Bachelor's thesis - Multi-Class classification of functional data

Bachelor's Project Thesis

Emilio Oldenziel, s2509679, e.oldenziel.1@student.rug.nl,
Supervisors: prof. dr. M. Biehl & F. Melchert, MSc

Abstract: In classifying functional data we can take advantage of its functional properties by approximating the underlying function that it represents. This has been implemented using Chebyshev Polynomials [7]. Approximation of functional data reduces the amount of feature dimensions for the data. For generalized relevance learning vector quantization (GMLVQ) this results in speedup in learning and less overfitting. The classification results using GMLVQ are comparable and sometimes better than using the original feature vectors [7]. The base functions that are used for approximating the underlying function have different relevances per class. To make use of this we implemented local relevance matrices (LGMLVQ) as in [13] to extend the no-nonsense LVQ toolbox [5]. This resulted in an improvement of classification performance using multiple functional data sets in relation with the shape variance per class. The feature relevances show that the base functions which fit the shape of the underlying function better are more important. After the training process we translated the LGMLVQ relevance matrices results back to the original feature space which resulted in a similar result as with training with the data in the original dimension.

1 Introduction

A large quantity of data is gathered nowadays from things such as devices and sensors and most of this data has to be classified. Many unsupervised and supervised machine learning techniques exist to do this [14]. One of them is prototype based learning, where prototypes which are a representation of the classes and distance measures are used to classify data. An advantage of this technique is that it is very intuitive and straightforward to implement. [7]
When it comes to classification of data, advantage can be taken from the type of data that needs to be classified. In the case of classification of functional data, like timeseries or spectra, the functional nature of the data can be used in advantage of classification. In functional data, the order of features is relevant because the features are consecutive measurements in time or order. The approach that we use in this study is looking at high dimensional feature vectors containing functional data as a discretized representation of a continuous function [7]. The function has its own characteristics that can be used for classification.
To make use of these characteristics we can approximate the feature vector to retrieve the function as a combination of functions that it represents. In previous studies [7, 9] this was done using two-class classification problems and applying prototype based learning in the form of Generalized Matrix Learning Vector Quantization (GMLVQ) on the functional approximation of the input data. This resulted in equal or superior classification performances compared to the feature vector in the original dimension [7]. The biggest benefit from this is that the dimension of the data can be heavily reduced. This results in speedup of the training process and the approximation smoothens the function which results in less overfitting [7]. In this study we want to investigate what the classification performances are for real world multi-class problems. While doing this we also want to look at the extent to which the classification performance increases for functional data if local matrix LVQ (LGMLVQ) is used instead of GMLVQ.
To do this we used real world data sets to measure classification performances. The primary data set that we used is the sugar data set [8]. This data set consists of infrared spectra of 9 sugar types from various infrared-sensors.
The goal of this research is to discover what the rel-

evance results mean if we translate the results from the approximated dimension back to the original feature dimension of the dataset. We expect that the results will show resemblance with the results of training with the original input data. It will also contain relevances based on the functional nature of the data that come from the polynomial approximation.

2 Background Information

To improve the classification of multi-class functional data we considered the polynomial approximation of the data and the use of local matrices over global matrices. To understand where the performance improvement comes from we first have to explain the mathematical framework that is used.

2.1 Functional Data

With functional data other then regular data is defined by data with a specific order, this means that the order of the features matter. This data can also be described by, as the name says, functions. Examples of functional data are, time-series, spectrograms and sensor data. A characteristic that comes with functional data is that a single feature does not tell so much about the data, but a sequence or interval of features tells us about the shape or form of the data. This shape tells us more about the characteristics that the data has rather than a single feature.
To classify functional data we want to use the shape to get a better classification result instead of using the individual features. To do that we need to transform the data to a space where we get a description of its shape. A method to do this is, approximate the feature vectors or curve as what is represented with the use of base functions. We can combine the base functions in such a way that it approximates a feature vector as close as possible. With that we can obtain coefficients to describe the combination of the base functions.

2.1.1 Chebyshev Polynomials approximation

To approximate the curve we can use Chebyshev polynomials as base functions for the polynomial fitting. With a set of Chebyshev polynomials we can derive coefficients $c_{i,j}$ so that

$$f_i(x) = \sum_{k=0}^{\infty} c_{i,j} \cdot g_k(x), \tag{2.1}$$

where $f_i(x)$ is the curve that we want to approximate and $g_k(x)$ is the set of Chebyshev polynomials. Here we can limit k in the sum from infinity to a $n > 0$ but also $n < d$ where d is the length of the original feature vector. The coefficients $c_{i,j}$ are obtained by a linear transformation F from the original feature vector ξ_o by:

$$\xi_c = F \cdot \xi_o. \tag{2.2}$$

This results in a new feature vector ξ_c with $c^i \in \mathbb{R}^n$. By taking a number n less then 20 the approximation fits the general shape of the curve. This is a more smoothed version of the curve which helps with classification because overfitting is less likely to occur.

The Chebyshev Polynomials of the first kind that we use as base functions are described as:

$$T_0 = 1 \tag{2.3}$$
$$T_1 = x \tag{2.4}$$
$$T_n = 2xT_n(x) - T_{n-1}(x), \tag{2.5}$$

where n is the n^{th} polynomial. For this research we used the Chebyshev implementation of [7]. We ran the classification with multiple configurations of n. The previous study [7] concluded that we need at least around 20 polynomials to reach to the same classification scores as with the whole feature vectors. Therefore, we took at least 20 or more polynomials for classification.

2.2 Classification

2.2.1 Learning Vector Quantization

Learning vector quantization is a classification system that uses prototypes and distance measures to classify objects [1]. It is an supervised learning method introduced by Kohonen based on Self Organizing Maps [3]. The procedure begins with taking a number of prototypes per class (usually one) to represent the data in the class. This prototype

can be initialized randomly or at a more specific point, like the mean of the class of that prototype. To complete the initialization we also need to determine how fast we want our prototypes to learn, this is called a learning rate described by the symbol η.
After initialization we can start the training process, here we visit all training samples ξ^μ, at every training sample with class label y^μ we calculate the distance to all prototypes. This is done using a distance measure, in most cases the squared-Euclidean distance measure is used because it is a simple metric but other distance measures exist as well. Then, two prototypes are marked as winners. The first winner is the correct winner w_j^* which is the prototype that has the smallest distance to the training sample and where:

$$y_j^* = y^\mu. \tag{2.6}$$

The second winner is the incorrect winner that has smallest distance to the training sample and where:

$$y_k^* \neq y^\mu. \tag{2.7}$$

An update is then performed on both winners under the winner-takes-all principle (WTA) [1] which is called LVQ 2.1 [4] and is described as:

$$w_j^* \leftarrow w_j^* - \eta(\xi^\mu - w_j^*) \tag{2.8}$$

$$w_k^* \leftarrow w_k^* + \eta(\xi^\mu - w_k^*). \tag{2.9}$$

This results in an update where the of correct winner is moved closer to the training sample and the incorrect winner is moved away from the training sample. After the update the next training sample is picked to do an update step with, this is repeated till all training samples are visited. The term for visiting all training samples is called an epoch, multiple epochs can be executed for learning. An extension of this LVQ version is Generalized Learning Vector Quantization (GLVQ) [11]. Learning in GLVQ is based on minimizing a cost function S with monotonic function Φ where:

$$S = \sum_{i=1}^{N} \Phi(\mu(\xi)). \tag{2.10}$$

This cost function is based on the distances from the training sample to the correct winner d_j from the training sample to the incorrect winner, d_k. From a new distance measure, $\mu(\xi)$ is derived as

$$\mu(\xi) = \frac{d_j - d_k}{d_j + d_k}. \tag{2.11}$$

The quantity $\mu(\xi)$ is a number between 1 and -1 and is negative if the training sample is classified correctly and positive if it is classified incorrectly. Using function gradient descent in order to minimize the cost function S, update steps of the form

$$w_j^* \leftarrow w_j^* + \eta \frac{\partial \Phi}{\partial \mu} \frac{d_k}{(d_j + d_k)^2} (\xi^\mu - w_j^*) \tag{2.12}$$

$$w_k^* \leftarrow w_k^* - \eta \frac{\partial \Phi}{\partial \mu} \frac{d_j}{(d_j + d_k)^2} (\xi^\mu - w_k^*) \tag{2.13}$$

are obtained. These are the GLVQ update steps where the learning rate is adjusted by the cost function. If the error approaches zero, the prototypes do not move anymore during training and the training process is completed.

2.2.2 Relevance learning in LVQ

Not all features in the feature vector are equally important in deciding which class it belongs. For instance if we are classifying vehicles the color is not that relevant for deciding which class it belongs to but the amount of wheels is important to see if it a car or motorcycle. To take this into account, we can use relevance learning where the relevance of each feature is determined. This relevance is used during calculation of the distance measure between the training sample and prototype $d^\Lambda(w, \xi)$. There are multiple forms of relevance learning schemes but the ones used in this research are (Global) Generalised Matrices also known as GMLVQ and Local Generalised matrices (LGMLVQ) which are extensions of GLVQ [13]. The global matrix system uses one matrix Λ to for calculating

$$d^\Lambda(w, \xi) = (\xi - w)^T \Lambda (\xi - w) \tag{2.14}$$

$$\text{where } \Lambda = \Omega^T \Omega \text{ and } \Omega \in \mathbb{R}^{N \times N}. \tag{2.15}$$

Here we calculate the distance measure taking the relevances of the features into account by multiplying by the relevance matrix [13]. Using this distance metric the prototype updates in GMLVQ are described as:

$$\Delta w_j^* = \eta \cdot 2 \cdot \Phi'(\mu(\xi)) \cdot \mu^+(\xi) \cdot \Lambda \cdot (\xi - w_j) \quad (2.16)$$

$$\text{where } \mu^+(\xi) = \frac{2 \cdot d_k}{(d_j + d_k)^2} \quad (2.17)$$

$$\Delta w_k^* = -\eta \cdot 2 \cdot \Phi'(\mu(\xi)) \cdot \mu^-(\xi) \cdot \Lambda \cdot (\xi - w_k) \quad (2.18)$$

$$\text{where } \mu^-(\xi) = \frac{2 \cdot d_j}{(d_j + d_k)^2}. \quad (2.19)$$

The matrix is updated according to the correct (same class) and incorrect (other class) winning prototypes where the update is defined as:

$$\Delta\Omega_{lm} = -\eta \cdot 2 \cdot \Phi'(\mu(\xi)) \cdot (\mu^+(\xi) \cdot ((\xi_m - w_{j,m})[\Omega_k(\xi - w_j)]_l) - \mu^-(\xi) \cdot (\xi_m - w_{k,m})[\Omega_k(\xi - w_k)]_l). \quad (2.20)$$

Each class or prototype can have different feature relevances. To take this into account we can use local relevance matrices (LGMLVQ). In LGMLVQ every prototype or class has its own relevance matrix that is updated with that prototype. This is described as:

$$d^{\Lambda j}(w_j, \xi) = (\xi - w_j)^T \Lambda^j (\xi - w_j). \quad (2.21)$$

The matrix correct update is calculated for the matrix of the corresponding correct winning prototype and the incorrect update is calculated for the matrix of the corresponding incorrect winning prototype, which are defined as:

$$\Delta\Omega_{lm}^j = -\eta_{matrix} \cdot 2 \cdot (\mu(\xi)) \cdot \mu^+(\xi) \cdot ((\xi_m - w_{j,m})[\Omega_j(\xi - w_j)]_l) \quad (2.22)$$

$$\Delta\Omega_{lm}^k = -\eta_{matrix} \cdot 2 \cdot \Phi'(\mu(\xi)) \cdot \mu^-(\xi) \cdot ((\xi_m - w_{k,m})[\Omega_k(\xi - w_k)]_l). \quad (2.23)$$

The updates for each matrix and prototype over all training samples are summed up. This is performed to compute the full gradient of S, which corresponds to batch gradient descent.

3 Methods

3.1 Implementation

We started with the no-nonsense LVQ toolbox [5] and Chebyshev implementation from Friedrich Melchert using chebfun [2] which was also used for the experiments in [7]. The experiments where done in various benchmark functional data sets. The workflow that is maintained is that the data is read from a file, then each feature vector is approximated using the first n Chebyshev polynomials. This results in feature vectors of n new features per feature vector which are now in coefficient space. These vectors are then used for the training and validation process. We ran experiments with the feature vectors both in original and coefficient space. 90% of the feature vectors for training and 10% is used for validation. These feature vectors are randomly divided over the split, 10 splits are used in 10 according runs of training process followed by a validation run. The parameters that can be chosen are the learning rates for the prototypes η_p and for the relevance matrices η_m and the number of epochs (where one epoch is one sweep over the data set). First we benchmarked the data sets with global relevance matrices to set a standard that has the same settings as used in [7]. This is to compare it with the classification score that is obtained using LGMLVQ. To implement this we added a new mode to the toolbox that uses local matrices and, therefore, obtains LGMLVQ by giving every prototype a relevance matrix which is updated with the according prototype. The matrices of both GMLVQ and LGMLVQ are initialized as proportional to the identity matrix of size ($ndim \times ndim$) where $ndim$ is the number of dimensions which is the same as length of a feature vector.
The results of the validation runs are averaged per class over all 10 runs. All datasets are run in functional and original space using GMLVQ and LGMLVQ.

3.2 F-measure

The F-measure (also knows as F1-score or F-score) is a measurement of classification performance for classification systems [6]. It can be interpreted as an average over the recall, which is the fraction of items that is classified over the total number of items in the class (Eq 3.1, 3.4) and precision, which the accuracy of classification (Eq. 3.2, 3.5). For multi-class problems there are two different methods of averaging [6]. The two methods are micro averaging, which tells more about the general performance of classification and macro averaging, which indicates the performances within classes. These are calculated using true-positives, false-positives and false-negatives which are calculated for each class. For class i, tp_i are the true positives, these are the samples that are correctly classified as class i. fp_i are the false positives, these are the samples classified as class i but are not labeled as class i. fn_i are the false negatives, these are the samples that are not classified as i but are labeled as class i.

3.2.1 Micro average

Micro averaging is a score mostly used as an indication over the end result for the whole classification system which is defined as:

$$\text{Recall}_\mu = \frac{\sum_l^{i=1} tp_i}{\sum_l^{i=1}(tp_i + fp_i)} \tag{3.1}$$

$$\text{Precision}_\mu = \frac{\sum_l^{i=1} tp_i}{\sum_l^{i=1}(tp_i + fn_i)} \tag{3.2}$$

$$\text{F-score}_\mu = \frac{2 * \text{Recall}_\mu * \text{Precision}_\mu}{\text{Recall}_\mu + \text{Recision}_\mu}. \tag{3.3}$$

The effect that it has is that the micro score tends to favor bigger classes over smaller ones. The problem with this is that if a big class gets classified very well then a small class could perform very badly but this is not visible in the end score.

3.2.2 Macro average

The second score is the macro score which is a combination of recall and precision which is divided by the number of classes l defined as:

$$\text{Recall}_M = \frac{\sum_l^{i=1} \frac{tp_i}{(tp_i+fp_i)}}{l} \tag{3.4}$$

$$\text{Precision}_M = \frac{\sum_l^{i=1} \frac{tp_i}{(tp_i+fnes_i)}}{l} \tag{3.5}$$

$$\text{F-score}_M = \frac{2 * \text{Recall}_M * \text{Precision}_M}{\text{Recall}_M + \text{Precision}_M}. \tag{3.6}$$

To overcome the bias of the micro score towards bigger classes, macro score treats all classes with the same weight. This helps if a certain class has very poor classification results because this will be visible in the macro score. The F-measure is a clear performance measure for classification using an LVQ system, it indicates the overall performance using the micro score and the class performances with the macro score.

3.3 Data sets

For benchmarking the performance improvement of LGMLVQ we used two datasets. The goal is to measure whether the classification performance improvement is due to using local matrices instead of global matrices. We ran both training processes with equal parameter settings and with optimized parameter settings.

3.3.1 Sugar Dataset

The sugar data set consists of spectral data from 9 different types of sugars sampled with various infrared sensors. The sensor's data contains 160 to 3000 feature vectors with features that consist of light intensities with a wavelength between 300nm and 2000nm [8]. The sensor data that we used for the classification experiments is from the NEO VNIR-1600. This sensor data consists of 1151 feature vectors of 160 dimensions over a wavelength range from 400nm to 1000nm.

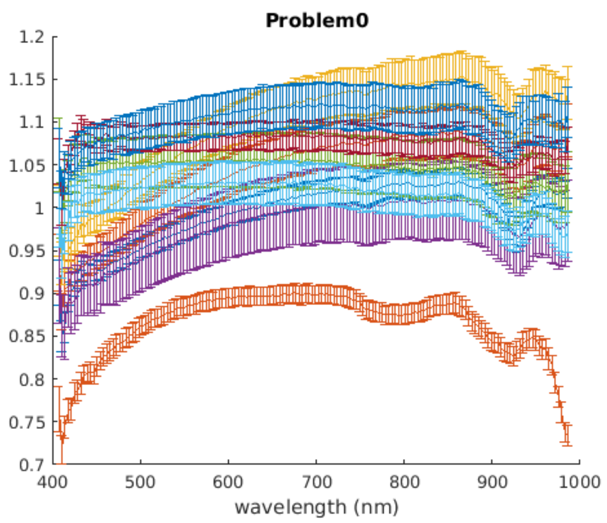


Figure 3.1: Problem 0, mean and variance per class, NEO VNIR-1600

The 9 sugar types are divided into 9 classes that can be grouped into 3 classes of sugar types namely, the Sugar Ester group, -itol group and -ose group. This regrouping is named subproblem

1 which can be seen in Fig. 3.2. We relabeled the data to the subproblem 1 before the approximation with Chebyshev polynomials.

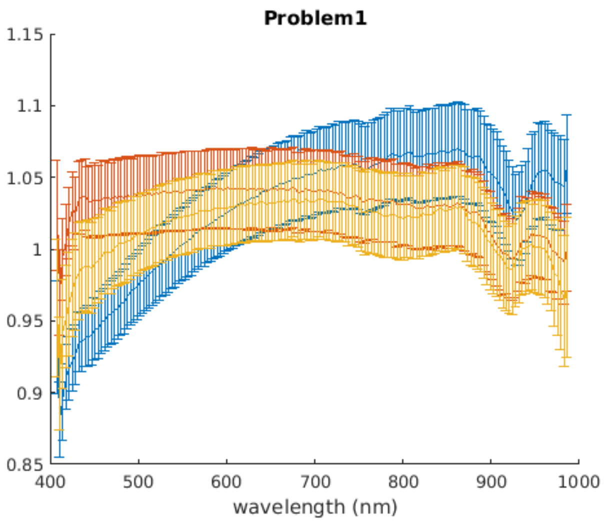


Figure 3.2: Problem 1, mean and variance per class, NEO VNIR-1600

We can see the 3 classes in Figure 3.2, class 1 (blue) has little overlap space with the other 2 classes at the beginning and end. It also is generally an increasing curve. Due to these dissimilarities we expect a good classification performance for class 1. Classes 2 and 3 (yellow and red) have a lot of overlap and share most of the characteristics, this makes it hard to make distinction between the two classes in classification.

3.3.2 UCR Dataset

The UCR Plane data set [10] contains 105 time series feature vectors each containing 160 features. The data set is labeled with 7 classes of equal sizes. The overall shape of the functions in every class are very dissimilar. This makes it a suited data set for classification with polynomial approximation since the coefficients that result from the approximation are also dissimilar among the classes.

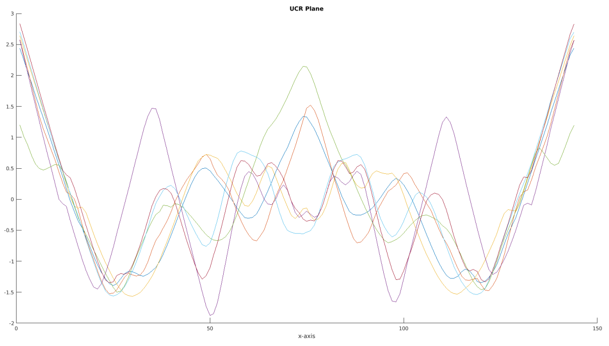


Figure 3.3: UCR Plane mean per class

In Figure 3.3 we see the mean of all feature vectors per class, the means show periodic and symmetric functions.

3.4 Back transformation

After training, the relevance matrices and prototypes are still in the coefficient feature space. Previous studies have shown that back transformation to the original feature space is possible for relevance matrices and prototypes in GMLVQ [9]. To see if this also holds for LGMLVQ we have to transform the relevance matrices back to the original feature space after training.
We use the (linear) transformation F to transform the matrices back where ξ_o is the feature vector in original space and ξ_c is the feature vector in coefficient space. Using the equation:

$$\xi_c' \cdot \Lambda_c \cdot \xi_c \tag{3.7}$$
$$= \xi_o' \cdot F' \cdot \Lambda_c \cdot F \cdot \xi_o \tag{3.8}$$
$$= \xi_o' \cdot [F' \cdot \Lambda_c \cdot F] \cdot \xi_o. \tag{3.9}$$

From this we can take $[F'\Lambda_c F]$ which is the relevance matrix in the original feature space Λ_o.
Because the implementation of the Chebysev polynomials approximation uses subsampling of the data, $[F'\Lambda_c F]$ results in a back transformation of only the points that are subsampled. To obatain a back transformation result that includes all features we have to inverse the subsampling matrix S. This results in a new matrix Q that can be used to transform the relevance matrices back to the original feature space considering:

$$\Lambda_o = Q \cdot F'\Lambda_c F \cdot Q'. \tag{3.10}$$

4 Results

4.1 Image segmentation

First we tested the LGMLVQ implementation by reproducing an experiment from [12] where a UCR image segmentation data set [10] was used with LGMLVQ. In the end we compared the end results of the relevance matrices after training, these results looked very similar when our experiment was trained for a few minutes and the other one for a lot of hours. because of this we can conclude that the implementation of the local matrices was correct.

4.2 Sugar dataset

The results that we obtained from the sugar dataset training runs are the F-measure and the error rates. The parameters we chose for this dataset are $\eta_p = 0.01$ and $\eta_m = 0.02$ and 750 epochs in 10 runs. In table 4.1 we observe the classification F-measures of both systems using the original input data and the data in coefficient space. From this table we can observe that LGMLVQ obtains a higher classification performance in both feature spaces then GMLVQ. The classification performances for GMLVQ in both feature spaces are very similar where in LGMLVQ the performance increases in the coefficient space. Figure 4.1 shows us the resulting error rates of the 3 classes for all systems. Here we can see as well that the error rates are significantly reduced with LGMLVQ in the (functional) coefficient space. GMLVQ is mostly struggling with class 2 and 3. In figure A.9 and A.10 (Appendix A) we can see that the two classes are related in terms of error because where the one goes up the other goes down and vice versa.

Table 4.1: Sugar Dataset: average F-measures over 10 runs for LGMLVQ and GMLVQ in original and functional space

	Original		Functional	
Variant	Local	Global	Local	Global
Micro	0.8133	0.7903	0.8327	0.7991
Macro	0.2816	0.2728	0.2888	0.2774

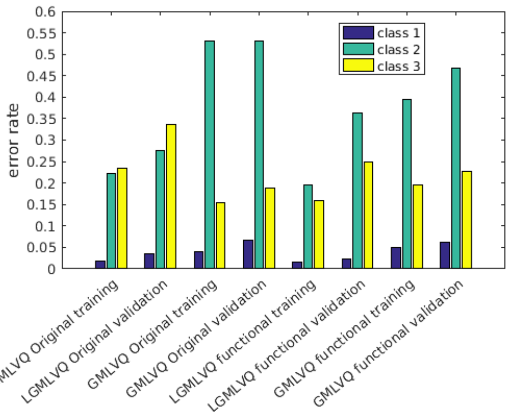


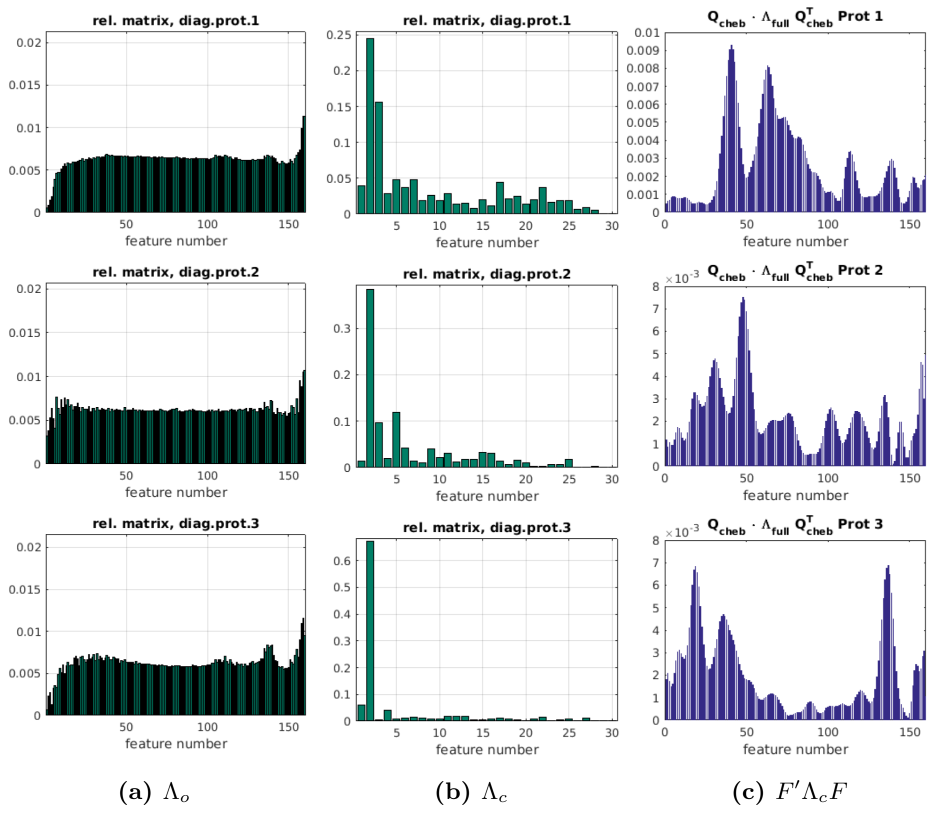
Figure 4.1: Error rates for LGMLVQ/GMLVQ in functional coefficient space and original space

4.2.1 Relevance Matrices

In figure 4.2 the relevance matrices of LGMLVQ are shown. The relevance matrix of the original space data Λ_o is very flat in the middle but shows some interesting relevances at the begin and at the end. In the coefficient space feature number 2 is very dominant for all matrices Λ_c, prototype 1 has some relevance over all features where the matrix of prototype 3 shows only some significant relevance at feature 1,2 and 4. This matrix is very similar to the relevance matrix of the GMLVQ system (Fig. A.11, Appendix A).
In the last set of results we can see the result of the back transformation from the relevance matrices in the coeffient space Λ_c to the matrices in the original space Λ_o. Although the result has a lot of steep peaks as result of the subsampling, the matrices reflect the relevant areas as in Λ_o from the training process with the original input data.

4.3 UCR Plane

The UCR Plane dataset shows that both systems reach a classification F-scores of 100% in most of the 10 splits where $\eta_p = 0.1$ and $\eta_m = 0.2$. The difference here between LGMLVQ and GMLVQ is that LGMLVQ reaches zeros training error earlier then GMLVQ (Fig. A.6).



(a) Λ_o **(b)** Λ_c **(c)** $F'\Lambda_c F$

Figure 4.2: Visualization of the diagonal relevances from the sugar dataset after LGMLVQ training with $\eta_p = 0.02$ $\eta_m = 0.01$, 10 splits, 750 epochs. (a) after training in the original feature space (160 features) (b) after training in the coefficient space with 30 coefficients (c) result of backtransformation from the coefficient space (as in figure b) to the original space

5 Discussion

The goal was to discover what the classification performance improvement is using LGMLVQ using polynomial approximation and what the relevance matrix means if it is translated from the coefficient space back to the original space of the input data. The effect of lower error rates in LGMLVQ for the sugar dataset is related to the fact that the GMLVQ system is not able classify differences between the class 2 and 3, this is caused by lot of feature space overlap between the classes (Fig 3.2). LGMLVQ is more capable of doing this using a relevance matrix for each class. This makes it possible to differentiate the relevances per class, which shows a great effect in test- and training error which is reduced during the training steps (Fig 4.1). The relevances in the LGMLVQ matrices do not show a lot of differences in relevant features per prototype (Fig 4.2b). We can even see that the relevance matrices of LGMLVQ show the same relevances as in the general matrix of the GMLVQ system (Fig A.11). In all matrices feature 2 is particularly dominant. In terms of polynomial approximation feature 2 translates to the coefficient for $k = 1$. If we look at the Chebyshev polynomials that we used as base functions for the polynomial approximation we see that the base function that is used for $k = 1$ is the function $f(x) = x$ which is a linear function. This means that the relevance of the polynomial fitting with a linear line is important for the data set in terms of relevance. This is caused by the fact that the spectra in the sugar data set are not of a parabolic, cubic shape or more complex shape like the Chebyshev polynomials for $3 \leq k \leq 30$.
The cause for the reduction in error rate in LGMLVQ is mostly due to the fact that the relevance matrix of prototype 2 has feature 3 and 5 as relevant features. This makes it possible for the system to classify the differences between class 2 and 3, which results in more error reduction for these classes than in GMLVQ.
The back transformed matrix $F'\Lambda_o F$ shows greater relevance effects than when the original data is used. This effect is the result of making the use of the functional nature of the data. If we compare the matrices with the data in figure 3.2 we can see that the relevances are corresponding with the parts where the average of the class has a deviation from the other classes. These relevances are not clearly shown in the matrices of the original data and, therefore, the classification performances are lower when the original input data is used. The results of the UCR data set show the effect of learning speedup using LGMLVQ from GMLVQ. GMLVQ struggles to get the error of class 2 down to zero (Fig A.2). LGMLVQ reaches zero training error for all classes already at less than 20 steps (Fig. A.6). This is a considerably smaller number of steps than GMLVQ needs. What happens in steps 20-50 is that the LGMLVQ starts to overfit, the test error for class 1 goes up (Fig. A.7). This means that we could have stopped training at 20 steps to get better test scores with LGMLVQ. The total error scores (Fig A.1, A.5) show that the test error for both system are equal, but training error is lower in LGMLVQ.

6 Conclusion

From the UCR Plane and sugar data set we can conclude that LGMLVQ improves the classification and error rates of classification of multi-class functional data. For the UCR data set LGMLVQ finishes training considerably faster than with GMLVQ. From the sugar dataset we can conclude that when local matrices are used in the coefficient space there is a significant classification performance that GMLVQ does not have.
An important factor that has an influence on the classification result are the base functions that are used for the function approximation. The kind of base functions that would obtain the best approximation depends on the data set.

References

[1] Michael Biehl, Barbara Hammer, and Thomas Villmann. Prototype-based models in machine learning. *Wiley Interdisciplinary Reviews. Cognitive Science*, 7(2):92–111, 2016. ISSN 1939-5078. doi: 10.1002/wcs.1378.

[2] Hale N. Trefethen Driscoll, T.A. Chebfun guide. 2014.

[3] T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001. ISBN 3540679219.

[4] Teuvo Kohonen. Improved versions of learning vector quantization. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 545–550. IEEE, 1990.

[5] Biehl M. A no-nonsense beginners tool for gmlvq (version 2.3). 1 2017. Available online, http://www.cs.rug.nl/˜biehl.

[6] Guy Lapalme Marina Sokolova. A systematic analysis of performance measures for classification tasks. *Neural Computation 21, 35323561*, 2009.

[7] Friedrich Melchert, Udo Seiffert, and Michael Biehl. *Polynomial Approximation of Spectral Data in LVQ and Relevance Learning*, pages 25–32. Machine Learning Reports. Bielefeld University, 10 2015.

[8] Friedrich Melchert, Andrea Matros, Michael Biehl, and Udo Seiffert. *The sugar dataset: A multimodal hyperspectral dataset for classification and research*, volume 03, page 15. Univ. of Bielefeld, 7 2016. Machine Learning Reports 03/2016, Bielefeld University.

[9] Friedrich Melchert, Udo Seiffert, and Michael Biehl. Functional representation of prototypes in lvq and relevance learning. In *Advances in Self-Organizing Maps and Learning Vector Quantization*, pages 317–327. Springer, 2016.

[10] Hettich S. Blake C. L. Merz C. J. Newman, D. J. *UCI repository of machine learning databases.* University of California, Department of Information and Computer Science., 1998. Available online at http://archive.ics.uci.edu/ml/.

[11] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. In *Advances in neural information processing systems*, pages 423–429, 1996.

[12] Petra Schneider. Advanced methods for prototype-based classification, 2010. Relation: http://www.rug.nl/ Rights: University of Groningen.

[13] Petra Schneider, Michael Biehl, and Barbara Hammer. Adaptive relevance matrices in learning vector quantization. *Neural computation*, 21(12):3532–3561, 12 2009. ISSN 0899-7667.

[14] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning.* Springer-Verlag, New York, Inc., 2nd edition, 2009. ISBN 978-0-387-84857-0.

A Appendix

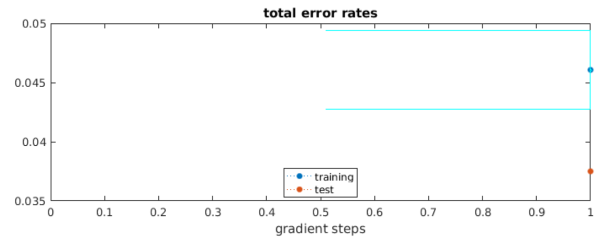


Figure A.1

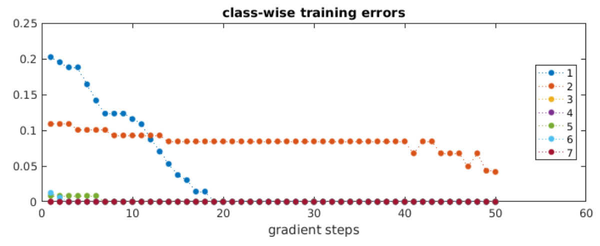


Figure A.2

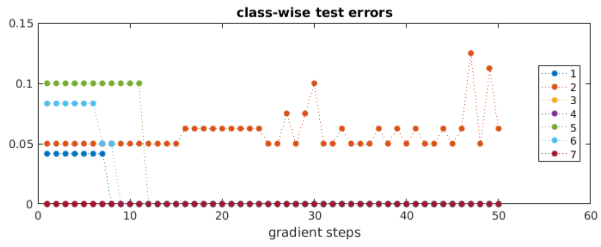


Figure A.3

Figure A.4: Global matrix $\eta_p = 0.02$ $\eta_m = 0.01$, 10 splits

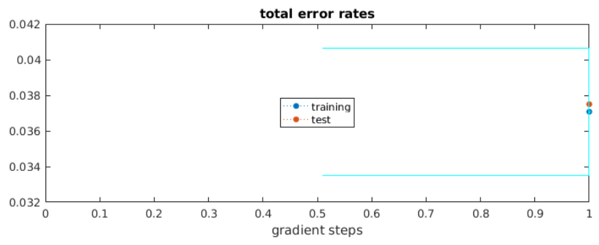


Figure A.5

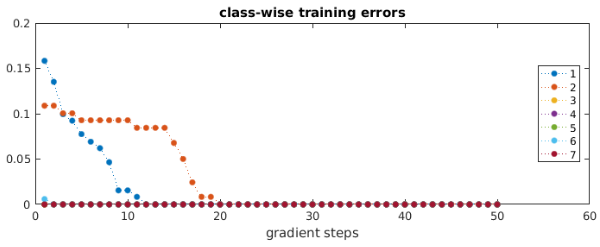


Figure A.6

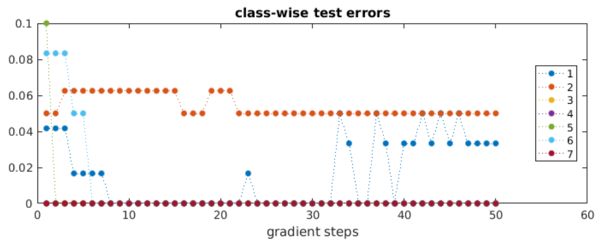


Figure A.7

Figure A.8: Local matrix $\eta_p = 0.02$ $\eta_m = 0.01$, 10 splits

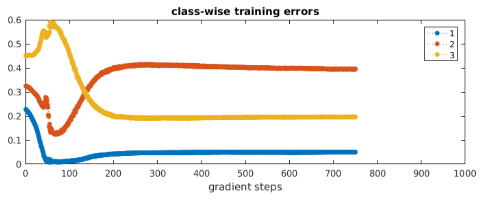


Figure A.9

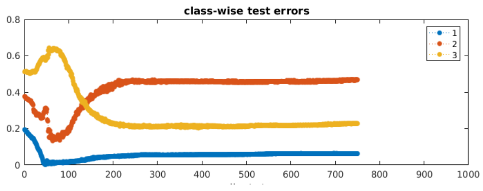


Figure A.10

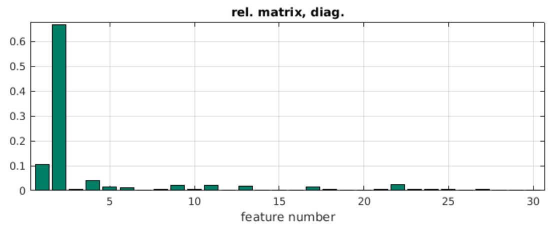


Figure A.11

Figure A.12: Global matrix results $\eta_p = 0.002$ $\eta_m = 0.001$**, 10 splits, 30 polynomials, 750 epochs**