gaussian kernel and can be significantly different to the classical one using the squared Euclidean Dissimilarity.

This behavior is illustrated in experiments visualized in the Figures 4.7, 4.8, 4.9 for an artificial toy data set. This data set consist of three different cluster. Each of them is Gaussian distributed. The smaller one consists of 20 data points and the two larger clusters consist of 100 data points each. Furthermore, we consider only two prototypes in this setting, which are initialized at the center of the large clusters. Afterwards, we evaluate the weights $\beta, \hat{\beta}$ regarding to this initialization for the classical batch variant as well as for the EM variant where we choose two different values for the with σ of the Gaussian Kernel. The last picture in the figures shows the prototype positions after the update regarding to the weights $\beta, \hat{\beta}$. The NG neighborhood range $\lambda = 0.01$ was equal for each experiment.

Looking at Figure 4.7 we see the result for the classical Neural Gas update step. The position of the red prototype is, as expected, according to the mean of the data points belonging to both, the small and the left great cluster. Opposite to that we see in Figure 4.9 that the red prototype approximately is positioned at the center of only the left larger cluster. The main reason for that is that we choose a small kernel parameter $\sigma = 0.1$. Hence, the similarity value between all points of the smaller cluster and the prototype are approximately zero and therefore these data points do not contribute to the update (equation (4.35)). For larger $\sigma = 1$ we get an mixture of the last two extremes. This can be seen at Figure (4.8).

To summarize this subsection, we notice that it is possible to derive a batch variant of the neural gas algorithm using the generalized EM-algorithm from Section 4.2.2. The resulting NG is derived from a modified cost function but comparable to the well known batch variant of the NG from [76]. It is shown for a demonstration example that the prototype behavior differs for both algo-

rithms as consequence of the used proximity measures.

Figure 4.7.: Update Steps Neural Gas from [76]. The upper left figure shows the initial situation and the upper right figure shows the prototypes after the update step. The lower figures visualize the weights $\beta(x_i)$ regarding to equation (4.31). The values of the weights are color coded. The left figure is for the weights regarding the green prototype and the right figure is for the red prototype. For detailed explanation, see text.

Figure 4.8.: Update Steps Neural Gas with Gaussian Similarity and $\sigma = 1$

The upper left figure shows the initial situation and the upper right figure shows the prototypes after the update step. The lower figures visualize the weights $\hat{\beta}(x_i)$ regarding to equation (4.35). The values of the weights are color coded. The left figure is for the weights regarding the green prototype and the right figure is for the red prototype.

Compared to Figure 4.7 the red prototype is placed slightly closer to the mean of the left larger cluster after update as consequence of the smaller weights for all data points from the small cluster.

Figure 4.9.: Update Steps Neural Gas with Gaussian Similarity and $\sigma = 0.1$ The upper left figure shows the initial situation and the upper right figure shows the prototypes after the update step. The lower figures visualize the weights $\hat{\beta}(x_i)$ regarding to equation (4.35). The values of the weights are color coded. The left figure is for the weights regarding the green prototype and the right figure is for the red prototype. Compared to Figure 4.7 and Figure 4.8 we have only a minor, and therefore not visible, change of the red prototype after the update step. This is caused by the weights of the small cluster, which are approximately zero and, therefore, they do not contribute to the update of the prototype.

generalized EM-algorithm for Neural Gas

1. **Initialization:** initialize the free parameters (the prototypes Θ) of the model, the neighborhood range λ , the kernel parameter σ and set the abort criteria $\varepsilon > 0$,

2. **Expectation (E-Step):** evaluate the ranks

$$k_j(x_i, \Theta) = |\{\theta_j | d^2(x, \theta_j) < d^2(x, \theta_i)\}|$$

and the neighborhood function

$$h_\lambda(k_j(x_i, \Theta)) = \exp\left(-\frac{k_j(x_i, \Theta)}{\lambda}\right)$$

3. **Maximization (M-Step):** for fixed $h_\lambda(k_j(x_i, \Theta))$ evaluate the new prototype positions

$$\theta_j^{\text{new}} = \sum_{i=1}^{n_d} \hat{\beta}_i x_i$$

$$\hat{\beta}_i = \frac{h_\lambda(k_j(x_i, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_i, \theta_j)}{\sum_{r=1}^{n_d} h_\lambda(k_j(x_r, \Theta)) \mathfrak{K}_{\text{Gauss}}(x_r, \theta_j)}$$

4. **Convergence check:** evaluate the cost function and check for convergence. If

$$\mathcal{C}_f(\mathcal{X}, \Theta^{(\text{new})}) - \mathcal{C}_f(\mathcal{X}, \Theta^{(\text{old})}) \leq \varepsilon$$

then stop. Else start again with step 2.

Figure 4.10.: generalized EM-algorithm for Neural Gas with Gaussian Kernel. Note that always $\mathcal{C}_f(\mathcal{X}, \Theta^{(\text{new})}) \geq \mathcal{C}_f(\mathcal{X}, \Theta^{(\text{old})})$ is valid because of the algorithmic restrictions.

5. Median Algorithms for Supervised Vector Quantization Using General Proximity Measures

The goal of this chapter is to introduce the generalized Expectation Maximization Algorithm for median models in *supervised* learning. The first subsection give a general overview to the idea. The second and third subsections show solutions for classifiers regarding to different objective functions.

The chapter basically relates to the following publications: [77], [78] and [79].

5.1. The General Scheme for Supervised Vector Quantization Optimizing by Means of Generalized EM

Coming back to the median idea in classification learning (earlier mentioned in subsection 3.4) our goal is to optimize a properly chosen objective function under the restriction that the prototypes have to be data objects, i.e. we assume that the prototypes θ_j should be assigned to a certain data objects x_i according

to $\theta_j = x_i$. This implies that

$$\mathfrak{d}(x_k, \theta_j) = \mathfrak{d}(x_k, x_i), \quad \forall k$$

where $\mathfrak{d}(\cdot, \cdot)$ is the given general dissimilarity measure for data. To emphasize the fact that we do not use the data objects itself we write shortly $\mathfrak{d}(k, j)$ instead of $\mathfrak{d}(x_k, \theta_j)$. Similar to this we rewrite the general objective function (equation (4.23)) from subsection 4.2.2 as

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \mathcal{G}(\mathfrak{d}(i, j))$$

where the local classifications function $\mathcal{G}(\mathfrak{d}(i, j))$ is a positive and bounded function, to point out that the cost depends only on the dissimilarity between data objects and not on the data objects and prototypes itself. Following the discussion from subsection 4.2.2 we start with the formulation of the optimization task as

$$\mathcal{C}_f(\mathcal{X}, \Theta) = f \left[\sum_{i,j} \mathcal{G}(\mathfrak{d}(i, j)) \right] \rightarrow \max \quad (5.1)$$

for the objective function where we assume that f is an monotonic increasing and concave function. For the cost function (5.1) we have the decomposition

$$\mathcal{C}_f(\mathcal{X}, \Theta) = \mathcal{L}(\mathcal{X}, \Theta) + \mathcal{R}(\mathcal{X}, \Theta) \quad (5.2)$$

according to (4.27) where

$$\mathcal{L}(\mathcal{X}, \Theta) = \sum_{i,j} g_{ij} f \left(\frac{\mathcal{G}(\mathfrak{d}(i, j))}{g_{ij}} \right) \quad (5.3)$$

is a lower bound of the cost function for arbitrary chosen $g_{ij} \geq 0$ with $\sum_{i,j} g_{ij} = 1$. The choice of the optimal probability scores $\hat{g}$ in the expectation step directly

follows from equation (4.28) from Section 4.2.2 as

$$\hat{g}_{i,j} = \frac{\mathcal{G}(\mathfrak{d}(i,j))}{\sum_{k,l} \mathcal{G}(\mathfrak{d}(k,l))} \quad (5.4)$$

whereby it is easy to see that we can calculate the formal probability scores using only the pairwise dissimilarities between all data objects. Coming up to the maximization step we need to optimize lower bound $\mathcal{L}(\mathcal{X}, \Theta)$ (5.3) of the objective function (5.1) which depends only on the pairwise dissimilarities of the given data objects. Furthermore, for a given, finite set of data objects and one selected θ_m we can derive¹

$$x_w = \arg \max_{x_l \in \hat{\mathcal{X}}} \left[\sum_{i,j} \hat{g}_{i,j} f \left(\frac{\mathcal{G}(\mathfrak{d}(i,j))}{\hat{g}_{i,j}} \right) \right]_{\theta_m = x_l}$$

with $\hat{\mathcal{X}} = \{x_i \in \mathcal{X} | \nexists l : \theta_l = x_i\}$. This means, we can find that assignment $\theta_m = x_w$ for an fixed prototype θ_m to a data object x_w , which gives a maximum for the lower bound regarding to this single prototype.

Following from the above discussion we can take one prototype θ_m in the maximization step and set the assignment of this prototype to the data object x_w , i.e.

$$\theta_m = x_w$$

where x_w is calculated as

$$x_w = \arg \max_{x_l \in \mathcal{X} \cup \{x_k\}} \left[\sum_{i,j} \hat{g}_{i,j} f \left(\frac{\mathcal{G}(\mathfrak{d}(i,j))}{\hat{g}_{i,j}} \right) \right]_{\theta_m = x_l}$$

and x_k is the data object to which prototype θ_m is actually assigned. Doing so we have two possible outcomes: It holds that $x_w \neq x_k$ after the maximization

¹The notation $[F]_{\theta_m = x_l}$ means that F is evaluated taking θ_m equal to the data point x_l .

step, which means that the prototype is assigned to a new data object and, hence, we have an improvement of the lower bound. Otherwise, if $x_w = x_k$ holds, the lower bound stays obviously the same. In both cases, we have a non-decreasing behavior in the maximization step. Thus we can formulate the respective algorithm as given in Figure 5.1.

Yet, for large data sets a complete enumeration of all possible assignments is time consuming. Therefore, one can use heuristic strategies with lower computational costs. One obvious strategy would be just to take the first assignments, which gives a greater value for the upper bound instead of maximizing. A more clever strategy is to use an optimal stopping criteria related to the secretary problem [80], which gives the optimal choice with probability $1/e$ without enumerating all possible assignments. Due to the discussion in Section 4.2.2 the algorithm (Figure 5.1) converges to an optimum of the objective function. However, this optimum is for general cases only a local one. This fact follows directly from the definition of the maximization step: In the algorithm, only one prototype assignment is changed in each iteration. After the system is converged, the change of one prototype assignment does not deliver any further improvement of the objective function. Nevertheless, it is not guaranteed that the change of more than one prototype assignment at the same time does not yield any benefit regarding to the objective function value. Therefore, the maximization step can be seen as a greedy strategy to avoid a fully enumeration of all possible assignments.

5.2. Median Variants for cost function based LVQ Classifiers

5.2.1. Median Robust Soft LVQ (mRSLVQ)

The (continuous) Robust Soft LVQ [29] was already introduced and discussed in Section 2.2.2. Now we want to discuss, how we can optimize the objective

generalized EM algorithm for Median Algorithms

1. **Initialization:** initialize the assignments $\theta_j = x_k$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$

2. **for** $m = 1 : n_p$

a) **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_{i,j} = \frac{\mathcal{G}(\mathfrak{d}(i,j))}{\sum_{k,l} \mathcal{G}(\mathfrak{d}(k,l))}$$

b) **Maximization (M-Step):** evaluate the new assignment $\theta_m = x_w$ with

$$x_w = \arg \max_{x_l \in \hat{\mathcal{X}} \cup \{x_k\}} \left[\sum_{i,j} \hat{g}_{i,j} f \left(\frac{\mathcal{G}(\mathfrak{d}(i,j))}{\hat{g}_{i,j}} \right) \right]_{\theta_m = x_l}$$

and fixed probability scores $\hat{g}_{i,j}$

endfor

3. **Convergence check:** evaluate the cost function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else start again with step 2.

Figure 5.1.: generalized EM algorithm for Median Algorithms

function under the median restrictions. The respective objective of RSLVQ

(2.9) in subsection 2.2.2 can be written as

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \ln \left[\sum_{j=1}^{n_p} \delta_{c(x_i)}^{c(\theta_j)} p(\theta_j | x_i) \right]$$

with $p(\theta_j | x_i)$ according to (2.12). Thus, the class probability for class k becomes

$$p(k | x_i) = \sum_{j=1}^{n_p} \delta_{c(x_i)}^k \frac{\exp \left(-\frac{\mathfrak{d}(i, j)}{2\sigma^2} \right)}{\sum_{l=1}^{n_p} \exp \left(-\frac{\mathfrak{d}(i, l)}{2\sigma^2} \right)}$$

using the formal notation $\mathfrak{d}(i, j) = \mathfrak{d}(x_i, \theta_j)$ from the last subsection to point out that the likelihood depends only on the pairwise dissimilarities between data points. Taking into account the discussion from subsection 5.1 we get for the decomposition regarding (4.27)

$$\mathcal{C}(\mathcal{X}, \Theta) = \mathcal{L}_{RSLVQ}(\mathcal{X}, \Theta) + \mathcal{R}_{RSLVQ}(\mathcal{X}, \Theta)$$

and for the lower bound regarding (5.3)

$$\mathcal{L}_{RSLVQ}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \ln \left[\frac{\delta_{c(x_i)}^{c(\theta_j)} p(\theta_j | x_i)}{g_{ij}} \right]$$

with the formal probability scores g_{ij} (regarding Equation (4.25)) and the agreement that $0 \cdot \ln(0/0) = 0$ holds. The optimal probability scores $\hat{g}_{i,j}$ (5.4) for the expectation step becoming after simplifications

$$\hat{g}_{i,j} = \frac{\delta_{c(x_i)}^{c(\theta_j)} \exp \left(-\frac{\mathfrak{d}(i, j)}{2\sigma^2} \right)}{\sum_{k=1}^{n_p} \delta_{c(x_i)}^{c(\theta_k)} \exp \left(-\frac{\mathfrak{d}(i, k)}{2\sigma^2} \right)}.$$

Thus, the lower bound as well as the probability scores depend only on the

generalized EM algorithm for RSLVQ

1. **Initialization:** initialize the assignments $\theta_j = x_k$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$ and the parameter σ for the Gaussian

2. **for** $m = 1 : n_p$

- a) **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_{i,j} = \frac{\delta_{c(x_i)}^{c(\theta_j)} \exp\left(-\frac{\mathfrak{d}(i,j)}{2\sigma^2}\right)}{\sum_{k=1}^{n_p} \delta_{c(x_i)}^{c(\theta_k)} \exp\left(-\frac{\mathfrak{d}(i,k)}{2\sigma^2}\right)}$$

- b) **Maximization (M-Step):** evaluate the new assignment $\theta_m = x_w$ with

$$x_w = \arg \max_{x_l \in \mathcal{X} \cup \{x_k\}} \left[\sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \hat{g}_{i,j} \ln \left(\frac{\delta_{c(x_i)}^{c(\theta_j)} p(\theta_j | x_i)}{\hat{g}_{i,j}} \right) \right]_{\theta_m = x_l}$$

and fixed probability scores $\hat{g}_{i,j}$

endfor

3. **Convergence check:** evaluate the likelihood and check for convergence. If

$$\mathcal{C}_{RSLVQ}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}_{RSLVQ}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else, start again with step 2.

Figure 5.2.: generalized EM algorithm for RSLVQ

pairwise dissimilarities between prototypes and data points.

In consequence, the model fits very well the formalism from subsection (5.1) and, therefore, the algorithm shown in Figure (5.2) constitutes a Median variant of the RSLVQ.

5.2.2. Median Generalized LVQ (mGLVQ)

The generalized Learning Vector Quantization [27] is already introduced and discussed in Section 2.2.1. We will now discuss two possibilities to optimize the objective function of GLVQ under median restrictions. The first way follows mainly the ideas in [77] whereby the second approach follows the ideas from Section 5.1.

The objective function to be minimized in GLVQ (see Equation (2.4)) is

$$\mathcal{C}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} f_a(\kappa(x_i)) \quad (5.5)$$

with

$$\kappa(x_i) = \frac{\mathfrak{d}^+(i) - \mathfrak{d}^-(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)}.$$

For the gEM algorithm, we need an objective which has to be maximized, for that we have a closer look on

$$\tilde{\mathcal{C}}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} f_a(-\kappa(x_i)). \quad (5.6)$$

The optimal solutions for Equation (5.5) and Equation (5.6) are equal iff the activation function f_a is radial symmetric, meaning $f_a(-x) = -f_a(x)$. As example, we could use the identity as activation function.

Further, for realizing a gEM we have to choose f_a as concave function. Thus, we want to approximate the identity by a suitable chosen variant of a parametrized logarithmic function which is concave. For the approximation we start from a

parameterized version of the logarithm in the form

$$\ln_{a,b,\alpha}(x) = a \cdot \ln(x + \alpha) + b \quad (5.7)$$

and try to determine the values for a, b, α such that $\ln_{a,b,\alpha}(x) \approx x$, i.e. we want to approximate the identity function. First we use the property that the classifier function $\kappa(x_i)$ is bounded by $\kappa(x) \in [-1, 1]$ such that we have to approximate the identity in this interval. Therefore, we require

$$\begin{aligned} \ln_{a,b,\alpha}(-1) &= -1 \\ \ln_{a,b,\alpha}(1) &= 1 \end{aligned}$$

for the bounds. Assuming α as a free parameter in (5.7) we find

$$\begin{aligned} a^* &= -\frac{2}{\ln \left[\frac{\alpha - 1}{\alpha + 1} \right]} \\ b^* &= \frac{2}{\ln \left[\frac{\alpha - 1}{\alpha + 1} \right]} \ln[\alpha - 1] - 1 \end{aligned} \quad (5.8)$$

as respective values. Thus, the parameter α controls the gradient of the logarithm in the given interval. This behavior is illustrated in Figure 5.3. The larger the parameter α is chosen the better we approximate the identity function. In general the relation

$$\ln_{a^*,b^*,\alpha}(x) \xrightarrow{\alpha \rightarrow \infty} x, \quad \forall x \in [-1, 1]$$

holds. Therefore, it follows directly that the approximation

$$\mathcal{C}_{\ln}(\mathcal{X}, \Theta, a^*, b^*, \alpha) = \sum_{i=1}^{n_d} \left(\frac{1}{a^*} \ln[-\kappa(x_i) + \alpha] + b^* \right) \xrightarrow{\alpha \rightarrow \infty} \sum_{i=1}^{n_d} [-\kappa(x_i)]$$

Figure 5.3.: Properties of the function $\ln_{a,b,\alpha}(x)$ depending on the parameter α . The green curve is the identity function $f(x) = x$. The black one is the logarithm for $\alpha = 2$, the blue one for $\alpha = 5$ and the red one for $\alpha = 10$, with a and b chosen according to (5.8)

is valid. Hence $\mathcal{C}_{\ln}(\mathcal{X}, \Theta, a^*, b^*, \alpha)$ approximates the GLVQ objective function (5.6) with the identity as activation function f_a . Canceling out all constant parts in $\mathcal{C}_{\ln}(\mathcal{X}, \Theta, a, b, \alpha)$ we obtain

$$\mathcal{C}_{\ln}(\mathcal{X}, \Theta, a, b, \alpha) \xrightarrow{\Theta} \max \Leftrightarrow \sum_{i=1}^{n_d} \ln [-\kappa(x_i) + \alpha] \xrightarrow{\Theta} \max.$$

From this discussion, we can write the objective for the first median version of GLVQ as

$$\begin{aligned}
 \mathcal{C}_{GLVQ1}(\mathcal{X}, \Theta) &= \sum_{i=1}^{n_d} \ln [-\kappa(x_i) + \alpha] \\
 &= \sum_{i=1}^{n_d} \ln \left[\frac{\mathfrak{d}^-(i) - \mathfrak{d}^+(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \alpha \right] \\
 &= \sum_{i=1}^{n_d} \ln \left[\frac{\mathfrak{d}^-(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \frac{-\mathfrak{d}^+(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \alpha \right] \\
 &= \sum_{i=1}^{n_d} \ln \left[\left(\frac{\mathfrak{d}^-(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \frac{\alpha}{2} \right) + \left(\frac{-\mathfrak{d}^+(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \frac{\alpha}{2} \right) \right]
 \end{aligned} \tag{5.9}$$

where the parameter $\alpha > 1$ ensures positivity of the argument of the logarithm. Technically, this trick helps us because we now have a formal sum in the logarithm and we can apply the theories shown in Section 5.1. With these considerations we are ready to apply the theoretical results from Section 5.1 to obtain a first median variant of GLVQ denoted as mGLVQ1.

Now (5.9) reads as

$$\sum_{i=1}^{n_d} \ln [-\kappa(x_i) + \alpha] = \sum_{i=1}^{n_d} \ln [p^-(i) + p^+(i)]$$

with

$$p^-(i) = \frac{\mathfrak{d}^-(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \frac{\alpha}{2}$$

and

$$p^+(i) = \frac{-\mathfrak{d}^+(i)}{\mathfrak{d}^+(i) + \mathfrak{d}^-(i)} + \frac{\alpha}{2}.$$

The formal probability scores² are g_i^+ and g_i^- fulfilling $g_i^\pm \geq 0$ and $g_i^- + g_i^+ = 1$. The respective decomposition of (5.6) according to (4.27) becomes

$$\mathcal{C}_{GLVQ1}(\mathcal{X}, \Theta) = \mathcal{L}_{GLVQ1}(\mathcal{X}, \Theta) + \mathcal{R}_{GLVQ1}(\mathcal{X}, \Theta)$$

where

$$\mathcal{L}_{GLVQ1}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \left(g_i^- \ln \left[\frac{p^-(i)}{g_i^-} \right] + g_i^+ \ln \left[\frac{p^+(i)}{g_i^+} \right] \right)$$

is the respective lower bound. In the expectation step we set the optimal formal probability scores $\hat{g}_i$ as

$$\hat{g}_i^\pm = \frac{p^\pm(i)}{p^-(i) + p^+(i)},$$

such that $\mathcal{L}_{GLVQ1}(\mathcal{X}, \Theta) = \mathcal{C}_{GLVQ1}(\mathcal{X}, \Theta)$ holds. In the maximization step we optimize the lower bound for fixed $\hat{g}^\pm$. The involved parameters $p^\pm$ depend only on the pairwise dissimilarities between data objects and formal prototypes. Thus, the resulting EM-algorithm realizes a median variant of GLVQ (mGLVQ1) (see Figure 5.4).

Remembering the discussion about the interpretation of the objective function of GLVQ we emphasize here that this median version mGLVQ1 works on an approximation of the identity as the activation function. However, in Section 2.2.1 we argued that also other functions can constitute as a useful activation function for GLVQ. Thus, in the next step, we derive a second version of a median algorithm for the GLVQ, which allows taking more general activation functions than the identity into account. Our starting point for that is the formulation of the objective function of GLVQ from equation (5.6). Following

²Note that for the median versions of GLVQ the probability scores g_i do not depend explicitly on a prototype θ_j and, therefore, are indexed only by the explicit data point x_i .

generalized EM algorithm for mGLVQ1

1. **Initialization:** initialize the assignments $\theta_j = x_k$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$ and the approximation parameter $\alpha > 0$

2. **for** $m = 1 : n_p$

- a) **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_i^\pm = \frac{p^\pm(i)}{p^-(i) + p^+(i)}$$

- b) **Maximization (M-Step):** evaluate the new assignment $\theta_m = x_w$ with

$$x_w = \arg \max_{x_l \in \mathcal{X} \cup \{x_k\}} \left[\sum_{i=1}^{n_d} \left(\hat{g}_i^- \ln \left[\frac{p^-(i)}{\hat{g}_i^-} \right] + \hat{g}_i^+ \ln \left[\frac{p^+(i)}{\hat{g}_i^+} \right] \right) \right]_{\theta_m = x_l}$$

and fixed probability scores $\hat{g}_i^\pm$

endfor

3. **Convergence check:** evaluate the objective function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else, start again with step 2.

Figure 5.4.: generalized EM algorithm for mGLVQ1

the ansatz (Equation (5.1)) from Section 5.1 we want to maximize

$$\mathcal{C}_f(\mathcal{X}, \Theta) = \ln \left[\sum_{i=1}^{n_d} f_a(-\kappa(x_i)) \right] \quad (5.10)$$

for a given parametrized and radial symmetric activation function f_a . Note, that the logarithm is outside the sum compared to Equation (5.9), where it is inside. Using again formal probabilities

$$g_i > 0$$

with

$$\sum_{i=1}^{n_d} g_i = 1$$

we find for the objective the decomposition according to (4.27)

$$\mathcal{C}_{GLVQ2}(\mathcal{X}, \Theta) = \mathcal{L}_{GLVQ2}(\mathcal{X}, \Theta) + \mathcal{R}_{GLVQ2}(\mathcal{X}, \Theta)$$

where

$$\mathcal{L}_{GLVQ2}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} g_i \ln \left[\frac{f_a(\kappa(x_i))}{g_i} \right]$$

is the respective lower bound. In the expectation step we have the optimal probability scores $\hat{g}_i$ as

$$\hat{g}_i = \frac{f_a(-\kappa(x_i))}{\sum_{k=1}^{n_d} f_a(-\kappa(x_k))}.$$

such that $\mathcal{L}_{GLVQ2}(\mathcal{X}, \Theta) = \mathcal{C}_{GLVQ2}(\mathcal{X}, \Theta)$ holds. The resulting algorithm is summarized in Figure 5.5.

In the following, we discuss both variants of median GLVQ: First mGLVQ2 can be seen as a generalization of mGLVQ1. This follows directly from the fact that mGLVQ1 optimizes the GLVQ objective function with the identity as activation function whereas the second algorithm gives the opportunity to use more general activation functions. Further, the mGLVQ1 algorithm has better

generalized EM algorithm for mGLVQ2

1. **Initialization:** initialize the assignments $\theta_j = x_k$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$

2. **for** $m = 1 : n_p$

a) **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_i = \frac{f_a(\kappa(x_i))}{\sum_{k=1}^{n_d} f_a(\kappa(x_k))}$$

b) **Maximization (M-Step):** evaluate the new assignment $\theta_m = x_w$ with

$$x_w = \arg \max_{x_l \in \hat{\mathcal{X}} \cup \{x_k\}} \left[\sum_{i=1}^{n_d} \left(\hat{g}_i \ln \left[\frac{f_a(\kappa(x_i))}{\hat{g}_i} \right] \right) \right]_{\theta_j = x_l}$$

and fixed probability scores $\hat{g}_i$

endfor

3. **Convergence check:** evaluate the objective function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else, start again with step 2.

Figure 5.5.: generalized EM algorithm for mGLVQ2

numerical properties: This result from the different definitions of the formal probability scores g . In mGLVQ1 the formal probability scores are defined via

the winning prototypes such that $g_i^+ + g_i^- = 1$, $\forall i$ holds. This means especially that the magnitude of the scores is independent of the number of data points. In the contrary, in mGLVQ2 the scores are defined such that $\sum_{i=1}^{n_d} g_i = 1$ holds, i.e. the formal probability scores scale with the number of data points. This means, if the amount of data is increasing, the formal probability scores tend to get very small and equal due to the arithmetic precision. Yet, equal values g_i have to be interpreted as some weighting of prototype importance.

5.2.3. Median Nearest Prototype Classification (mNPC)

Next we consider a median variant of NPC. The objective is

$$\mathcal{C}_{NPC}(\mathcal{X}, \Theta) = \frac{1}{n_d} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \left[1 - \delta_c^{c(x_i)} \right] \delta_{\theta_{s(i)}}^{\theta_j}$$

according to the cost function of sNPC (2.10) with crisp assignments (2.11). Instead of minimizing of $\mathcal{C}_{NPC}$ we can maximize

$$\mathcal{C}_{mNPC}(\mathcal{X}, \Theta) = \frac{1}{n_d} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} \delta_c^{c(x_i)} \delta_{\theta_{s(i)}}^{\theta_j} \quad (5.11)$$

which we can be rewritten as

$$\begin{aligned} \mathcal{C}_{mNPC}(\mathcal{X}, \Theta) &= \frac{1}{n_d} \sum_{i=1}^{n_d} \frac{1}{\log(2)} \log \left[\sum_{j=1}^{n_p} \left(\delta_c^{c(x_i)} \delta_{\theta_{s(i)}}^{\theta_j} + \frac{1}{n_p} \right) \right] \\ &= \frac{1}{n_d} \sum_{i=1}^{n_d} \frac{1}{\log(2)} \log \left[\sum_{j=1}^{n_p} \mathcal{G}(\mathfrak{d}(i, j)) \right]. \end{aligned} \quad (5.12)$$

This can be shown as follows: In case of a correctly classified data point x_i there exist exactly one prototype θ_k such that this prototype is the winning prototype $\theta_k = \theta_{s(i)}$ and this prototype has the same class as the data point

$c(\theta_k) = c(x_i)$. Thus

$$\mathcal{G}(\mathfrak{d}(i, k)) = 1 + \frac{1}{n_p}$$

and

$$\mathcal{G}(\mathfrak{d}(i, j)) = \frac{1}{n_p}, \forall j \neq k$$

hold. Further

$$\begin{aligned} \frac{1}{\log(2)} \log \left[\sum_{j=1}^{n_d} \mathcal{G}(\mathfrak{d}(x_i, \theta_j)) \right] &= \frac{1}{\log(2)} \log [\mathcal{G}(\mathfrak{d}(x_i, \theta_1)) + \dots \\ &\quad + \mathcal{G}(\mathfrak{d}(x_i, \theta_k)) + \dots + \mathcal{G}(\mathfrak{d}(x_i, \theta_{n_p}))] \\ &= \frac{1}{\log(2)} \log \left[\frac{1}{n_p} + \dots + \left(1 + \frac{1}{n_p}\right) + \dots + \frac{1}{n_p} \right] \\ &= \frac{1}{\log(2)} \log \left[n_p \frac{1}{n_p} + 1 \right] \\ &= \frac{1}{\log(2)} \log(2) \\ &= 1 \end{aligned}$$

contributes to the objective function in this case. In case of a wrongly classified data point there does not exist a prototype θ_k such that $\theta_k = \theta_{s(i)}$ with

$c(\theta_k) = c(x_i)$ is valid. Hence, we get

$$\begin{aligned}
 \frac{1}{\log(2)} \log \left[\sum_{j=1}^{n_d} \mathcal{G}(\mathfrak{d}(x_i, \theta_j)) \right] &= \frac{1}{\log(2)} \log [\mathcal{G}(\mathfrak{d}(x_i, \theta_1)) + \dots \\
 &\quad + \mathcal{G}(\mathfrak{d}(x_i, \theta_k)) + \dots + \mathcal{G}(\mathfrak{d}(x_i, \theta_{n_p}))] \\
 &= \frac{1}{\log(2)} \log \left[\frac{1}{n_p} + \dots + \frac{1}{n_p} + \dots + \frac{1}{n_p} \right] \\
 &= \frac{1}{\log(2)} \log \left[n_p \frac{1}{n_p} \right] \\
 &= \frac{1}{\log(2)} \log(1) \\
 &= 0
 \end{aligned}$$

and therefore, this term does not contribute to the objective.

Applying the theory from Section 5.1 to the objective function (Equation (5.12)) and using again formal probabilities $g_{ij} > 0$ with

$$\sum_{j=1}^{n_p} g_{ij} = 1$$

we find the decomposition according to (4.27)

$$\mathcal{C}_{mNPC}(\mathcal{X}, \Theta) = \mathcal{L}_{mNPC}(\mathcal{X}, \Theta) + \mathcal{R}_{mNPC}(\mathcal{X}, \Theta)$$

where

$$\mathcal{L}_{mNPC}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} g_{ij} \log \left[\frac{\mathcal{G}(\mathfrak{d}(i, j))}{g_{ij}} \right]$$

is the respective lower bound. In the expectation step, we set the optimal probability scores as

$$\hat{g}_{i,j} = \frac{\mathcal{G}(\mathfrak{d}(i, j))}{\sum_{k=1}^{n_p} \mathcal{G}(\mathfrak{d}(i, k))}.$$

such that $\mathcal{L}_{mNPC}(\mathcal{X}, \Theta) = \mathcal{C}_{mNPC}(\mathcal{X}, \Theta)$ holds.

The algorithm is summarized in Figure 5.6 and is denoted as median NPC (mNPC).

generalized EM algorithm for mNPC

1. **Initialization:** initialize the assignments $\theta_j = x_k$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$

2. **for** $m = 1 : n_p$

- a) **Expectation (E-Step):** evaluate the probability scores

$$\hat{g}_{i,j} = \frac{\mathcal{G}(\mathfrak{d}(x_i, \theta_j))}{\sum_{k=1}^{n_p} \mathcal{G}(\mathfrak{d}(x_i, \theta_k))}$$

- b) **Maximization (M-Step):** evaluate the new assignment $\theta_m = x_w$ with

$$x_w = \arg \max_{x_l \in \mathcal{X} \cup \{x_k\}} \left[\sum_{i=1}^{n_d} \hat{g}_{i,j} \log \left[\frac{\mathcal{G}(\mathfrak{d}(x_i, \theta_j))}{\hat{g}_{i,j}} \right] \right]_{\theta_m = x_l}$$

and fixed probability scores $\hat{g}_{i,j}$

endfor

3. **Convergence check:** evaluate the objective function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else start again with step 2.

Figure 5.6.: generalized EM algorithm for mNPC

5.2.4. Experiments

We tested the algorithms mGLVQ1, mRSLVQ and mNPC for several data sets and compare them with standard kernel/relational approaches like k-NN, SVM and kernel/relational LVQ. We used mGLVQ1 instead mGLVQ2 because of the numerical reasons mentioned in Section 5.2.2. For comparison reasons we chose data sets which were already used in [47] and [42] for relational and kernel methods for classification. These benchmark sets are proximity data, which are given in form of proximity matrices. In general, these matrices are not psd and therefore a preprocessing is required if kernel methods are used [42]. The used data sets are as follows:

1. **Voting:** The Voting dataset contains 435 samples splitted into 2 classes, representing categorical data. Each data point represents the votes of a U.S.House of Representatives Congressmen to 16 different problems. Each data point includes 16 features, the voting results (yes,no). The goal is to classify democrats and republicans. The data points are compared based on the value difference metric [42]. The Signature of the data is $\Sigma = (16, 1, 418)$ meaning the data is almost Euclidean.
2. **Aural:** The Aural Sonar data set consists of 100 sonar signals out of two classes (target of interest/clutter). The proximities were determined by an ad hoc classification of humans [42]. The signature of the data set is $\Sigma = (61, 38, 1)$ meaning the data is severely non-Euclidean.
3. **Protein:** The Protein data set consists of 213 data points from 4 classes, representing globin proteins. They are compared by an evolutionary measure [42]. The signature of the data set is $\Sigma = (169, 38, 6)$ meaning the data is moderately non-Euclidean.
4. **Face Recognition:** The Face Recognition data (FaceRec) set consists of 945 samples with 139 classes. The data represent faces of people, compared by the cosine similarity [42]. The signature of the data is $\Sigma = (45, 0, 900)$ meaning the data is Euclidean.

5. **Sonatas:** The Sonatas data set provides data similar to [81]. It contains proximities between 1068 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'³. Their mutual dissimilarities are measured with the normalized compression distance (NCD), see [82], which is applied to a specific preprocessing, which integrates invariances for music information retrieval, see [81]. The musical pieces are classified according to their composer. The signature of the data is $\Sigma = (1063, 4, 1)$ meaning the data is almost Euclidean.
6. **Chromosomes:** The Copenhagen Chromosomes data set constitutes a benchmark from cytogenetics [83]. A set of 4200 human chromosomes from 21 classes (the autosomal chromosomes) are represented by grey-valued images. These are transferred to strings measuring the thickness of their silhouettes. These strings are compared using edit distance [84]. The signature of the data set is $\Sigma = (1951, 2206, 43)$ meaning the data is severely non-Euclidean.

We applied the same setup as in [42], [47] for the experiments to make the results comparable: We performed an average over a repeated ten-fold cross validation. Therefore we used exactly the same splitting of the data as provided by [42], [47]. Training and test data were originally created as follows:

- for Voting, Aural, Protein, FaceRec:
 - 20% data were randomly selected for testing;
 - the remaining 80% were splitted into 10 disjoint training sets.

This process was repeated for 20 random partitions of the given data. It resulted in 200 training scenarios for each data sets.

- for Sonatas and Chromosomes:

³<http://www.kunstderfuge.com/>

Voting	Aural	Protein	FaceRec	Sonatas	Chromosomes
0.5	0.8	0.25	0.1	0.5	5

Table 5.1.: Values for the σ parameter in median RSLVQ

- 5% data were randomly selected for testing;
- the remaining 95% were splitted into 10 disjoint training sets.

This process was repeated for 20 random partitions. It results in 200 training scenarios for each data set.

For mRSLVQ we optimized the parameter σ within the cross validation scheme via grid search and report the results for the best setting. The final values are shown in Table 5.1.

The number of prototypes for median algorithms is chosen as a small multiple of the number of classes. We have used the number of prototypes of the kernel/relational variants as orientation. If it was sufficient to reach a comparable accuracy, the number of prototypes is smaller than in kernel/relational variants (e.g.: Aural Sonar and Protein dataset). A larger number of prototypes compared to kernel/relational is chosen if it is necessary for a better classification performance (e.g.: Chromosome dataset).

The results for k-NN, SVM and kernel/relational LVQ are taken from [47] and summarized in Table 5.2, with the standard deviations are reported in brackets. For kernel/relational methods we report the best solution of the algorithms kernel RSLVQ, kernel GLVQ, relational GLVQ and relational RSLVQ. In the last column, we report the number of prototypes which were used for kernel/relational LVQ. For k-NN the best result is reported regarding the number k of neighbors from $\{1, 3, 5\}$. In Table 5.3 we report our results for the three median LVQ variants. Generally, we can observe that the results for mGLVQ1, mRSLVQ and mNPC are similar, i.e. the results depend only slightly on the chosen objective function.

	k-NN	Relational/Kernel	SVM	Protos
Voting	95.34 (0.02)	94.7 (0.03)	95.1 (1.78)	20
Aural	85.25 (0.49)	88.8 (0.39)	88.0 (5.94)	10
Protein	92.2 (0.18)	98.6 (0.05)	98.0 (2.85)	20
FaceRec	92.7 (0.04)	96.6 (1.29)	96.3 (1.32)	139
Sonatas	89.5 (0.19)	84.9 (0.12)	89.1 (3.74)	5
Chromosom	96.2 (0.01)	95.7 (0.0)	97.6 (0.98)	21

Table 5.2.: Averaged test accuracies for k-NN, relational and kernel variants of LVQ and Support Vector Machines(SVM) taking the best data preprocessing from clip/flip/shift for SVM and kernel LVQ variants. The standard deviation is given in brackets. The last column contains the number of prototypes used for the kernel/relational variants.

	mRSLVQ	mGLVQ1	mNPC	Protos
Voting	95.6 (1.7)	95.6 (1.6)	95.8 (1.6)	20
Aural	91.0 (5.4)	90.7 (5.1)	89.9 (5.2)	6
Protein	91.2 (4.2)	90.4 (4.6)	92.2 (4.1)	4
FaceRec	98.6 (0.8)	98.7 (0.8)	98.7 (0.7)	139
Sonatas	80.5 (3.3)	80.8 (3.2)	79.9 (3.2)	5
Chromosom	85.4 (1.8)	88.9 (1.4)	88.7 (1.6)	105

Table 5.3.: Averaged test accuracies of median RSLVQ (mRSLVQ), median GLVQ (mGLVQ) and median NPC (mNPC) with standard deviations in brackets. The classification accuracy was produced by repeated 10-fold cross-validation with 10 repeats. The last column contains the number of prototypes, which was used.

Comparing the median variants with the other methods one can see a tendency that for smaller data sets (Voting, Aural, Protein) we are comparable in the results. For larger data sets (Sonatas, Chromosom) the median results are weaker. This is maybe caused by the fact that we have in the median case a very restricted solution space. Especially the kernel/relational methods benefit for these data sets from their richer solution space. However, the median algorithms are the only variants, which offer a sparse and interpretable solution as already discussed. If we have a focus on interpretable models, the loss of accuracy of less than 10% might be acceptable, depending on the practical use case.

Furthermore, looking closer at the results for the Aural data set, which is a small but severely non-Euclidean, the data preprocessing, needed for kernel LVQ and SVM, rises some loss of information. In this case, the median variants benefit from the fact that we can use very general proximities without Euclidean approximation.

Summarizing, the results show that median techniques are able to achieve comparable performance for most of the data sets. Since these methods can be used without data preprocessing also for non-Euclidean settings, and since they offer interpretable models in terms of few data exemplars, median LVQ variants provide a valuable alternative in settings where interpretability and sparsity constitutes crucial requirements.

5.3. Median Variants for Optimizing Statistical Classification Evaluation Measures behind Accuracy

On the following pages, we will discuss median variants based on general statistical classification evaluation measures. The ideas presented here are an adaptation of the ideas from the publications [85] and [86] considering here

proximity data.

We start with some useful abbreviations. We assume two-class problems with positive and negative labels $\{\oplus, \ominus\}$. Further let N^+ (N^-) the number of positive (negative) samples and $\tilde{N}^+$ ($\tilde{N}^-$) is the number of data points predicted as positive (negative) by a considered classifier. Further, we introduce a data dependent Kronecker abbreviation as short notation

$$\begin{aligned}\delta^+(x) &= \begin{cases} 1 & \text{if } c(x) = \oplus \\ 0 & \text{if } c(x) = \ominus \end{cases} \\ \delta^-(x) &= \begin{cases} 1 & \text{if } c(x) = \ominus \\ 0 & \text{if } c(x) = \oplus \end{cases}\end{aligned}$$

which is similar defined for prototypes as

$$\begin{aligned}\delta^+(\theta) &= \begin{cases} 1 & \text{if } c(\theta) = \oplus \\ 0 & \text{if } c(\theta) = \ominus \end{cases} \\ \delta^-(\theta) &= \begin{cases} 1 & \text{if } c(\theta) = \ominus \\ 0 & \text{if } c(\theta) = \oplus. \end{cases}\end{aligned}$$

Assume an arbitrary classification model and a fixed set of data samples $\mathcal{X}$. The positive (negative) data examples classified as positive (negative) are named as true positive (true negative). The regarding amount of data with this property is denoted as TP (TN). The same applies to positive data which is classified as negative (false negatives (FN)) and negative data which is classified as positive (false positive (FP)). These values constitute the so-called confusion matrix (see Table 5.4). Many statistical evaluation measures are based on the entries of the confusion matrix. As an example the accuracy

$$acc = \frac{TP + TN}{n_d}$$

		True Label		Σ
		$\oplus$	$\ominus$	
Predicted	$\oplus$	TP	FP	$\tilde{N}^+$
Label	$\ominus$	FN	TN	$\tilde{N}^-$
Σ		N^+	N^-	n_d

Table 5.4.: confusion matrix

is the relative number of correctly classified data points. To describe an arbitrary classification evaluation measure based on the confusion matrix as objective function for a prototype based classifier we have to find a description of these quantities in dependence on the prototype set $\Theta = \{\theta_1, \dots, \theta_m\}$. Obviously, these measures could be take also as objective for a self-learning classifier model. In this thesis, we only consider RSLVQ and GLVQ based classifiers.

Description based on RSLVQ In section 2.2.2 we found $p(c_k|x_i, \Theta)$ (2.8) as the probabilistic class assignments. In the line of the abbreviations above we rewrite (2.8) as

$$p(\oplus|x_i, \Theta) = \frac{\sum_j \delta^+(\theta_j) p(x_i|\theta_j)}{\sum_k p(x_i|\theta_k)} \quad (5.13)$$

as probability that a data point is classified as positive and

$$p(\ominus|x_i, \Theta) = \frac{\sum_j \delta^-(\theta_j) p(x_i|\theta_j)}{\sum_k p(x_i|\theta_k)} \quad (5.14)$$

is the probability that a data point is classified as negative. Using the probability (5.13) and (5.14) we can estimate the entries of the confusion matrix by

the probabilistic variants

$$pTP = \sum_{i=1}^{n_d} \delta^+(x_i) p(\oplus|x_i, \Theta) \quad (5.15)$$

$$pTN = \sum_{i=1}^{n_d} \delta^-(x_i) p(\ominus|x_i, \Theta) \quad (5.16)$$

$$pFP = \sum_{i=1}^{n_d} \delta^-(x_i) p(\oplus|x_i, \Theta) \quad (5.17)$$

$$pFN = \sum_{i=1}^{n_d} \delta^+(x_i) p(\ominus|x_i, \Theta). \quad (5.18)$$

Thus, we result a probabilistic interpretation of the confusion matrix. Yet, the values become crisp in the limit $\sigma \rightarrow 0$, because

$$p(\oplus|x_i, \Theta) \xrightarrow{\sigma \rightarrow 0} \begin{cases} 1, & \text{if } c(x_i) = \oplus \\ 0, & \text{else} \end{cases} \quad (5.19)$$

$$p(\ominus|x_i, \Theta) \xrightarrow{\sigma \rightarrow 0} \begin{cases} 1, & \text{if } c(x_i) = \ominus \\ 0, & \text{else} . \end{cases} \quad (5.20)$$

is valid. Hence, we can use the RSLVQ quantities to model every component of the confusion matrix.

Description based on GLVQ The classifier function of the GLVQ (see subsection 2.2.1) can also be modified to count correctly and incorrectly classified data points. For this purpose, let

$$\mathfrak{d}^\oplus(x_i) = \min_{\theta_k: c(\theta_k) = \oplus} \mathfrak{d}(x_i, \theta_k)$$

be the smallest dissimilarity from data point x_i to any prototype θ_k belonging to the positive class, and in an analogous way

$$\mathfrak{d}^\ominus(x_i) = \min_{\theta_k: c(\theta_k) = \ominus} \mathfrak{d}(x_i, \theta_k)$$

is the smallest dissimilarity from the data point to any prototype responsible for the negative class. Using the GLVQ classifier function (see Equation (2.4)) and the sigmoid as activation function f_a (see Equation (2.5)) we can formulate smooth scores for the class dependency of an data point as ⁴

$$p(\oplus|x_i, \Theta) = f_a \left(\frac{\mathfrak{d}^\ominus(x_i) - \mathfrak{d}^\oplus(x_i)}{\mathfrak{d}^\oplus(x_i) + \mathfrak{d}^\ominus(x_i)} \right) \quad (5.21)$$

$$p(\ominus|x_i, \Theta) = f_a \left(\frac{\mathfrak{d}^\oplus(x_i) - \mathfrak{d}^\ominus(x_i)}{\mathfrak{d}^\oplus(x_i) + \mathfrak{d}^\ominus(x_i)} \right). \quad (5.22)$$

Using these scores we can estimate the quantities TP, TN, FP and FN according to (5.15), (5.16), (5.17), (5.18) for the GLVQ model.

Moreover, from the discussion in subsection 2.2.1 about the sigmoid $\text{sig}(z)$ we know that the functions becomes the Heaviside function for $\sigma \rightarrow 0$ meaning that equation (5.19) and equation (5.20) are also valid for the GLVQ-based variant.

In the following, we do not distinguish between both models any more keeping in mind that we can represent the entries of the confusion matrix by those two models.

Simple Quality Measures For the set of simple measures like true positive rate (TPR), true negative rate (TNR), positive predicted value (PPV) and negative predicted value (NPV), which are all directly based on the confusion matrix, we give only one detailed example. All other results will be listed in a table.

⁴Because of the fact that we are looking for a consistent designation we also abbreviate the score with p .

Let us construct the median variant for the positive predictive value (PPV)

$$PPV = \frac{TP}{TP + FP} = \frac{TP}{\tilde{N}^+}$$

which is also referred as the precision. The objective function for this measure can be written as

$$\mathcal{C}_{PPV}(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} \mathcal{G}(x_i, \Theta)$$

with

$$\mathcal{G}(x_i, \Theta) = \frac{\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{i=1}^{n_d} \delta^+(x_i)p(\oplus|x_i, \Theta) + \sum_{i=1}^{n_d} \delta^-(x_i)p(\oplus|x_i, \Theta)}.$$

Thus, we can take

$$\tilde{\mathcal{C}}_{PPV}(\mathcal{X}, \Theta) = \log \left(\sum_{i=1}^{n_d} \mathcal{G}(x_i, \Theta) \right)$$

for the optimization via gEM using the monotonicity of the log-function. Hence, we get for the optimal probability scores $\hat{g}_i = \hat{g}(x_i)$ ⁵, used in the expectation step

$$\begin{aligned} \hat{g}_i &= \frac{\mathcal{G}(x_i, \Theta)}{\sum_{i=1}^{n_d} \mathcal{G}(x_i, \Theta)} \\ &= \frac{\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{i=1}^{n_d} \delta^+(x_i)p(\oplus|x_i, \Theta)}. \end{aligned}$$

Hence, the decomposition according to (4.27) becomes

$$\mathcal{C}_{PPV}(\mathcal{X}, \Theta) = \mathcal{L}_{PPV}(\mathcal{X}, \Theta) + \mathcal{R}_{PPV}(\mathcal{X}, \Theta)$$

⁵Note that in the following the probability scores g_i do not depend explicitly on a prototype θ_j and, therefore, are indexed only by the explicit data point x_i .

statistical measure	definition	$\mathcal{G}(x_i, \Theta)$
true positive rate (TPR), sensitivity or recall	$\frac{TP}{TP+FN} = \frac{TP}{N^+}$	$\frac{\delta^+(x_i)p(\oplus x_i, \Theta)}{N^+}$
true negative rate (TNR) or specificity	$\frac{TN}{TN+FP} = \frac{TN}{N^-}$	$\frac{\delta^-(x_i)p(\ominus x_i, \Theta)}{N^-}$
positive predictive value (PPV) or precision	$\frac{TP}{TP+FP} = \frac{TP}{\tilde{N}^+}$	$\frac{\delta^+(x_i)p(\oplus x_i, \Theta)}{\tilde{N}^+}$
negative predictive value (NPV)	$\frac{TN}{TN+FN} = \frac{TN}{\tilde{N}^-}$	$\frac{\delta^-(x_i)p(\ominus x_i, \Theta)}{\tilde{N}^-}$

Table 5.5.: simple statistical measures

with the lower bound

$$\begin{aligned}
\mathcal{L}_{PPV}(\mathcal{X}, \Theta) &= \sum_{i=1}^{n_d} g_i \log \left(\frac{\mathcal{G}(x_i, \Theta)}{g_i} \right) \\
&= \sum_{i=1}^{n_d} g_i \log \left(\frac{\delta^+(x_i)p(\oplus|x_i, \Theta)}{g_i \left(\sum_{i=1}^{n_d} \delta^+(x_i)p(\oplus|x_i, \Theta) + \sum_{i=1}^{n_d} \delta^-(x_i)p(\oplus|x_i, \Theta) \right)} \right)
\end{aligned}$$

used in the maximization step.

Thus, we obtain a median algorithm, which optimizes the positive predictive value. In a similar way, it is possible to optimize the true positive rate (TPR), true negative rate (TNR) and the negative predictive value (NPV) using their definitions and respective $\mathcal{G}(x_i, \Theta)$ functions depicted in Table 5.5. In the following, we turn to more complex statistical evaluation measures. They all are based on the entries of the confusion matrix. To avoid tedious repetitions, we just give the idea/definition of the measure, the associated objective function for gEM and the resulting optimal probability scores $\hat{g}(x_i)$ for the usage in gEM-optimization.

F_β -Measure The F_β -Measure introduced by [87] is a measure combining precision and recall

$$\begin{aligned} F_\beta &= \frac{(1 + \beta^2) \cdot TPR \cdot PPV}{\beta^2 \cdot PPV + TPR} \\ &= \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 FN + FP}. \end{aligned} \quad (5.23)$$

For $\beta = 1$ the F_β -Measure is the harmonic mean of precision and recall. For $\beta > 1$ we have a larger influence of the recall and the other way around for $\beta \in (0, 1)$ the precision will be more weighted. Using equation (5.23) the objective function can be found as

$$\mathcal{C}_{F_\beta}(\mathcal{X}, \Theta) = \log \left(\sum_{i=1}^{n_d} \mathcal{G}(x_i, \Theta) \right)$$

with

$$\mathcal{G}(x_i, \Theta) = \frac{(1 + \beta^2)\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{j=1}^{n_d} ((1 + \beta^2)\delta^+(x_j)p(\oplus|x_j, \Theta)) + \beta^2\delta^+(x_j)p(\ominus|x_j, \Theta) + \delta^-(x_j)p(\oplus|x_j, \Theta)}.$$

The optimal formal probability scores $\hat{g}_i = \hat{g}(x_i)$ resulting to

$$\hat{g}_i = \frac{(1 + \beta^2)\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{j=1}^{n_d} (1 + \beta^2)\delta^+(x_j)p(\oplus|x_j, \Theta)}.$$

Jaccard Index The Jaccard Index, as introduced in [88], is defined as

$$J = \frac{TP}{FP + TP + FN}.$$

The regarding objective function can be found as

$$\mathcal{C}_J(\mathcal{X}, \Theta) = \log \left(\sum_{i=1}^{n_d} \mathcal{G}(x_i, \Theta) \right)$$

with

$$\mathcal{G}(x_i, \Theta) = \frac{\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{j=1}^{n_d} \delta^-(x_j)p(\oplus|x_j, \Theta) + \delta^+(x_j)p(\oplus|x_j, \Theta) + \delta^+(x_j)p(\ominus|x_j, \Theta)}.$$

The optimal formal probability scores $\hat{g}_i = \hat{g}(x_i)$ resulting to

$$\hat{g}_i = \frac{\delta^+(x_i)p(\oplus|x_i, \Theta)}{\sum_{j=1}^{n_d} \delta^+(x_j)p(\oplus|x_j, \Theta)}.$$

G-Measure The G-Measure, also known as Fowlkes-Mallows index [89] is defined as [90]

$$\begin{aligned} G &= \sqrt{PPV \cdot TPR} \\ &= \sqrt{\frac{TP}{TP + FP} \frac{TP}{TP + FN}} \end{aligned}$$

i.e. it is the geometric mean of precision and recall. We can formulate a respective objective function as

$$\mathcal{C}_G(\mathcal{X}, \Theta) = \sqrt{\sum_{i=1}^{n_d} \sum_{j=1}^{n_d} \mathcal{G}(x_i, x_j)}$$

with

$$\mathcal{G}(x_i, x_j, \Theta) = \frac{\delta^+(x_i)\delta^+(x_j)p(\oplus|x_i, \Theta)p(\oplus|x_j, \Theta)}{\tilde{N}^+ N^+}.$$

Now, we use the fact that the square root is a positive and concave function (see Section 4.2.2) to obtain the optimal formal probability scores $\hat{g}(x_i, x_j) = \hat{g}_{i,j}$ ⁶ as

$$\hat{g}_{i,j} = \frac{\delta^+(x_i)\delta^+(x_j)p(\oplus|x_i, \Theta)p(\oplus|x_j, \Theta)}{\sum_{k=1}^{n_d} \sum_{l=1}^{n_d} \delta^+(x_k)\delta^+(x_l)p(\oplus|x_k, \Theta)p(\oplus|x_l, \Theta)}$$

⁶Note, in this case the probability scores $\hat{g}(x_i, \theta_j)$ depend on two data points x_i, x_j and therefore will be abbreviated as $\hat{g}_{i,j}$. Please do not be confused with the earlier scores g_{ij} , where the index j referred to the index of prototype θ_j .

and the lower bound becomes

$$\begin{aligned}\mathcal{L}(\mathcal{X}, \Theta) &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_d} g_{i,j} \sqrt{\frac{\mathcal{G}(x_i, x_j, \Theta)}{g_{i,j}}} \\ &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_d} g_{i,j} \sqrt{\frac{\delta^+(x_i) \delta^+(x_j) p(\oplus|x_i, \Theta) p(\oplus|x_j, \Theta)}{g_{i,j} \tilde{N}^+ N^+}}.\end{aligned}$$

Further measures like Mathews correlation [91] are considered in [78].

Area under the ROC curve The receiver operating characteristics (ROC) curve is a graphical tool for comparison of binary classifiers [92] and the area under this curve can be used as quality measure for comparison of classifiers [93], [94].

To formalize the basic ideas regarding LVQ classifiers let

$$\mathcal{P}_{\mathcal{X}} = \{(x_i, x_j) | c(x_i) = \oplus, c(x_j) = \ominus\}, \quad x_i, x_j \in \mathcal{X}$$

the set of ordered pairs of data objects with different labels. Further $\gamma \in [0, 1)$ is a threshold parameter such that a data object x_i is classified as positive if

$$p(\oplus|x_i, \Theta) > \gamma \tag{5.24}$$

holds and for

$$p(\oplus|x_i, \Theta) \leq \gamma \tag{5.25}$$

it is classified to be negative. For any fixed value of γ , the classifier which is obtained using the decision functions (5.24), (5.25), delivers corresponding estimates for true positive rate TPR and false positive rate FPR . Varying the value for γ and plotting the respective values for TPR (as y-axis) and FPR (as x-axis) in a 2-dimensional diagram results in the ROC curve. The area under

this curve (AUROC) has the useful probabilistic interpretation

$$AUROC = P(p(\oplus|x_i, \Theta) > p(\oplus|x_j, \Theta))$$

for a randomly chosen pair $(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}$ [92], which is valid due the underlying rank statistics [95], [96]. Using the so called ordering function

$$O(x_i, x_j) = \mathbb{H}(p(\oplus|x_i, \Theta) - p(\oplus|x_j, \Theta)),$$

where $\mathbb{H}$ is the Heaviside function, we can estimate this probability via (see e.g.[86])

$$AUROC \approx A_{ROC} = \frac{1}{|\mathcal{P}_{\mathcal{X}}|} \sum_{(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}} O(x_i, x_j). \quad (5.26)$$

for any finite data set $\mathcal{X}$. In order to maximize the AUROC we can describe a corresponding objective function as

$$\begin{aligned} \mathcal{C}_{AUROC}(\mathcal{X}, \Theta) &= \log \left(\sum_{(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}} \mathcal{G}(x_i, x_j, \Theta) \right) \\ &= \log \left(\sum_{(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}} O(x_i, x_j) \right). \end{aligned}$$

Resulting from this objective we get the optimal formal probability scores $\hat{g}_{i,j} = \hat{g}(x_i, x_j)$

$$\begin{aligned} \hat{g}_{i,j} &= \frac{\mathcal{G}(x_i, x_j, \Theta)}{\sum_{(x_k, x_l) \in \mathcal{P}_{\mathcal{X}}} \mathcal{G}(x_k, x_l, \Theta)} \\ &= \frac{O(x_i, x_j)}{\sum_{(x_k, x_l) \in \mathcal{P}_{\mathcal{X}}} O(x_k, x_l)} \end{aligned}$$

and the lower bound becomes

$$\begin{aligned}\mathcal{L}(\mathcal{X}, \Theta) &= \sum_{(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}} g_{i,j} \log \left(\frac{\mathcal{G}(x_i, x_j, \Theta)}{g_{i,j} \sum_{(x_k, x_l) \in \mathcal{P}_{\mathcal{X}}} \mathcal{G}(x_k, x_l, \Theta)} \right) \\ &= \sum_{(x_i, x_j) \in \mathcal{P}_{\mathcal{X}}} g_{i,j} \log \left(\frac{O(x_i, x_j)}{g_{i,j} \sum_{(x_k, x_l) \in \mathcal{P}_{\mathcal{X}}} O(x_k, x_l)} \right)\end{aligned}$$

such that we can optimize the AUROC via the given objective using the ideas of the gEM algorithm.

To sum up we have shown for many statistical classification evaluation measures how we can optimize them using the gEM. Moreover, due the reason that we need only pairwise dissimilarities to evaluate the label probabilities (see equations (5.13), (5.14), (5.21), (5.22)), we keep the median idea to optimize this measures. Numerical studies for these algorithms can be found in [79] and [78].

5.4. Mixtures of Proximities

The application of various (dis-)similarity measures in a classification model usually leads to different results. For a given classification task it is usually unknown, which proximity measure is most appropriate or whether a combination of several proximities provides better results. Successfully applied approaches working around this problem are based on ensemble learning or fusion models (e.g. [97], [98]). One idea for ensemble learning is to train the same class of classifier with different proximities [99], [100]. Beside its successful application, this approach is time consuming because we have to train a set of classifiers instead of one. Other possibilities are provided by kernel averaging or scalable multiple kernel learning [101], [102]. Here, we want to adopt the ideas of combining dissimilarities from [103].

We start with an illustrative toy example to motivate the idea behind the mix-

ing approach. The artificial data set, referred as Triangle Data (TD), consists of two-dimensional data belonging to three overlapping classes (see Figure 5.7). The classes are generated by multivariate Gaussians with the covariance matri-

Figure 5.7.: Triangle data set with three classes displayed in yellow, red and blue

ces

$$\Sigma_1 = \begin{pmatrix} 20 & -18 \\ -18 & 20 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 50 & 0 \\ 0 & 1 \end{pmatrix}, \Sigma_3 = \begin{pmatrix} 20 & 18 \\ -18 & 20 \end{pmatrix}$$

and with the respective means $\mu_1 = (10, 7)^T$, $\mu_2 = (0, 0)^T$, $\mu_3 = (-10, 7)^T$. The number of data examples per class are $N_1 = N_3 = 500$ and $N_2 = 1000$. For the experiments, we calculated the dissimilarities as class-specific Mahalanobis

distances [104]

$$\mathfrak{d}^{\Sigma_k}(x_i, x_j) = \sqrt{(x_i - x_j)^T \Sigma_k^{-1} (x_i - x_j)}, \quad k = 1, 2, 3.$$

As baseline we trained a mGLVQ with one prototype per class on all three distances which results in 93.5% ($\pm 1.2\%$) accuracy for $\mathfrak{d}^{\Sigma_1}$, 91.6% ($\pm 0.6\%$) accuracy for $\mathfrak{d}^{\Sigma_2}$ and 93.0% ($\pm 1.0\%$) accuracy for $\mathfrak{d}^{\Sigma_3}$. These results are obtained by averaging over 15 runs, where we randomly split the data in 80% train and 20% test data. As a second baseline we trained a mGLVQ, with the same setting, on a fixed linear combination of the three distances $\mathfrak{d}^{\text{flC}} = \frac{1}{3}\mathfrak{d}^{\Sigma_1} + \frac{1}{3}\mathfrak{d}^{\Sigma_2} + \frac{1}{3}\mathfrak{d}^{\Sigma_3}$, which results the accuracy of 95% ($\pm 0.8\%$). Due to the construction of the data set, the use of the local distances $\mathfrak{d}^{\Sigma_k}$ is of course a local reflection of the individual class distributions, whereas the use of the linear combination is a more global perspective. The results are therefore not surprising but the question arises whether the naively chosen linear combination is a good one or if we can do even better. To investigate this question, we merge the dissimilarities in a general bilinear mixing measure

$$\mathfrak{d}_\Lambda(x_i, x_j) = \mathbf{d}(x_i, x_j)^T \Lambda_M \mathbf{d}(x_i, x_j)$$

where $\Lambda_M \in \mathbb{R}^{m \times m}$ is a positive (semi-)definite mixing matrix and $\mathbf{d}(x_i, x_j) = (\mathfrak{d}_1(x_i, x_j), \dots, \mathfrak{d}_m(x_i, x_j))^T$ denotes the vector of all dissimilarities under consideration. Setting Λ_M as diagonal matrix with positive entries λ_{kk} the bilinear mixture reduces to a linear combination

$$\begin{aligned} \mathfrak{d}_\lambda(x_i, x_j) &= \sum_{k=1}^m \lambda_k \mathfrak{d}_k(x_i, x_j) \\ &= \sum_{k=1}^m \sqrt{\lambda_{kk}} \mathfrak{d}_k(x_i, x_j). \end{aligned}$$

with the abbreviation $\lambda_k = \sqrt{\lambda_{kk}}$. Rewriting the mixing matrix as $\Lambda_M = \Omega^T \Omega$ ($\Omega \in \mathbb{R}^{l \times m}$) ensures that Λ_M is psd and leads to the mixed dissimilarity measure

$$\mathfrak{d}_\Omega(x_i, x_j) = \langle \Omega \mathbf{d}(x_i, x_j), \Omega \mathbf{d}(x_i, x_j) \rangle_E \quad (5.27)$$

whereby the entries of the matrix Ω can serve as free parameters for optimization.

We can plug the mixing dissimilarities in every presented median classifier. For optimization, we propose a two-step scheme. The prototypes are adapted in one stage using the proposed median methods and in the other stage the mixing coefficients are tuned using stochastic gradient ascent learning (SGAL).

We illustrate this approach explicitly for mGLVQ. The mGLVQ is based on the objective function $\mathcal{C}(\mathcal{X}, \Theta)$ (5.10) and the corresponding median scheme (Figure: 5.5). We take the prototype assignment as fixed at a certain step of optimization and want to adopt the objective regarding the mixing coefficients. The objective function writes as

$$\mathcal{C}_f(\mathcal{X}, \Theta) = \sum_{i=1}^{n_d} f_a(-\kappa(x_i)) \quad (5.28)$$

$$= \sum_{i=1}^{n_d} f_a \left(\frac{\mathfrak{d}_\Omega^-(x_i) - \mathfrak{d}_\Omega^+(x_i)}{\mathfrak{d}_\Omega^+(x_i) + \mathfrak{d}_\Omega^-(x_i)} \right) \quad (5.29)$$

using the mixed dissimilarities. To realize the SGAL, we determine the local gradient with respect Ω , and obtain for the update

$$\Delta \Omega = 2 \frac{\partial f_a(\kappa(x_i))}{\kappa(x_i)} \frac{\left(\frac{\partial \mathfrak{d}_\Omega^-(x_i)}{\partial \Omega} \mathfrak{d}_\Omega^+(x_i) - \frac{\partial \mathfrak{d}_\Omega^+(x_i)}{\partial \Omega} \mathfrak{d}_\Omega^-(x_i) \right)}{(\mathfrak{d}_\Omega^+(x_i) + \mathfrak{d}_\Omega^-(x_i))^2}$$

with

$$\frac{\partial \mathfrak{d}_\Omega^\pm(x_i)}{\partial \Omega} = 2 \Omega \mathbf{d}^\pm(x_i) (\mathbf{d}^\pm(x_i))^T.$$

Thus, we can update the mixing matrix in every single step of the SGAL via

$$\Omega = \Omega + \alpha \Delta \Omega.$$

The learning rate α requires the same conditions as for GLVQ discussed in subsection 2.2.1.

We summarize the optimization algorithm in the pseudo code in Figure 5.8. Applying this algorithm to the triangle data, using the same setting as for the

alternating algorithm for mGLVQ with mixing dissimilarity

1. **Initialization:**

- initialize the assignments $x_k \leftarrow \theta_j$ of prototypes to data objects $\forall j = \{1, \dots, n_p\}$
- initialize the mixing matrix Ω

2. for fixed Ω optimize the prototype assignments via the EM algorithm Figure 5.5

3. for fixed prototype assignments optimize the mixing matrix Ω by SGAL

4. **Convergence check:** evaluate the objective function and check for convergence. If

$$\mathcal{C}(\mathcal{X}, \Theta^{(\text{new})}) = \mathcal{C}(\mathcal{X}, \Theta^{(\text{old})})$$

then stop. Else start again with step 2.

Figure 5.8.: alternating algorithm for mGLVQ with mixing dissimilarity

baseline results, we consider two kinds of classifiers: One classifier with Λ_M is a diagonal matrix and the second with a full bilinear mixture $\Lambda_M = \Omega^T \Omega$.

For the first, linear model we achieved an accuracy of 95.5% ($\pm 0.8\%$) which exceeds the baseline results. The corresponding averaged mixing coefficients are $(\lambda_1, \lambda_2, \lambda_3) = (0.406, 0.373, 0.221)$ with standard deviations $(0.043, 0.045, 0.031)$. In the second experiment we achieved an accuracy of 96.5% ($\pm 0.3\%$) which further improves the linear results. The mixing matrix

$$\Lambda_M = \begin{pmatrix} 0.352 & 0.22 & 0.136 \\ 0.22 & 0.384 & 0.084 \\ 0.136 & 0.084 & 0.264 \end{pmatrix}$$

is displayed in Figure 5.9, as useful visualization.

Figure 5.9.: Mixing matrix Λ_M for the triangle data set

Real World Example The coffee data set (CD) [105] as a real world example consists of 256-band hyperspectra with equidistant bands in the range between 970 nm and 2500 nm, i.e. the data dimensionality is $D = 256$. The data consists of 5 different coffee classes with 500 data samples for each class. As dissimilarity measures we used the squared Euclidean distance $\mathfrak{d}_E^2$, the Sobolev

distance

$$\mathfrak{d}_S(x_i, x_j) = (1 - \alpha) \sqrt{\sum_{t=1}^D (x_i(t) - x_j(t))^2} + \alpha \sqrt{\sum_{t=1}^D (\dot{x}_i(t) - \dot{x}_j(t))^2}$$

where $\dot{x} = \frac{\partial x}{\partial t}$ ⁷ and $\alpha \in [0, 1]$ [106] and the γ -divergences

$$\begin{aligned} \mathfrak{d}^\gamma(x_i, x_j) &= \frac{1}{1 + \gamma} \log \left[\left(\sum_{t=1}^D x_i(t)^{\gamma+1} \right)^{\frac{1}{\gamma}} \left(\sum_{t=1}^D x_j(t)^{\gamma+1} \right)^{\frac{1}{\gamma}} \right] \\ &\quad - \log \left[\left(\sum_{t=1}^D x_i(t)^{\gamma+1} x_j(t)^{\gamma+1} \right)^{\frac{1}{\gamma}} \right] \end{aligned}$$

for several parameter choices γ ⁸. For demonstration purposes we also consider the dissimilarity based on the Gaussian kernel $\mathfrak{d}_{\text{Gauss}}$, for which we already know from section 3.2 that it is rank equivalent to the squared Euclidean distance.

For the experiments, we used the following parametrization

- for the Sobolev distance we set $\alpha = 1$, i.e. we take only the information of the derivative into account
- for the Gaussian kernel distance we obtained $\sigma = 0.25$ as best parameter via grid search
- for the γ -distance we choose the parameters $\gamma \in \{0.25, 1, 1.5\}$

The visualization of the resulting dissimilarity matrices for a subset of 100 randomly chosen exemplars per class can be found in Figure: 5.10.

We also evaluated the rank matrices corresponding to the dissimilarities which are depicted in Figure 5.11. The comparison of the rank matrices using the normalized rank equivalence measure (see equation (3.33)) is displayed in Figure: 5.12.

⁷Approximated by the first order forward difference quotient.

⁸Note that for $\gamma = 1$ the γ -divergence becomes the Cauchy-Schwarz divergence [48]

As expected, the squared Euclidean dissimilarity is rank equivalent to the dissimilarity based on the Gaussian Kernel also to the Cauchy-Schwarz divergence. The most excited dissimilarity is the Sobolev dissimilarity, which differs most from all other dissimilarities in terms of the rank equivalence measure.

The training setup for mGLVQ is the same as that for the triangle data. We did 5-fold cross validation, meaning we split the data set in 80% training and 20% testing data for 5 times.

All reported results are the average over the test results for the five runs. Comparable to the TD data set we evaluate the baselines by training classifiers for the single dissimilarities (see Table: 5.6). The single results are on the same level except the result for the Sobolev dissimilarity, which also showed already a significant discrepancy regarding the ranks. As a second baseline, we trained

Coffee	$\mathfrak{d}_E$	$\mathfrak{d}_S$	$\mathfrak{d}_{\text{Gauss}}$	$\mathfrak{d}_{0.25}^\gamma$	$\mathfrak{d}_{CS}$	$\mathfrak{d}_{1.5}^\gamma$
acc	76.3	72.3	76.2	76.2	76.7	76.8
$\pm$ std	1.2	1.2	1.6	1.1	0.8	1.9

Table 5.6.: Average classification test accuracy with standard deviation (in %) for the Coffee data set (5-fold-CV).

classifiers with a fixed linear combination, where all λ_k chosen equally. The achieved averaged accuracy over the five runs is 81.8% with a standard deviation of 1.5%. This is a significant improvement comparing the single results. As for the triangle experiment we also trained a model with a linear mixing, taking the mixing matrix as diagonal matrix, and a model with a bilinear mixing, taking the mixing matrix as full matrix. With the first one we achieve an average accuracy of 82.7% and a standard deviation of 0.9%. The trained average mixing coefficients are

$$(\lambda_{\mathfrak{d}_E}, \lambda_{\mathfrak{d}_S}, \lambda_{\mathfrak{d}_{\text{Gauss}}}, \lambda_{\mathfrak{d}_{0.25}^\gamma}, \lambda_{\mathfrak{d}_{CS}}, \lambda_{\mathfrak{d}_{1.5}^\gamma}) = (0.236, 0.118, 0, 0.235, 0.16, 0.251).$$

We achieved a further improvement but we can see that the influence of the

Figure 5.10.: Dissimilarity matrices for the coffee data set calculated for 100 random data points per class. From the top left to the bottom right we have the squared Euclidean distance, Sobolev distance, the dissimilarity based on the RBF kernel, the Cauchy-Schwarz divergence and the γ -divergence for $\gamma = 0.25$ and $\gamma = 1.5$.

Figure 5.11.: Rank matrices for the different dissimilarity measures based on the same data as in Figure 5.10

Figure 5.12.: Matrix of rank equivalence measures between all dissimilarities evaluated on the coffee data set

dissimilarity based on the Gaussian kernel is vanished completely. This implies that for this case the usage of two rank equivalent dissimilarities, the Euclidean distance and the Gaussian based dissimilarity, is not necessary. The Sobolev based dissimilarity, which has delivered a weak result in the single cases, has now a significant influence in the linear combination. This shows that a single dissimilarity with weak performance can contribute to better results in combination with other proximities.

Using the full bilinear mixing we achieved an average accuracy of 83.4% with a standard deviation of 1.0%. The resulting mixing matrix is illustrated in Figure 5.13. We can conclude from there that the resulting improvement can be dedicated to the general linear combination of the several proximities. This is in analogy to the comparison of generalized Matrix LVQ (GMLVQ) and generalized relevance LVQ (GRLVQ) [37], [107] where input dimensions are combined/weighted instead of proximity measures.

Figure 5.13.: The mixing matrix Λ obtained from the adaptive bilinear mixing (bLM) for the coffee data set.

6. Summary and Conclusion

In this thesis, we focused mainly on the extension of the well-known EM principle for probabilistic models to more general not necessarily probabilistic scenarios. The resulting gEM algorithm allows deriving median variants of learning vector quantization for proximity data.

For this purpose, we started with a systematic introduction of essential types of proximities including similarities, dissimilarities and inner products. This taxonomy extends the scheme provided by *Pekalska&Duin*. In particular, we introduced more subtypes according to the properties of dissimilarity measures. Moreover, we introduced respective similarity measures as counterparts to these subtypes. These similarity types are in agreement with the concept in cognitive psychology regarding similarities of objects as discussed by *Tversky*. Related to this taxonomy we considered the question of how to compare different measures in dependence on their behavior for a given data set. The derived quantity also allows comparing the discrimination ability of prototype based methods.

Another main objective of the thesis was to generalize the standard EM approach in such a way that it becomes applicable for more general proximity data with the focus to apply the framework for probabilistic Learning Vector Quantization. The developed gEM approach was motivated by consideration of GMM and the mathematical derivation of the resulting algorithm is considered and discussed in detail. One result of these investigations is a new batch variant of neural gas based on gEM. Further median variants of prototype based classifiers like GLVQ, RSLVQ and NPC are derived using gEM. These variants extend the application range of those methods dramatically because many real classification problems do not rely on continuous data. By means of the new

algorithms the relational approaches, which implicitly assume continuous data distributions, are extended to handle true discrete data.

Further, the developed methodologies allow to adapt those classifiers to more sophisticated cost functions than accuracy as shown for several statistical quality measures. These measures include true positive(negative) rate, positive(negative) predictive value, F_β -Measure, Jaccard Index, G-Measure and the Area under the ROC curve.

In particular, we have the following main results in the chapters:

- Chapter 3 gives a systematic characterization of proximity measures and provides a respective mathematical taxonomy. This taxonomy allows a comparison of general proximity measures. A respective tool was developed using topological rank properties.
- Chapter 4 is dedicated to the mathematical derivation of the gEM scheme for optimization of general cost functions. For this purpose, a general decomposition strategy was developed to decompose general cost functions into two parts one of them reflecting a lower bound for the cost function during each optimization step.
- Chapter 5 applies the developed gEM scheme to several prototype based classifiers including RSLVQ and GLVQ. Doing so these methods become applicable for general proximity data. These methods are denoted as median variants. Further, this strategy is applied for these algorithms when more sophisticated cost functions than simple accuracy are used. Finally, we considered in this chapter the usage of mixtures of proximity measures in median LVQ variants.

In summary, we have shown that the EM principle is a flexible tool for optimizing objective functions in very general settings. The formal mathematical processing of this method allows an extension to very general use cases in supervised and unsupervised learning as we have shown in the thesis. However,

usually these approaches require proximity measures to compare data and prototypes. As it was demonstrated, a careful choice of those measures is strongly demanded because it heavily influences the behavior of the algorithms as well as the resulting outcome and performance. The provided tool for measure comparison could be used to make an appropriate selection in advance. However, application specific requirements may restrict this choice. In that case should be kept in mind that the assumed object properties maybe corrupted or violated.

A disadvantage of median methods is the restricted solution space. Therefore, median methods frequently deliver weaker results for continuous problems than respective continuous methods. Further, median variants may suffer from stability and convergence problems, which could prevent an application in crucial and sensitive tasks. Several open problems and future directions of research can be identified while studying this thesis and may serve as potential research foci's:

- If a proximity matrix is given it is not clear which mathematical characteristics allow to categorize the unknown underlying proximity in general. An open question is for which problems the similarity or the dissimilarity view would be advantageous.
- Assuming that a proximity has some certain properties for fixed data objects, new arriving data points maybe violate these properties. It should be investigated what general characteristics contribute to those violations.
- It would be interesting to see whether the introduced comparison measure for proximities could be directly transferred to the application for inner products and kernels. This is of particular interest because kernel methods are still one of the most successful approaches in machine learning also in the period of deep networks.
- The introduced median variants suffer from high complexity in case of

huge number of data. For relational methods, Patch approaches are known to deal with this problem. A respective methodology should be easily develop-able also for the median variants.

Nederlandse samenvatting

In dit proefschrift concentreerden we ons hoofdzakelijk op de uitbreiding van het welbekende EM principe voor probabilistische modellen naar meer generieke, niet noodzakelijk probabilistische scenario's. Het resulterende gEM algoritme maakt de afleiding van mediaanvarianten van lerende vectorkwantisatie mogelijk voor nabijheidsdata.

Met deze doelstelling begonnen we met een systematische introductie van essentiële typen nabijheidsmaten, inclusief gelijkheidsmaten, ongelijkheidsmaten en inproducten. Deze taxonomie breidt het schema uit van Pekalska en Duin. In het bijzonder introduceren we meer subtypen volgens de eigenschappen van ongelijkheidsmaten. Bovendien introduceren we desbetreffende gelijkheidsmaten als tegenhangers van deze subtypen. Deze gelijkheidsmaten zijn in overeenstemming met het concept in de cognitieve psychologie betreffende gelijkheidsmaten van objecten zoals besproken door Tversky. Gerelateerd aan deze taxonomie behandelden we de vraag hoe we verschillende maten kunnen vergelijken betreffende hun gedrag op een bepaalde dataset. De afgeleide maat maakt het ook mogelijk om het onderscheidingsvermogen van op prototype gebaseerde methoden te vergelijken.

Een andere hoofddoelstelling van dit proefschrift was om de standaard EM benadering te veralgemeniseren op zo'n manier dat het toepasbaar wordt voor algemenere nabijheidsdata met het belangrijke punt om het raamwerk toe te passen voor probabilistische lerende vectorkwantisatie. De ontwikkelde gEM benadering werd gemotiveerd door de in acht neming van GMM en de mathematische afleiding van het resulterende algoritme wordt behandeld en besproken in detail. Een resultaat van deze onderzoeken is een nieuwe batch variant van neuraal gas gebaseerd op gEM. Verdere mediaanvarianten van prototype-gebaseerde classificeerders zoals GLVQ, RSLVQ en NPC worden afgeleid met gEM. Deze varianten breiden het toepassingsgebied van deze methoden enorm uit omdat veel reële classificatieproblemen niet steunen op continue data. Door middel van de nieuwe algoritmen worden de relationele benaderingen, die im-

pliciet continue dataverdelingen aannemen, uitgebreid om ook om te gaan met discrete data.

Verder maken de ontwikkelde methodologieën het mogelijk om die classificeerder aan te passen aan geavanceerdere kostenfuncties dan nauwkeurigheid zoals getoond voor verschillende statistische kwaliteitsmaten. Deze maten bevatten ware positieve (negatieve) percentages, positieve (negatieve) voorspelde waarde, F_β -Maat, Jaccard Index, G-Maat en de oppervlakte onder de ROC kromme.

In het bijzonder hebben we de volgende hoofdresultaten in de hoofdstukken:

- Hoofdstuk 3 geeft een systematische karakterisering van nabijheidsmaten en geeft een desbetreffende mathematische taxonomie. Deze taxonomie maakt een vergelijking van algemene nabijheidsmaten mogelijk. Een desbetreffend hulpmiddel werd ontwikkeld met topologische rangeigenschappen.
- Hoofdstuk 4 is toegewijd aan de wiskundige afleiding van het gEM schema voor de optimalisatie van algemene kostenfuncties. Voor dit doeleinde werd een algemene decompositiestrategie ontwikkeld om algemene kostenfuncties te ontleden in twee gedeelten, een daarvan een ondergrens van de kostenfunctie gevend in iedere optimalisatiestap.
- Hoofdstuk 5 past het ontwikkelde gEM schema toe op verschillende prototypegebaseerde classificeerders, inclusief RSLVQ en GLVQ. Deze methoden worden dan toepasbaar voor algemene nabijheidsdata. Deze methoden worden aangeduid als mediaanvarianten. Verder wordt deze strategie toegepast voor deze algoritmen wanneer geavanceerdere kostenfuncties dan simpele nauwkeurigheid gebruikt worden. Tenslotte behandelden we in dit hoofdstuk het gebruik van samenvoegingen van nabijheidsmaten in mediaan LVQ varianten.

Samenvattend, we hebben laten zien dat het EM principe een flexibel hulpmiddel is voor het optimaliseren van objectieve functies in zeer algemene scenario's. De formele wiskundige verwerking van deze methode maakt een uitbrei-

ding mogelijk naar heel algemene gebruikssituaties in gesuperviseerd en onge-superviseerd leren zoals we hebben laten zien in het proefschrift. Echter, deze benaderingen vereisen nabijheidsmaten om data te vergelijken en prototypen. Zoals gedemonstreerd, een zorgvuldige keuze van deze maten is zeer vereist omdat het een grote invloed heeft op het gedrag van de algoritmen alsook de resulterende uitkomst en de prestatie. Het gegeven hulpmiddel voor het vergelijken van maten kan worden gebruikt om een geschikte selectie van te voren te maken. Echter, toepassingsspecifieke eisen kunnen deze keuze beperken. In dat geval zouden we ons eraan moeten herinneren dat de aangenomen objecteigenschappen geschonden zijn. Een nadeel van mediaanmethoden is de beperkte oplossingsruimte. Daarom leveren mediaanmethoden vaak zwakkere resultaten voor continue problemen dan desbetreffende continue. Verder lijden mediaanvarianten aan stabiliteits- en convergentieproblemen, wat een toepassing in krietie en gevoelige taken kan beletten.

Verscheidene open problemen en toekomstige onderzoeksrichtingen kunnen worden onderscheiden bij het lezen van dit proefschrift en kunnen dienen als potentiële onderzoeksfoci:

- Als een nabijheidsmatrix gegeven is, is het niet duidelijk welke wiskundige karakteristieken het mogelijk maken om de onbekende onderliggende nabijheid in het algemeen te categoriseren. Een open vraag is voor welke problemen de gelijkheids of de ongelijkheidsbenadering beter is.
- Aannemend dat een nabijheid een zeker aantal eigenschappen heeft voor vaststaande objecten, nieuwe datapunten die erbij komen kunnen deze eigenschappen schenden. Het zou onderzocht moeten worden welke algemene karakteristieken bijdragen aan die schendingen.
- Het zou interessant zijn om te zien of de geïntroduceerde vergelijkingsmaat voor nabijheden direct overdragen kan worden naar de toepassing voor inproducten en kernels. Dit is van groot belang omdat kernelmetho-den nog steeds een van de meest succesvolle benaderingen in machine learning zijn, ook in de periode van diepe netwerken.

- De geïntroduceerde mediaanvarianten leiden onder een hoge complexiteit in het geval van een enorme hoeveelheid data. Voor relationele methoden, patch benaderingen staan bekend om om te gaan met dit probleem. Een desbetreffende methodologie zou ook gemakkelijk te ontwikkelen moeten zijn voor mediaanvarianten.

A. Appendix

A.1. Decomposition Proof

Proof of the decomposition (4.18) from subsection 4.1.2, i.e. the goal is to show that $\mathcal{L}(g) + \mathcal{K}(g||p) = \ell_{\log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X})$ holds.

$$\begin{aligned}
 \mathcal{L}(g) + \mathcal{K}(g||p) &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j)p(x_i|j)}{g_j(x_i)} \right] \\
 &\quad - \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j|x_i)}{g_j(x_i)} \right] \\
 &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [p(j)p(x_i|j)] \\
 &\quad - \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [p(j|x_i)] \\
 &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln [p(j)p(x_i|j)] \\
 &\quad - \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\frac{p(j)p(x_i|j)}{\sum_{l=1}^{n_c} p(l)p(x_i|l)} \right] \\
 &= \sum_{i=1}^{n_d} \sum_{j=1}^{n_c} g_j(x_i) \ln \left[\sum_{l=1}^{n_c} p(l)p(x_i|l) \right] \\
 &= \sum_{i=1}^{n_d} \underbrace{\left(\sum_{j=1}^{n_c} g_j(x_i) \right)}_{=1} \ln \left[\sum_{l=1}^{n_c} p(l)p(x_i|l) \right] \\
 &= \sum_{i=1}^{n_d} \ln \left[\sum_{l=1}^{n_c} p(l)p(x_i|l) \right] \\
 &= \ell_{\log}(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathcal{X})
 \end{aligned}$$

A.2. Proof for $\mathcal{R}(\mathcal{X}, \Theta) = 0$

Proof that for the optimal choice

$$\hat{g}_{ij} = \frac{\mathcal{G}(x_i, \theta_j)}{\sum_{k,l} \mathcal{G}(x_k, \theta_l)}$$

$\mathcal{C}_f(\mathcal{X}, \Theta) = \mathcal{L}(\mathcal{X}, \Theta)$ holds in equation (4.27) and, therefore, $\mathcal{R}(\mathcal{X}, \Theta) = 0$ is valid.

$$\begin{aligned}
 \mathcal{L}(\mathcal{X}, \Theta) &= \sum_{i,j} \hat{g}_{ij} f\left(\frac{\mathcal{G}(x_i, \theta_j)}{\hat{g}_{ij}}\right) \\
 &= \sum_{i,j} \hat{g}_{ij} f\left(\frac{\mathcal{G}(x_i, \theta_j)}{\frac{\mathcal{G}(x_i, \theta_j)}{\sum_{k,l} \mathcal{G}(x_k, \theta_l)}}\right) \\
 &= \sum_{i,j} \hat{g}_{ij} f\left(\sum_{k,l} \mathcal{G}(x_k, \theta_l)\right) \\
 &= f\left(\sum_{k,l} \mathcal{G}(x_k, \theta_l)\right) \underbrace{\sum_{i,j} \hat{g}_{ij}}_{=1} \\
 &= f\left(\sum_{k,l} \mathcal{G}(x_k, \theta_l)\right) \\
 &= \mathcal{C}_f(\mathcal{X}, \Theta)
 \end{aligned}$$

A.3. Derivation for prototype update (Neural Gas)

Derivation of the prototype update from condition 4.33 in subsection 4.2.3, i.e. consideration of $\frac{\partial}{\partial \theta_k} \mathcal{L}(\mathcal{X}, \Theta) = 0$. For the update step the ranks are fixed such that we can shorten $k_i(x, \theta_j)$ as k_{ij} to emphasize that the ranks are constant.

$$\begin{aligned}
 \frac{\partial}{\partial \theta_k} \mathcal{L}(\mathcal{X}, \Theta) &= \underbrace{\frac{\partial}{\partial \theta_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \left(-\frac{k_{ij}}{\lambda} \right)}_{=0} + \frac{\partial}{\partial \theta_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \left(-\frac{d^2(x_i, \theta_j)}{2\sigma^2} \right) \\
 &\quad - \underbrace{\frac{\partial}{\partial \theta_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \ln(g_{ij})}_{=0} \\
 &= \frac{\partial}{\partial \theta_k} \sum_{i=1}^{n_d} \sum_{j=1}^{n_p} g_{ij} \left(-\frac{d^2(x_i, \theta_j)}{2\sigma^2} \right) \\
 &= -\frac{1}{2\sigma^2} \sum_{i=1}^{n_d} g_{ik} \frac{\partial}{\partial \theta_k} d^2(x_i, \theta_k) \\
 &= -\frac{1}{\sigma^2} \sum_{i=1}^{n_d} g_{ik} (x_i - \theta_k) \\
 &\stackrel{!}{=} 0
 \end{aligned}$$

Thus we find

$$\begin{aligned}
 \sum_{i=1}^{n_d} g_{ik} \theta_k &= \sum_{i=1}^{n_d} g_{ik} x_i \\
 \theta_k &= \frac{\sum_{i=1}^{n_d} g_{ik} x_i}{\sum_{i=1}^{n_d} g_{ik}}
 \end{aligned}$$

as new prototypes for fixed g_{ik} .

Publications

Journal Papers

- [1] T. Villmann, M. Kaden, D. Nebel, M. Riedel. Lateral enhancement in adaptive metric learning for functional data. *Neurocomputing 131* (2014). pp. 23-31.
- [2] M. Kaden, M. Lange, D. Nebel, M. Riedel, T. Geweniger and T Villmann. Aspects in classification learning-Review of recent developments in Learning Vector Quantization. *Foundations of Computing and Decision Sciences 39.2* (2014). pp. 79-105.
- [3] T. Zok. M. Antczak, M. Riedel, D. Nebel, T. Villmann, P. Lukasiak, J. Blazewicz, M. Szachniuk. Building the library of RNA 3D nucleotide conformations using the clustering approach. *International Journal of Applied Mathematics and Computer Science 25.3* (2015). pp. 689-700.
- [4] D. Nebel, B. Hammer, K. Frohberg and T. Villmann. Median variants of learning vector quantization for learning of dissimilarity data. *Neurocomputing, 169*, (2015). pp. 295-305.

- [5] D. Nebel, M. Kaden, A. Villmann and T. Villmann. Types of (dis-)similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, 268 (2017): pp. 42-54.

Conference Papers

- [1] T. Villmann, M. Kästner, D. Nebel, M. Riedel. ICMLA Face Recognition Challenge—Results of the Team Computational Intelligence Mitweida. *2012 11th International Conference on Machine Learning and Applications* (2012). pp. 592-595.
- [2] M. Kästner, D. Nebel, M. Riedel, M. Biehl, T. Villmann. Differentiable kernels in generalized matrix learning vector quantization. *2012 11th International Conference on Machine Learning and Applications* (2012). pp. 132-137.
- [3] D. Nebel and T. Villmann. About the equivalence of robust soft learning vector quantization and soft nearest prototype classification. *Workshop New Challenges in Neural Computation* (2013). pp. 114-118.
- [4] D. Nebel, B. Hammer and T. Villmann. A median variant of generalized learning vector quantization. *International Conference on Neural Information Processing* (2013). pp. 19-26.
- [5] D. Nebel, B. Hammer and T. Villmann. Supervised Generative Models for Learning Dissimilarity Data. *ESANN, 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2014). pp. 35-40.

- [6] B. Hammer, D. Nebel, M. Riedel and T. Villmann. Enforcing interpretability in classification by modeling constrained optimization problems using LVQ decision rules. *Workshop New Challenges in Neural Computation* (2014). pp.52.
- [7] M. Lange, D. Nebel, T. Villmann. Partial mutual information for classification of gene expression data by learning vector quantization. *Advances in Self-Organizing Maps and Learning Vector Quantization* (2014). pp. 259-269.
- [8] L. Fischer, D. Nebel, T. Villmann, B. Hammer and H. Wersing. Rejection strategies for learning vector quantization a comparison of probabilistic and deterministic approaches. *Advances in Self-Organizing Maps and Learning Vector Quantization* (2014). pp. 109-118.
- [9] B. Hammer, D. Nebel, M. Riedel and T. Villmann. Generative versus discriminative prototype based classification. *Advances in Self-Organizing Maps and Learning Vector Quantization* (2014). pp. 123-132.
- [10] M. Lange, D. Nebel, T. Villmann. Non-Euclidean principal component analysis for matrices by Hebbian learning. *International Conference on Artificial Intelligence and Soft Computing*. (2014). pp. 77-88.
- [11] T. Villmann, M. Kaden, D. Nebel, M. Biehl. Learning vector quantization with adaptive cost-based outlier-rejection. *International Conference on Computer Analysis of Images and Patterns*. (2015). pp. 772-782.

- [12] D. Nebel and T. Villmann. Median-LVQ for classification of dissimilarity data based on ROC-optimization. *ESANN, 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2015). pp. 1-6.
- [13] S. Saralajew, D. Nebel, T. Villmann. Adaptive Hausdorff distances and tangent distance adaptation for transformation invariant classification learning. *International Conference on Neural Information Processing*. (2016). pp. 362-371.
- [14] T. Villmann, M. Kaden, D. Nebel and A. Bohnsack. Similarities, dissimilarities and types of inner products for data analysis in the context of machine learning. *International Conference on Artificial Intelligence and Soft Computing* (2016). pp. 125-130
- [15] M. Kaden, D. Nebel and T. Villmann. Adaptive dissimilarity weighting for prototype-based classification optimizing mixtures of dissimilarities. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2016). pp. 135-140
- [16] D. Nebel and T. Villmann. Optimization of Statistical Evaluation Measures for Classification by Median Learning Vector Quantization. *Advances in Self-Organizing Maps and Learning Vector Quantization*. (2016). pp. 281-291.

- [17] M. Kaden, D. Nebel, F. Melchert, A. Backhaus, U. Seiffert and T. Villmann. Data dependent evaluation of dissimilarities in nearest prototype vector quantizers regarding their discriminating abilities. *Advances in Self-Organizing Maps and Learning Vector Quantization*. (2017). pp. 1-7.

- [18] T. Villmann, J. Ravichandran, A. Villmann, D. Nebel, M. Kaden. Investigation of Activation Functions for Generalized Learning Vector Quantization. *International Workshop on Self-Organizing Maps*. (2019). pp. 179-188.

Bibliography

- [1] David Reinsel, John Gantz, and John Rydning. “The digitization of the world: from edge to core”. In: *Framingham: International Data Corporation* (2018).
- [2] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. 14. Oakland, CA, USA. Elsevier, 1967, pp. 281–297.
- [3] James C Bezdek, Robert Ehrlich, and William Full. “FCM: The fuzzy c-means clustering algorithm”. In: *Computers & Geosciences* 10.2-3 (1984), pp. 191–203.
- [4] Thomas M. Martinetz, Stanislav G. Berkovich, and Klaus J. Schulten. “‘Neural-gas’ network for vector quantization and its application to time-series prediction”. In: *IEEE Transactions on Neural Networks* 4 (1993), pp. 558–569.
- [5] Brendan J Frey and Delbert Dueck. “Clustering by passing messages between data points”. In: *Science* 315.5814 (2007), pp. 972–976.
- [6] G.J. McLachlan, D. Peel, and P. Prado. “Clustering via normal mixture models”. In: *Proceedings of the American Statistical Association*. 1997, pp. 98–103.
- [7] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.
- [8] Teuvo Kohonen. “The self-organizing map”. In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

- [9] T. Kohonen. *Learning Vector Quantization for Pattern Recognition*. Tech. rep. 1986.
- [10] Chaofan Chen, Oscar Li, Chaofan Tao, et al. “This looks like that: deep learning for interpretable image recognition”. In: *arXiv preprint arXiv:1806.10574* (2018).
- [11] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels*. MIT Press, Cambridge, MA, 2002.
- [12] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [13] Robert P. W. Duin and Elzbieta Pekalska. *Dissimilarity Representation For Pattern Recognition, The: Foundations And Applications*. Vol. 64. World scientific, 2005.
- [14] Paul Zador. “Asymptotic quantization error of continuous signals and the quantization dimension”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 139–149.
- [15] Thomas Villmann and Jens Christian Claussen. “Magnification control in self-organizing maps and neural gas”. In: *Neural Computation* 18.2 (2006), pp. 446–469.
- [16] Tom Heskes. “Energy functions for self-organizing maps”. In: *Kohonen Maps* (1999), pp. 303–316.
- [17] Thomas Martinetz and Klaus Schulten. *A "Neural-Gas" Network Learns Topologies*. University of Illinois, 1991.
- [18] Samuel Kaski. “Data exploration using self-organizing maps”. In: *Acta Polytechnica Scandinavica: Mathematics, Computing and Management in Engineering, Series No. 82*. Citeseer. 1997.
- [19] Kadim Tasdemir and Erzsébet Merényi. “Exploiting data topology in visualization and clustering of self-organizing maps”. In: *IEEE Transactions on Neural Networks* 20.4 (2009), pp. 549–562.
- [20] Thomas Villmann, Erzsébet Merényi, and Barbara Hammer. “Neural maps in remote sensing image analysis”. In: *Neural Networks* 16.3-4 (2003), pp. 389–403.

- [21] Georges Voronoi. “Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites.” In: *Journal für die reine und angewandte Mathematik* 133 (1908), pp. 97–178.
- [22] Helge Ritter, Thomas Martinetz, Klaus Schulten, et al. *Neural computation and self-organizing maps: an introduction*. Addison Wesley Longman Publishing Co., Inc., 1992.
- [23] Evelyn Fix and Joseph Lawson Hodges. “Discriminatory analysis. Nonparametric discrimination: consistency properties”. In: *International Statistical Review/Revue Internationale de Statistique* 57.3 (1989), pp. 238–247.
- [24] Thomas Cover and Peter Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27.
- [25] Michael Biehl, Barbara Hammer, and Thomas Villmann. “Prototype-based models in machine learning”. In: *Wiley Interdisciplinary Reviews: Cognitive Science* 7.2 (2016), pp. 92–111.
- [26] Teuvo Kohonen. “Learning vector quantization”. In: *Self-Organizing Maps*. Springer, 1995, pp. 175–189.
- [27] Atsushi Sato and Keiji Yamada. “Generalized learning vector quantization”. In: *Advances in Neural Information Processing Systems* (1996), pp. 423–429.
- [28] Sambu Seo, Mathias Bode, and Klaus Obermayer. “Soft nearest prototype classification”. In: *IEEE Transactions on Neural Networks* 14.2 (2003), pp. 390–398.
- [29] Sambu Seo and Klaus Obermayer. “Soft learning vector quantization”. In: *Neural Computation* 15.7 (2003), pp. 1589–1604.
- [30] Koby Crammer, Ran Gilad-Bachrach, Amir Navot, et al. “Margin analysis of the LVQ algorithm”. In: *Advances in Neural Information Processing Systems*. 2002, pp. 462–469.

- [31] Thomas Villmann, John Ravichandran, Andrea Villmann, et al. “Activation Functions for Generalized Learning Vector Quantization—A Performance Comparison”. In: *arXiv preprint arXiv:1901.05995* (2019).
- [32] Marika Kaden, Martin Riedel, Wieland Hermann, et al. “Border-sensitive learning in generalized learning vector quantization: an alternative to support vector machines”. In: *Soft Computing* 19.9 (2015), pp. 2423–2434.
- [33] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The Annals of Mathematical Statistics* (1951), pp. 400–407.
- [34] Siegfried Graf and Harald Luschgy. *Foundations of quantization for probability distributions*. Springer, 2007.
- [35] M Kaden, M Lange, D Nebel, et al. “Aspects in classification learning—Review of recent developments in Learning Vector Quantization”. In: *Foundations of Computing and Decision Sciences* 39.2 (2014), pp. 79–105.
- [36] Sascha Saralajew, Lars Holdijk, Maike Rees, et al. “Prototype-based neural network layers: incorporating vector quantization”. In: *arXiv preprint arXiv:1812.01214* (2018).
- [37] Petra Schneider, Michael Biehl, and Barbara Hammer. “Adaptive relevance matrices in learning vector quantization”. In: *Neural Computation* 21.12 (2009), pp. 3532–3561.
- [38] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.
- [39] Marc Strickert, Frank-Michael Schleif, Udo Seiffert, et al. “Derivatives of Pearson correlation for gradient-based analysis of biomedical data”. In: *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* 12.37 (2008).

- [40] Thomas Villmann and Sven Haase. “Divergence-based vector quantization”. In: *Neural Computation* 23.5 (2011), pp. 1343–1392.
- [41] Sascha Saralajew, David Nebel, and Thomas Villmann. “Adaptive Hausdorff distances and tangent distance adaptation for transformation invariant classification learning”. In: *International Conference on Neural Information Processing*. Springer. 2016, pp. 362–371.
- [42] Yihua Chen, Eric K Garcia, Maya R Gupta, et al. “Similarity-based classification: Concepts and algorithms”. In: *Journal of Machine Learning Research* 10.Mar (2009), pp. 747–776.
- [43] Amos Tversky and Itamar Gati. “Similarity, separability, and the triangle inequality.” In: *Psychological Review* 89.2 (1982), p. 123.
- [44] D Nebel, M Kaden, A Villmann, et al. “Types of (dis-) similarities and adaptive mixtures thereof for improved classification learning”. In: *Neurocomputing* 268 (2017), pp. 42–54.
- [45] JE Driskell and T McDonald. “Identification of incomplete networks”. In: *Florida Maxima Corporation Technical Report* 8 (2008).
- [46] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, et al. “Soft similarity and soft cosine measure: Similarity of features in vector space model”. In: *Computación y Sistemas* 18.3 (2014), pp. 491–504.
- [47] Barbara Hammer, Daniela Hofmann, Frank-Michael Schleif, et al. “Learning vector quantization for (dis-) similarities”. In: *Neurocomputing* 131 (2014), pp. 43–51.
- [48] Sven Haase. *The Application of Divergences in Prototype Based Vector Quantization*. University of Groningen, 2014.
- [49] John Robilliard Giles. “Classes of semi-inner-product spaces”. In: *Transactions of the American Mathematical Society* 129.3 (1967), pp. 436–446.
- [50] Günter Lumer. “Semi-inner-product spaces”. In: *Transactions of the American Mathematical Society* 100.1 (1961), pp. 29–43.

- [51] János Bognár. *Indefinite inner product spaces*. Vol. 78. Springer Science & Business Media, 1974.
- [52] Á. G. Horváth. “Semi-indefinite inner product and generalized Minkowski spaces”. In: *Journal of Geometry and Physics* 60.9 (2010), pp. 1190–1208.
- [53] Frank-Michael Schleif and Peter Tino. “Indefinite proximity learning: A review”. In: *Neural Computation* 27.10 (2015), pp. 2039–2096.
- [54] Andrea Bohnsack, Kristin Domaschke, Marika Kaden, et al. “Learning matrix quantization and relevance learning based on Schatten-p-norms”. In: *Neurocomputing* 192 (2016), pp. 104–114.
- [55] Mandy Lange, David Nebel, and Thomas Villmann. “Non-Euclidean principal component analysis for matrices by Hebbian learning”. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2014, pp. 77–88.
- [56] Mandy Lange, Michael Biehl, and Thomas Villmann. “Non-Euclidean principal component analysis by Hebbian learning”. In: *Neurocomputing* 147 (2015), pp. 107–119.
- [57] Haizhang Zhang, Yuesheng Xu, and Jun Zhang. “Reproducing kernel Banach spaces for machine learning”. In: *Journal of Machine Learning Research* 10.Dec (2009), pp. 2741–2775.
- [58] BA J Mercer. “Functions of positive and negative type, and their connection the theory of integral equations”. In: *Phil. Trans. R. Soc. Lond. A* 209.441-458 (1909), pp. 415–446.
- [59] Ingo Steinwart. “On the influence of the kernel on the consistency of support vector machines”. In: *Journal of Machine Learning Research* 2.Nov (2001), pp. 67–93.
- [60] Marika Kaestner, David Nebel, Martin Riedel, et al. “Differentiable kernels in generalized matrix learning vector quantization”. In: *11th International Conference on Machine Learning and Applications (ICMLA), 2012*. Vol. 1. IEEE. 2012, pp. 132–137.

- [61] Ryo Inokuchi and Sadaaki Miyamoto. “LVQ clustering and SOM using a kernel function”. In: *Fuzzy Systems, 2004. Proceedings. 2004 IEEE International Conference on Fuzzy Systems*. Vol. 3. IEEE. 2004, pp. 1497–1500.
- [62] Maurice G Kendall. “A new measure of rank correlation”. In: *Biometrika* 30.1/2 (1938), pp. 81–93.
- [63] Benjamin Paaßen, Claudio Gallicchio, Alessio Micheli, et al. “Tree Edit Distance Learning via Adaptive Symbol Embeddings”. In: *arXiv preprint arXiv:1806.05009* (2018).
- [64] Nicholas J Higham. “Computing a nearest symmetric positive semidefinite matrix”. In: *Linear Algebra and its Applications* 103 (1988), pp. 103–118.
- [65] Gang Wu, Edward Y Chang, and Zhihua Zhang. “An analysis of transformation on non-positive semidefinite similarity matrix for kernel machines”. In: *Proceedings of the 22nd International Conference on Machine Learning*. Vol. 8. 2005.
- [66] Karl Pearson. “On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.
- [67] Joseph B Kruskal. “Nonmetric multidimensional scaling: a numerical method”. In: *Psychometrika* 29.2 (1964), pp. 115–129.
- [68] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1.5 (2019), p. 206.
- [69] Daniela Hofmann, Frank-Michael Schleif, Benjamin Paaßen, et al. “Learning interpretable kernelized prototype-based models”. In: *Neurocomputing* 141 (2014), pp. 84–96.
- [70] R. A. Fisher. “On an absolute criterion for fitting frequency curves.” In: *Messenger of Mathematics* (1912).

- [71] Davide Bacciu, Paulo JG Lisboa, Alessandro Sperduti, et al. “Probabilistic Modeling in Machine Learning”. In: *Springer Handbook of Computational Intelligence* (2015), pp. 545–575.
- [72] Léon Bottou. “Stochastic learning”. In: *Advanced Lectures on Machine Learning* (2004), pp. 146–168.
- [73] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd. New York: Springer, 2006.
- [74] Radford M Neal and Geoffrey E Hinton. “A view of the EM algorithm that justifies incremental, sparse, and other variants”. In: *Learning in Graphical Models*. Springer, 1998, pp. 355–368.
- [75] J. L. W. V. Jensen. “Sur les fonctions convexes et les inégalités entre les valeurs moyennes”. In: *Acta Mathematica* 30.1 (1906), pp. 175–193.
- [76] Marie Cottrell, Barbara Hammer, Alexander Hasenfuß, et al. “Batch and median neural gas”. In: *Neural Networks* 19.6 (2006), pp. 762–771.
- [77] David Nebel, Barbara Hammer, Kathleen Frohberg, et al. “Median variants of learning vector quantization for learning of dissimilarity data”. In: *Neurocomputing* 169 (2015), pp. 295–305.
- [78] D Nebel and T Villmann. “Optimization of Statistical Evaluation Measures for Classification by Median Learning Vector Quantization”. In: *Advances in Self-Organizing Maps and Learning Vector Quantization*.
- [79] D Nebel and T Villmann. “Median-LVQ for classification of dissimilarity data based on ROC-optimization”. In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2015)*. 2015, pp. 1–6.
- [80] PR Freeman. “The secretary problem and its extensions: A review”. In: *International Statistical Review/Revue Internationale de Statistique* (1983), pp. 189–206.

- [81] Bassam Mokbel, Alexander Hasenfuss, and Barbara Hammer. “Graph-based representation of symbolic musical data”. In: *International Workshop on Graph-Based Representations in Pattern Recognition*. 2009, pp. 42–51.
- [82] Rudi Cilibrasi and Paul MB Vitányi. “Clustering by compression”. In: *IEEE Transactions on Information Theory* 51.4 (2005), pp. 1523–1545.
- [83] Claes Lundsieen, John Philip, and Erik Granum. “Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes”. In: *Clinical Genetics* 18.5 (1980), pp. 355–370.
- [84] Michel Neuhaus and Horst Bunke. “Edit distance-based kernel functions for structural pattern classification”. In: *Pattern Recognition* 39.10 (2006), pp. 1852–1863.
- [85] M. Kaden, W. Hermann, and T. Villmann. “Optimization of General Statistical Accuracy Measures for Classification Based on Learning Vector Quantization.” In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2014)*. 2014.
- [86] T Villmann, M Kaden, W Hermann, et al. “Learning vector quantization classifiers for ROC-optimization”. In: *Computational statistics* 33.3 (2018), pp. 1173–1194.
- [87] C. J. Van Rijsbergen. “Information retrieval.” In: *Butterworths, London, 2nd edition* (1979).
- [88] Paul Jaccard. “The distribution of the flora in the alpine zone. 1”. In: *New Phytologist* 11.2 (1912), pp. 37–50.
- [89] Edward B Fowlkes and Colin L Mallows. “A method for comparing two hierarchical clusterings”. In: *Journal of the American Statistical Association* 78.383 (1983), pp. 553–569.
- [90] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. “On clustering validation techniques”. In: *Journal of Intelligent Information Systems* 17.2-3 (2001), pp. 107–145.

- [91] Brian W Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), pp. 442–451.
- [92] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.
- [93] Andrew P Bradley. “The use of the area under the ROC curve in the evaluation of machine learning algorithms”. In: *Pattern Recognition* 30.7 (1997), pp. 1145–1159.
- [94] Jin Huang and Charles X Ling. “Using AUC and accuracy in evaluating learning algorithms”. In: *IEEE Transactions on Knowledge and Data Engineering* 17.3 (2005), pp. 299–310.
- [95] Henry B Mann and Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The Annals of Mathematical Statistics* (1947), pp. 50–60.
- [96] Frank Wilcoxon. “Individual comparisons by ranking methods”. In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83.
- [97] Leijun Li, Qinghua Hu, Xiangqian Wu, et al. “Exploration of classification confidence in ensemble learning”. In: *Pattern Recognition* 47.9 (2014), pp. 3120–3131.
- [98] Yanning Zhang, Haichao Zhang, Nasser M Nasrabadi, et al. “Multi-metric learning for multi-sensor fusion based classification”. In: *Information Fusion* 14.4 (2013), pp. 431–440.
- [99] Uwe Knauer, Andreas Backhaus, and Udo Seiffert. “Beyond Standard Metrics—On the Selection and Combination of Distance Metrics for an Improved Classification of Hyperspectral Data”. In: *Advances in Self-Organizing Maps and Learning Vector Quantization*. Springer, 2014, pp. 167–177.
- [100] Uwe Knauer, Andreas Backhaus, and Udo Seiffert. “Fusion trees for fast and accurate classification of hyperspectral data with ensembles of γ -divergence-based RBF networks”. In: *Neural Computing and Applications* 26.2 (2015), pp. 253–262.

-
- [101] Fabio Aioli and Michele Donini. “EasyMKL: a scalable multiple kernel learning algorithm”. In: *Neurocomputing* 169 (2015), pp. 215–224.
 - [102] Lluís A Belanche and Alessandra Tosi. “Averaging of kernel functions”. In: *Neurocomputing* 112 (2013), pp. 19–25.
 - [103] Ernest Mwebaze, Gjalt Bearda, Michael Biehl, et al. “Combining dissimilarity measures for prototype-based classification”. In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2015)*. 2015, pp. 31–36.
 - [104] Prasanta Chandra Mahalanobis. “On the generalized distance in statistics”. In: *Proceedings of the National Institute of Science 12*. National Institute of Science of India. 1936.
 - [105] Andreas Backhaus, Felix Bollenbeck, and Udo Seiffert. “High-throughput quality control of coffee varieties and blends by artificial neural networks and hyperspectral imaging”. In: *Proceedings of the 1st International Congress on Cocoa, Coffee and Tea, CoCoTea*. 2011.
 - [106] Thomas Villmann. “Sobolev metrics for learning of functional data - mathematical and theoretical aspects”. In: *Machine Learning Reports* (2007).
 - [107] Barbara Hammer and Thomas Villmann. “Generalized relevance learning vector quantization”. In: *Neural Networks* 15.8-9 (2002), pp. 1059–1068.

David Nebel: *Median Variants of Prototype Based Learning Vector Quantization*,
PhD thesis, University of Groningen, 2020

ISBN: 978-94-034-2484-2 (paper)

ISBN: 978-94-034-2483-5 (eBook)