University of Groningen

BSc. thesis Computing Science

Time series classification in complex Fourier space

Author:
Michiel Straat

Supervisors:
Michael Biehl
Friedrich Melchert

Abstract

In this thesis Generalized Matrix Learning Vector Quantization (GMLVQ) is examined on labeled datasets of complex-valued feature vectors which represent time series in Fourier space. Recently, a version of GMLVQ for complex-valued data has been proposed that utilizes the mathematical formalism of Wirtinger calculus to define the derivatives of the GLVQ-cost function with respect to the prototypes and the relevance matrix [4][5]. In this thesis the proposed method is implemented and examined in the specific application of classification of Fourier transformed discrete sampled time series. Experiments show that the method yields classifiers with meaningful prototypes and a matrix of complex relevance values. Besides that, the performance of the functional Fourier representation of the data is compared to the time domain representation in which the functional nature of the data is not taken into account, along the lines of recent research [8][9]. In the experiments benefits are found of the functional approximation of the time series using Fourier basis functions for the classification. These include performance gains and dimensionality reduction.

Contents

1 Introduction

Common approaches when dealing with the classification of complex-valued data and a supervised learning method which is only formulated for real-valued data are taking the real parts only or concatenating the real- and imaginary parts of the data. The first approach ignores the information in the imaginary parts which can also be of importance and therefore might decrease classification performance. The second approach does include the information in the imaginary parts. However, the complex numbers are not considered as one feature and therefore the method may not adjust for correlations of the complex-valued features.

In this thesis the supervised learning method Generalized Matrix Learning Vector Quantization (GMLVQ) is used for the classification of time series data. The similarity measure is a quadratic form in which the matrix contains feature relevance weights giving importance to the features and pairs of features for the classification problem at hand. During the training stage of GMLVQ class prototypes and the relevance matrix (and thus the distance measure) are adapted. The output of the training stage are the prototypes in the same space of the data and the relevance matrix indicating which features and pairs of features were of importance for the classification. These define a classifier, and new data is classified to the class of the prototype which is closest according to the quadratic distance measure. This way the classification resembles a k-nearest neighbors (k-NN) classifier, with the main difference that in k-NN schemes classification is done on the examples in the dataset itself and Learning Vector Quantization (LVQ) schemes have a training phase in advance to find typical representatives of the classes in the dataset. In GMLVQ the updates of the prototypes and relevance matrix is guided by the minimization of a cost function [10][11] consisting of the relative distance between the closest correct prototype and the incorrect prototype. Since distance is always real-valued, the outer derivatives of the cost function are therefore always taken w.r.t. real variables. However, for the inner derivatives, there is a need to describe the derivatives of the distance with respect to the prototypes and the relevance matrix. Chapter 2 gives a quick overview of LVQ, the formulas and cost function that are used. The adaptation of GMLVQ to work on complex-valued feature vectors directly is described in chapter 3 in which in addition some of the derivatives in [4][5] are derived. The implementation of the method is then described as an extension for an existing GMLVQ implementation [1].

Unlike other classification problems with independent features, such as classification of vegetables where we may describe the objects with feature vectors $\xi \in <$ Color, Shape, Structure$>$, in high-dimensional functional data there are properties that we can take advantage of. As was shown in previous research, the naive approach of classification on discretized functional data directly has several disadvantages [9]. These include an unnecessary high dimensionality and therefore a large number of parameters and expensive computation, convergence problems and overfitting effects [9]. To take advantage of the functional nature of the data (such as the natural order and highly correlated neighbour-

ing features), a representation of the spectral data and prototypes in terms of appropriate basis functions (Chebyshev) yielding coefficients was combined with GMLVQ [9], thus the transformation $\xi \in \mathbb{R}^N \rightarrow \mathbf{c} = (c_0, c_1, ...c_n)^T \in \mathbb{R}^{n+1}$ for representation with a polynomial of degree n. Since the data is represented by fewer Chebyshev coefficients, the dimensionality of the classification problem is reduced and the method becomes less prone to noise because of the introduced smoothing of the data [8]. Furthermore it was shown that on some datasets with as few as 20 coefficients classification performance was at least as good as classification in original space, and that the performance gain was not only due to the implicit smoothing of the data, but also due to the appropriate functional representation itself [9].

In this contribution we consider the Fourier representation of time series yielding complex coefficients, where in this case the basis functions are sinusoidal basis functions of different frequencies and the complex coefficients indicate the amount of magnitude and phase present in the time series of the different sinusoids. This will allow us to test the implemented Complex GMLVQ (CGMLVQ) method based on Wirtinger derivatives, and also study the performance of classification of time series (thus prefarably with a periodic nature) in Fourier space along the lines of previous research citemelchert15[9]. The transformation of the data can be described as $\xi \in \mathbb{R}^N \rightarrow \mathbf{X} = (X_0, X_1, ..., X_{n-1})^T \in \mathbb{C}^n$ for the approximation of time series in Fourier space truncated at n frequencies. Some properties of the transform that need to be taken into account are described in chapter 4. Furthermore, note that the intuitiveness of LVQ schemes lies in the fact that the prototypes and relevance matrix are in the same space as the data [11]. Since training here is considered in the complex space the resulting prototypes and matrix are complex-valued. The interpretation of the complex prototypes and matrix and the back transformation in order to interpret the classifier in the original space of the data is the subject at the end of chapter 4.

The experiments in chapter 5 are on different datasets, starting from experiments on datasets purely used to test the Wirtinger CGMLVQ method, and progressing to labeled datasets which appear especially suitable for approximation using Fourier basis functions because of observed periodicity in the time series.

2 Learning Vector Quantization

As discussed GMLVQ is the machine learning method used in this paper. This chapter provides a quick overview of LVQ and then in particular GMLVQ.

LVQ is a supervised learning method that is employed on a labeled data set $\mathbf{X}$ consisting of feature vectors $(\mathbf{x}_i, y) \in \mathbb{R}^N \times C$ that aims to identify typical prototype vectors $(\mathbf{w}_i, y) \in \mathbb{R}^N \times C$ for the different classes C in the data set. Note that the set C is a discrete set of class labels, $\mathbf{C} = (y^1, ..., y^k)$

The prototype vectors are defined as:

$$\mathbf{W} = (\mathbf{w^1}, ..., \mathbf{w}^K), \mathbf{w}^k \in \mathbb{R}^N \tag{2.1}$$

Training phase

In the training phase a distance measure d is used to compute distances between data points and prototypes. The examples $\mathbf{x} \in \mathbf{X}$ are considered on a one by one basis causing an update to one or more prototypes in each such a step. Which prototype(s) are updated is determined by distance measure d. A popular choice of d is the squared Euclidean distance measure.

$$d(\mathbf{w}, \mathbf{x}) = \sum_{i=1}^{N} (x_i - w_i)^2 \tag{2.2}$$

Resulting classifier

After the training phase the classifier is defined by the prototype positions that resulted from the training phase and the distance measure d. An advantage of LVQ is that the resulting prototypes are intuitive, since the prototypes are feature vectors in the same space as the data. They can therefore be interpreted as typical representatives for their class.

An incoming datapoint $\xi \in \mathbb{R}^N$ which is to be classified is given the class label of the prototype which is closest according to d. In certain circumstances in which multiple prototypes per class were chosen, the K nearest prototypes can also be considered and data point ξ is then assigned to the class that has a majority among these K nearest prototypes.

2.1 LVQ1

The most basic version of LVQ updates the prototypes in the learning phase according to a winner-takes-all paradigm and usually the normal squared Euclidean distance is used as a measure of similarity. The prototypes are then trained via the following scheme:

1. Pick input $(\mathbf{x}, y) \in \mathbf{X}$
2. Calculate $\mathbf{w}^* = \text{argmin}_j d(\mathbf{w}^j, \mathbf{x})$
3. Update $\mathbf{w}^*$ according to:

$$\mathbf{w}^* = \begin{cases} \mathbf{w}^* + \alpha \cdot (x - \mathbf{w}^*), & \text{if } c(\mathbf{w}^*) == y \\ \mathbf{w}^* - \alpha \cdot (x - \mathbf{w}^*), & \text{if } c(\mathbf{w}^*) ! = y \end{cases} \tag{2.3}$$

Note that in step three if the winner prototype is of the same class it is attracted to data point $\mathbf{x}$. If the winner prototype is of a different class than the prototype is moved further away from $\mathbf{x}$. The winner-takes-all approach is reflected in the fact that exactly one prototype is updated per example. α denotes the learning rate. The higher α, the more the prototypes move (learn) in each step. α is usually annealed over the training epochs. This in order to move the prototypes by higher amounts in the beginning of the training phase and fine tune the position of prototypes in the last epochs. For example for training epoch t, α can be defined as:

$$\alpha(t) = \frac{a}{b + c \cdot t} \tag{2.4}$$

for constants a, b and c. In this scheme $\alpha(t)$ becomes smaller over the number of epochs and approaches 0 when t approaches ∞.

2.2 GLVQ

In [10] a specific cost function E was introduced that, given a dataset, a configuration of prototypes and distance measure, returns the cost of this configuration. The motivation for this cost function is the observation that the original version of LVQ does not satisfy the convergence condition.

$$E = \sum_{m \in \mathbf{X}} e^m \tag{2.5}$$

In equation (2.5) the cost for a LVQ configuration is defined as the sum of the cost per example. The cost for example $\mathbf{x}^m$ is defined in terms of the relative difference between the closest prototype with the same class label $\mathbf{w}^L$ and the closest prototype with a different class label $\mathbf{w}^K$.

$$e^m = \frac{d_L(\mathbf{x}^m) - d_K(\mathbf{x}^m)}{d_L(\mathbf{x}^m) + d_K(\mathbf{x}^m)} \tag{2.6}$$

The denominator in equation (2.6) ensures that the value of e^m is between -1 and 1. In case of a correct classification of example $\mathbf{x}^m$ the value of e^m is obviously negative ($d_L(\mathbf{x}^m) < d_K(\mathbf{x}^m)$) and in case of an incorrect classification of example $\mathbf{x}^m$ the value is positive ($d_K(\mathbf{x}) < d_L(\mathbf{x})$).

Each training epoch consists of the minimization of the cost function in equation (2.5) by adjusting the prototypes and potential other parameters following a gradient descent. Note that since the cost function is defined in terms of $\mathbf{w}^L$ and $\mathbf{w}^K$ two prototypes are updated in each presentation of an example instead of one. One batchstep of the algorithm can be summarized as:

$$\mathbf{W} = \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}} E \tag{2.7}$$

In which the term $\nabla_{\mathbf{W}} E$, i.e. the gradient of E w.r.t. $\mathbf{W}$, is understood as:

$$\nabla_{\mathbf{W}} E = \sum_{\mu=1}^{P} \nabla_{\mathbf{w}^L/\mathbf{w}^K} e^\mu \tag{2.8}$$

As can be seen in equation (2.8), the update is restricted to w^L and w^K for each example $\mathbf{x}^\mu$. The derivatives of e w.r.t. $\mathbf{w}^L$ and $\mathbf{w}^K$ need to be described. Following the chain rule the two derivatives are described as:

$$\nabla_{\mathbf{w}^L} e^\mu = \frac{\partial e^\mu}{\partial d(\mathbf{x}^\mu, \mathbf{w}^L)} \cdot \frac{\partial d(\mathbf{x}^\mu, \mathbf{w}^L)}{\partial \mathbf{w}^L} = -\frac{4 d_K}{(d_L + d_K)^2}(\mathbf{x} - \mathbf{w}^L) \tag{2.9}$$

$$\nabla_{\mathbf{w}^K} e^\mu = \frac{\partial e^\mu}{\partial d(\mathbf{x}^\mu, \mathbf{w}^K)} \cdot \frac{\partial d(\mathbf{x}^\mu, \mathbf{w}^K)}{\partial \mathbf{w}^K} = \frac{4 d_L}{(d_L + d_K)^2}(\mathbf{x} - \mathbf{w}^K) \tag{2.10}$$

Upon presentation of one example prototypes $\mathbf{w}^L$ and $\mathbf{w}^K$ are adapted in the direction of the negative gradients. This yields the learning rules for the prototypes $\mathbf{w}^L$ and $\mathbf{w}^K$.

$$\mathbf{w}^L = \mathbf{w}^L + \alpha \frac{d_K}{(d_L + d_K)^2}(\mathbf{x} - \mathbf{w}^L) \tag{2.11}$$

$$\mathbf{w}^K = \mathbf{w}^K - \alpha \frac{d_L}{(d_L + d_K)^2}(\mathbf{x} - \mathbf{w}^K) \tag{2.12}$$

Note that the outer derivative is an application of the quotient rule, and the derivative of the Euclidean distance with respect to a prototype is $\frac{\partial d}{\partial \mathbf{w}^*} = -2(\mathbf{x} - \mathbf{w}^*)$. The constant is not of much importance in gradient based learning since a learning rate is used.

2.3 Relevance learning

As mentioned above the squared Euclidean distance is a popular choice for distance measure d. The use of the squared Euclidean distance measure as a measure of similarity has several disadvantages. In the squared Euclidean distance measure every feature is treated as equally important in the calculation of the distance. However, in most classification problems features are of different importance in the classification; Feature 1 might be more discriminative between the classes than feature 2, that is, if we only had the value of feature 1 of an example we can predict the correct class with higher probability than if we only had the value of feature 2. Also if no normalization technique has been used for the features, the different value ranges of the features also impact the amount features influence the distance. Relevance learning is the incorporation of a relevance vector of weights as in [6] or matrix as in [11] in the distance measure to account for these observations and adapt the distance measure accordingly. The matrix of relevance weights accounts in addition for the importance of correlation between pairs of features. Note that the relevance vector or matrix are adapted during training.

Besides improved performance of the classification by adaptation of d one could argue that the intuitiveness of the classifier has also increased: After training the resulting relevance matrix tells meaningful information about the classification problem; On the diagonal we can see how important the different features are for the classification problem, on the off-diagonal elements we can see the importance of the pair-wise correlations.

2.4 GMLVQ

In [11] a full matrix $\mathbf{\Lambda}$ of relevance weights was introduced. The distance measure is a quadratic form.

$$d_{\mathbf{\Omega}}(\mathbf{w}, \mathbf{x}) = (\mathbf{x} - \mathbf{w})^T \Lambda (\mathbf{x} - \mathbf{w}) \tag{2.13}$$

If $\mathbf{x} - \mathbf{w} \neq 0$, then the distance must be greater than zero. To ensure this, Λ must be a positive definite matrix [11]. To this end Λ is written as $\Omega^T \Omega$, since for any non-singular matrix A, $A^T A$ is always a positive definite symmetric matrix. This leads to the equation:

$$d_{\mathbf{\Omega}}(\mathbf{w}, \mathbf{x}) = (\mathbf{x} - \mathbf{w})^T \mathbf{\Omega}^T \mathbf{\Omega} (\mathbf{x} - \mathbf{w}) \tag{2.14}$$

By the distribution property of the transpose operation, $B^T A^T = (AB)^T$ equation 2.14 can be written as:

$$d_{\mathbf{\Omega}}(\mathbf{w}, \mathbf{x}) = \|\mathbf{\Omega}(\mathbf{x} - \mathbf{w})\|^2 \tag{2.15}$$

This representation of the distance measure as in equation 2.15 can in particular be interpreted as the normal squared Euclidean distance after the vectors $\mathbf{x}$ and $\mathbf{w}$ have been transformed under the linear transformation matrix Ω. GMLVQ is also Generalized, meaning it uses the same cost function as GLVQ. Since Λ has been introduced in the distance measure a new derivative with respect to $\mathbf{w}$ needs to be described [11].

$$\frac{\partial d_{\mathbf{\Omega}}}{\partial \mathbf{w}^*} = -2\mathbf{\Omega}^T\mathbf{\Omega}(\mathbf{x} - \mathbf{w}) \tag{2.16}$$

Substituting this new derivative in 2.9 and 2.10 yields the gradients w.r.t. the prototypes and this leads to the new learning rules.

$$\mathbf{w}^L = \mathbf{w}^L + \alpha\frac{d^K}{(d^L + d^K)^2}\mathbf{\Lambda}(\mathbf{x} - \mathbf{w}^L) \tag{2.17}$$

$$\mathbf{w}^K = \mathbf{w}^K - \alpha\frac{d^L}{(d^L + d^K)^2}\mathbf{\Lambda}(\mathbf{x} - \mathbf{w}^K) \tag{2.18}$$

The matrix $\mathbf{\Omega}$ is a parameter of the the distance measure and is also adapted to minimize the cost function.

$$\frac{\partial d_{\mathbf{\Omega}}}{\partial \mathbf{\Omega}} = \mathbf{\Omega}(\mathbf{x} - \mathbf{w})(\mathbf{x} - \mathbf{w})^T \tag{2.19}$$

Then the two gradients of $\mathbf{\Omega}$ w.r.t. distance of $\mathbf{w}^L$ and $\mathbf{w}^K$ are:

$$\nabla_{\Omega_L} e^\mu = \frac{\partial e^\mu}{\partial d^\Omega(\mathbf{x}^\mu, \mathbf{w}^L)} \cdot \frac{\partial d^\Omega(\mathbf{x}^\mu, \mathbf{w}^L)}{\partial \Omega} = \frac{2d_K}{(d_L + d_K)^2}\Omega(\mathbf{x} - \mathbf{w}^L)(\mathbf{x} - \mathbf{w}^L)^T \tag{2.20}$$

$$\nabla_{\Omega_K} e^\mu = \frac{\partial e^\mu}{\partial d^\Omega(\mathbf{x}^\mu, \mathbf{w}^K)} \cdot \frac{\partial d^\Omega(\mathbf{x}^\mu, \mathbf{w}^K)}{\partial \Omega} = -\frac{2d_L}{(d_L + d_K)^2}\Omega(\mathbf{x} - \mathbf{w}^K)(\mathbf{x} - \mathbf{w}^K)^T \tag{2.21}$$

$\mathbf{\Omega}$ is then updated following gradient descent.

$$\mathbf{\Omega} = \mathbf{\Omega} - \beta(\nabla_{\mathbf{\Omega}_L} e^\mu + \nabla_{\mathbf{\Omega}_K} e^\mu) \tag{2.22}$$

As can be seen in equation (2.22) the learning rates of the prototypes and the $\mathbf{\Omega}$ matrix are different, and usually $\beta < \alpha$.

3 Wirtinger Calculus and complex GMLVQ

In this chapter an adaptation of GMLVQ with learning formulas based on derivatives formulated with the mathematical formalism of Wirtinger calculus is discussed [4][5]. With this theoretical derivation of the learning rules of the complex valued prototypes and matrix the adaptation of an existing implementation of GMLVQ in order to work with complex-valued data will be at the center of the attention in the last part of the chapter. This leads to an implementation of GMLVQ suitable for complex-valued datasets, which is used in the experiments in chapter 5.

In the previous section GMLVQ was briefly discussed. Since the data was in the space $\mathbb{R}^N$, the cost for one example (2.6) was a function $f\colon \mathbb{R}^N \to \mathbb{R}$. Similarly, the distance measure d was a function from real valued vectors and matrix to a real scalar. Now we consider data consisting of feature vectors $\mathbf{x} \in \mathbb{C}^N$. Therefore the Euclidean quadratic distance measure needs to be described which takes complex feature vectors and a complex matrix. In order to restate the adaptation rules for the prototypes and the relevance matrix, the mathematical formalism of Wirtinger calculus is used which simplifies differentiation in the complex domain [4][5] by establishing complex differentiation in analogy to real differentiation.

A function $f(z) : \mathbb{C} \to \mathbb{C}$ with $z = x + iy$ can be written as

$$f(z) = f(x + iy) = u(x, y) + iv(x, y) \tag{3.1}$$

As a simple but intuitive example consider the function $f(z) = z^2$:
$f(x + iy) = (x + iy)^2 = x^2 - y^2 + 2xyi$

From the above derivation it is easy to see u and v:

$u(x, y) = x^2 - y^2$
$v(x, y) = 2xy$

Consequently function f can also be regarded as two real-valued functions each of two real variables.

However, in complex GMLVQ the cost function only makes sense if it is real valued. The function that gives the cost for an example $\mathbf{x} \in \mathbb{C}^N$ must therefore be a function of the form $e : \mathbb{C}^N \to \mathbb{R}$. For this reason the discussion focuses on functions of the form as in equation (3.2).

Real-valued functions $f(z) : \mathbb{C} \to \mathbb{R}$ can in analogy to equation (3.1) be regarded as:

$$f(z) = f(x + iy) = u(x, y) \tag{3.2}$$

For this reason function f can also be regarded as a function of two real variables that maps to a real number, i.e. $f = u : \mathbb{R}^2 \to \mathbb{R}$. Hence the derivative of function f w.r.t z can be described in the same way as the derivative of a function of two real variables, i.e. by taking the partial derivative of the function u w.r.t. x and the partial derivative of u w.r.t. y. For the optimization of the function u it is required that these derivatives are both zero:

$\frac{\partial u}{\partial x} = 0$ and $\frac{\partial u}{\partial y} = 0$

The same requirement can be written as: [5]

$$a\frac{\partial u}{\partial x} + ib\frac{\partial u}{\partial y} = 0 \tag{3.3}$$

As pointed out in [5] any nonzero values can be chosen for a and b for equation (3.3) to hold. The choice of $a = \frac{1}{2}$ and $b = \frac{1}{2}$ turns out to be a particularly convenient choice since it leads to the establishment of a calculus in which differentiation w.r.t. complex variables is similar to real differentiation. Choosing $\frac{1}{2}$ as constant yields the aforementioned Wirtinger derivatives:

Wirtinger's derivatives

$$\frac{\partial}{\partial z} = \frac{1}{2}\left(\frac{\partial}{\partial x} - i\frac{\partial}{\partial y}\right) \tag{3.4}$$

$$\frac{\partial}{\partial \bar{z}} = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right) \tag{3.5}$$

Consider function $f : \mathbb{C} \to \mathbb{R}$ of complex variable $z = x + iy$ which is defined as:

$$f(z) = z^* z = x^2 + y^2 \tag{3.6}$$

This function is simply the squared magnitude of z: $f(x+iy) = (x-iy)(x+iy) = x^2+y^2$. Calculating $\frac{\partial}{\partial z}$ by applying the operator 3.4:

$$\frac{\partial}{\partial z}(z^* z) = \frac{1}{2}\left(\frac{\partial}{\partial x}(x^2 + y^2) - i\frac{\partial}{\partial y}(x^2 + y^2)\right) = \frac{1}{2}(2x - i2y) = x - iy = z^* \tag{3.7}$$

Equation (3.7) shows a first example of the convenience of applying Wirtinger calculus; When taking the derivative with respect to z, z^* can be treated as a constant. Likewise using equation (3.5) yields $\frac{\partial}{\partial z^*}(z^* z) = z$. The sum, product and quotient rule can also be shown to be directly applicable in this calculus.

The chain rule is different, but under certain circumstances it is the same as in real calculus [5].

Since cost functions usually depend on many variables, the Wirtinger gradients need to be defined. This is where the true convenience of the method appears, since the separate differentiation of real and imaginary parts would be cumbersome.

$$\frac{\partial f}{\partial \mathbf{z}} = \left(\frac{\partial f}{\partial z_1} \quad ,..., \quad \frac{\partial f}{\partial z_N}\right)^T \tag{3.8}$$

$$\frac{\partial f}{\partial \mathbf{z}^*} = \left(\frac{\partial f}{\partial z_1^*} \quad ,..., \quad \frac{\partial f}{\partial z_N^*}\right)^T \tag{3.9}$$

Consider the function $\|\mathbf{z}\|^2 = \mathbf{z}^H\mathbf{z}$ and:

$$\mathbf{z}^H\mathbf{z} = \begin{pmatrix} z_1^* & ,..., & z_N^* \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ ,..., \\ z_N \end{pmatrix} = z_1^* \cdot z_1 +, ..., + z_N^* \cdot z_N$$

Now using the definition of the operator in (3.8) yields:

$$\frac{\partial}{\partial \mathbf{z}}(\|\mathbf{z}\|^2) = \frac{\partial}{\partial \mathbf{z}}(z_1^* \cdot z_1 +, ..., + z_N^* \cdot z_N) = \begin{pmatrix} z_1^* \\ ,..., \\ z_N^* \end{pmatrix} = \mathbf{z}^* \tag{3.10}$$

And likewise using the definition of the operator in (3.9) yields:

$$\frac{\partial}{\partial \mathbf{z}^*}(\|\mathbf{z}\|^2) = \frac{\partial}{\partial \mathbf{z}^*}(z_1^* \cdot z_1 +, ..., + z_N^* \cdot z_N) = \begin{pmatrix} z_1 \\ ,..., \\ z_N \end{pmatrix} = \mathbf{z} \tag{3.11}$$

In GMLVQ the distance measure is a quadratic form, $\|\mathbf{z}\|_A^2 = \mathbf{z}^H\mathbf{A}\mathbf{z}$, and although we consider complex variables the derivatives are described in an elegant way using Wirtinger gradients:

$$\frac{\partial}{\partial \mathbf{z}}(\|\mathbf{z}\|_\mathbf{A}^2) = \mathbf{A}^T\mathbf{z}^* \tag{3.12}$$

$$\frac{\partial}{\partial \mathbf{z}^*}(\|\mathbf{z}\|_\mathbf{A}^2) = \mathbf{A}\mathbf{z} \tag{3.13}$$

Equation (3.13) follows directly from equation (3.9), as an example:
$\mathbf{z}^H\mathbf{A}\mathbf{z} = z_1^*(A_{11}z_1 +, ..., + A_{1N}z_N) +, ..., + z_N^*(A_{N1}z_1 +, ..., + A_{NN}z_N)$

Applying the definition of the gradient (3.9) yields:

$$\frac{\partial}{\partial \mathbf{z}^*}(z_1^*(A_{11}z_1+,...,+A_{1N}z_N)+,...,+z_N^*(A_{N1}z_1+,...,+A_{NN}z_N))$$
$$= \begin{pmatrix} A_{11}z_1+,...,+A_{1N}z_N \\ ,..., \\ A_{N1}z_1+,...,+A_{NN}z_N \end{pmatrix} = \mathbf{Az}$$

Equation (3.13) is used in the next section to describe the derivative of the distance measure w.r.t the prototypes.

Recall that in GMLVQ the $\mathbf{\Omega}$ matrix (containing relevance weights) of the quadratic distance measure is also updated according to the direction of the negative gradient. Hence it is also necessary to describe this derivative in order to derive the corresponding learning rule.

Derivation:

$$\frac{\partial}{\partial \mathbf{\Omega}^*}(\mathbf{z}^H\mathbf{\Omega}^H\mathbf{\Omega z}) = \frac{\partial}{\partial \mathbf{\Omega}^*}\left(\begin{pmatrix} z_1^*\Omega_{11}^* + z_2^*\Omega_{12}^* & z_1^*\Omega_{21}^* + z_2^*\Omega_{22}^* \end{pmatrix}\begin{pmatrix} \Omega_{11}z_1 + \Omega_{12}z_2 \\ \Omega_{21}z_1 + \Omega_{22}z_2 \end{pmatrix}\right)$$
$$= \frac{\partial}{\partial \mathbf{\Omega}^*}\left((\Omega_{11}z_1 + \Omega_{12}z_2)z_1^*\Omega_{11}^* + (\Omega_{11}z_1 + \Omega_{12}z_2)z_2^*\Omega_{12}^* + (\Omega_{21}z_1 + \Omega_{22}z_2)z_1^*\Omega_{21}^* + (\Omega_{21}z_1 + \Omega_{22}z_2)z_2^*\Omega_{22}^*\right)$$
$$= \begin{pmatrix} (\Omega_{11}z_1 + \Omega_{12}z_2)z_1^* & (\Omega_{11}z_1 + \Omega_{12}z_2)z_2^* \\ (\Omega_{21}z_1 + \Omega_{22}z_2)z_1^* & (\Omega_{21}z_1 + \Omega_{22}z_2)z_2^* \end{pmatrix} = \mathbf{\Omega z}\mathbf{z}^H$$

$$\frac{\partial}{\partial \mathbf{\Omega}^*}(\mathbf{z}^H\mathbf{\Omega}^H\mathbf{\Omega z}) = \mathbf{\Omega z}\mathbf{z}^H \tag{3.14}$$

Equation (3.14) is used to describe the derivative of the distance measure w.r.t. the $\mathbf{\Omega}$ matrix.

3.1 Wirtinger applied to complex-valued GMLVQ learning rules

The distance measure in the complex space is defined in analogy to the distance measure in the real space and maps to real positive numbers when $\mathbf{x} \neq \mathbf{w}$.

$$d^{\mathbf{\Omega}}(\mathbf{x}, \mathbf{w}) = (\mathbf{x} - \mathbf{w})^H\mathbf{\Omega}^H\mathbf{\Omega}(\mathbf{x} - \mathbf{w}) \tag{3.15}$$

The learning rules in GMLVQ for real-valued data were based on the gradients of the cost function w.r.t. $\mathbf{w}^L$, $\mathbf{w}^K$ and $\mathbf{\Omega}$. Recall the definition of the cost function for an example $\mathbf{x}^\mu$ in equation (2.6). The cost function itself consists of real distance values and therefore only the derivative of the distance measure is a derivative w.r.t. complex variables, i.e. a Wirtinger derivative.

Following the just derived equation (3.13) the derivative with respect to $\mathbf{w}$ yields:

$$\frac{\partial d_{\Omega}^2}{\partial \mathbf{w}^*} = -\Omega^H \Omega(\mathbf{x} - \mathbf{w}) \tag{3.16}$$

And following equation (3.14) the derivative w.r.t. Ω^* yields.

$$\frac{\partial d_{\omega}^2}{\partial \Omega^*} = \Omega(\mathbf{x} - \mathbf{w})(\mathbf{x} - \mathbf{w})^H \tag{3.17}$$

Substituting the derivative in equation (3.16) in equations (2.9) and (2.10) gives the gradients and this yields the learning rules of $\mathbf{w}^L$ and $\mathbf{w}^K$ in the complex space.

$$\mathbf{w}^L = \mathbf{w}^L + \alpha \frac{d^K}{(d^L + d^K)^2} \Omega^H \Omega(\mathbf{x} - \mathbf{w}^L) \tag{3.18}$$

$$\mathbf{w}^K = \mathbf{w}^K - \alpha \frac{d^L}{(d^L + d^K)^2} \Omega^H \Omega(\mathbf{x} - \mathbf{w}^K) \tag{3.19}$$

Substituting the derivative in equation (3.17) in equation (2.20) and (2.21) gives the gradients and this yields the learning rules of $\mathbf{\Omega}$

$$\Omega = \Omega + \frac{\beta}{(d^L + d^K)^2} \left(d^L \Omega(\mathbf{x} - \mathbf{w}^K)(\mathbf{x} - \mathbf{w}^K)^H - d^K \Omega(\mathbf{x} - \mathbf{w}^L)(\mathbf{x} - \mathbf{w}^L)^H \right) \tag{3.20}$$

Note that $\mathbf{\Omega}$ is adapted in such a way that $d_L(\mathbf{x}^\mu)$ becomes smaller and $d_K(\mathbf{x}^\mu)$ becomes bigger. A comparison of these learning rules for complex GMLVQ with the learning rules for real GMLVQ reveals that the updates are formally very similar. This follows directly from the fact that the derivatives of the distance measure w.r.t. the complex-valued variables ((3.16), (3.17)) are similar to the derivatives of the distance measure in the real domain, thanks to Wirtinger's calculus. Note that constant differences (like observed in the derivatives of d w.r.t. $\mathbf{w}$ in the real domain vs. the complex domain) are not of much importance when using learning rates. The transpose operation on the matrix Ω is the Hermitian transpose, however this is the standard formulation of the transpose operation in the complex domain and therefore is the default behavior of the transpose operation in software packages such as Matlab.

3.2 Extending an existing GMLVQ implementation

For an implementation of complex GMLVQ it is obvious that complex data types need to be available in order to represent the complex feature vectors, prototypes and matrix. The transpose operation on vectors and matrices needs to be defined as the Hermitian transpose.

In this paper an existing implementation of GMLVQ is used [1]. This implementation is in Matlab. Matlab has a complex data type already built in. The transpose operation on a matrix `A` (in Matlab: `A'`) is defined as the Hermitian transpose. When we want to transpose a vector (or matrix) without conjugating then we should use `A.'` to indicate this. Besides changes made in upcoming chapters, the learning rules as in equations (3.18), (3.19), (3.20) need to be implemented in the batchstep of the algorithm when examples are presented one by one in order to adapt the prototypes and matrix. In analogy to the similarity between the learning rules in real GMLVQ and complex GMLVQ, in Matlab the expression in code of the complex learning rules turns out to be very similar to the real learning rules.

Furthermore, after an adaptation of $\mathbf{\Omega}$ a normalization has to be applied such that $\sum \Lambda_{ii} = 1$. Recall that $\Lambda = \mathbf{\Omega}^H \mathbf{\Omega}$ and therefore Λ_{ii} is the sum of the squared absolute values of column i of $\mathbf{\Omega}$. Adding up the sum of the squared absolute values of every column of $\mathbf{\Omega}$ yields the normalization factor. $\mathbf{\Omega}$ is then divided by the square root of the normalization factor to satisfy the required $\sum \Lambda_{ii} = 1$. Hence after each batchstep normalization of $\mathbf{\Omega}$ is done as follows:

$$\mathbf{\Omega} = \frac{\mathbf{\Omega}}{\sqrt{\sum_j \sum_i \|\Omega_{ij}\|^2}} \tag{3.21}$$

4 Classification in coefficient space, Forward and backward transform

In the previous chapter an adaptation of the GMLVQ learning formulas was discussed and implemented such that complex-valued data can be handled. Complex valued data appears in various domains: Image analysis, Fourier- and Laplace transformed data and so forth. The specific application for testing the method is time-series data represented by feature vectors $\mathbf{x} \in \mathbb{R}^N$ to which the discrete Fourier transform is applied yielding complex coefficients $\mathbf{X} \in \mathbb{C}^N$.

Unlike applications of GMLVQ to datasets in which examples are described by concatenating features of a different nature (color, shape, material), the features in time series data can be seen as discrete samples of an underlying function, which is unknown. This allows for specific techniques to be used to take advantage of the functional nature of the data. Recent research has found benefits of learning in coefficient space [9]. GMLVQ was applied to the functional representation (coefficients representing contribution of Chebyshev basis functions) of spectral data and it was shown that an accurate classifier could be trained using those truncated Chebyshev series. Here, as a functional representation, we consider the Fourier approximation in which the coefficients in Fourier space indicate the contribution, which are magnitude and phase, of the sinusoidal Fourier basis functions. For this reason, although the Fourier transform can be applied to many kinds of functions, it would be particularly interesting to apply the technique to datasets of time series with a periodic nature.

The rest of this chapter briefly describes the discrete Fourier transform (DFT) and a resulting theorem about symmetry (4.1.1) which is important to take into account. Since training in coefficient space yields a classifier described by the prototypes and $\mathbf{\Omega}$ matrix in the same space, for intuitiveness a back transformation is described to the original real space, the time domain, in order to interpret the prototyeps and $\mathbf{\Omega}$ matrix in the original space.

4.1 Forward transformation: DFT

Given a dataset of feature vectors $\mathbf{x} \in \mathbb{R}^N$ representing time series of N samples, the DFT is applied to each of the time series yielding frequency spectra $\mathbf{X} \in \mathbb{C}^N$ [12].

$$X(\omega_k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, k = 0, 1, 2...N-1 \tag{4.1}$$

in which x is the sampled time series signal and $e^{-j2\pi kn/N}$ is the sampled complex sinusoid at frequency ω_k. The summation in equation (4.1) is understood as the dot product of the signal x and the complex sinusoid at frequency ω_k, which computes the coefficient of projection of the signal x onto the complex sinusoid at frequency ω_k [12]. Therefore the DFT can also be expressed as a multiplication of a signal with a DFT matrix, in which the kth row of the DFT matrix is the sampled complex sinusoid at frequency ω_k.

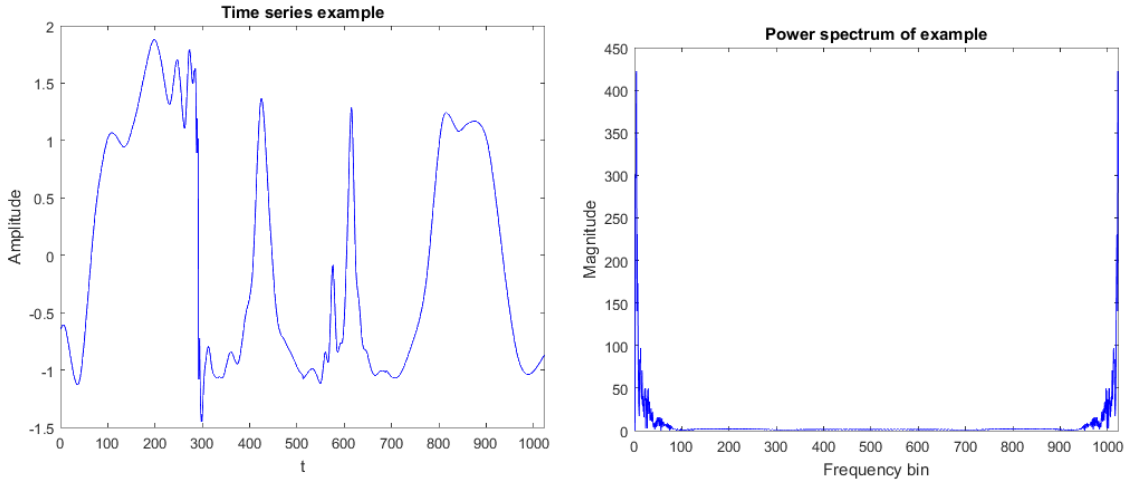
$$\begin{pmatrix} X(\omega_0) \\ X(\omega_1) \\ \dots \\ X(\omega_{N-1}) \end{pmatrix} = \begin{pmatrix} \overline{s_0(0)} & \overline{s_0(1)} & \dots & \overline{s_0(N-1)} \\ \overline{s_1(0)} & \overline{s_1(1)} & \dots & \overline{s_1(N-1)} \\ \dots & \dots & \dots & \dots \\ \overline{s_{N-1}(0)} & \overline{s_{N-1}(1)} & \dots & \overline{s_{N-1}(N-1)} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \dots \\ x(N-1) \end{pmatrix} = \mathbf{Fx} \tag{4.2}$$

Each $X(\omega_k)$ obtained from a signal x is then a complex coefficient in which $\|X(\omega_k)\| = \sqrt{\Re(X(\omega_k))^2 + \Im(X(\omega_k))^2}$ represents the amplitude and $\phi(X(\omega_k)) = \tan^-1(\frac{\Im(X(\omega_k))}{\Re(X(\omega_k))})$ represents the phase of the complex sinusoid at frequency ω_k.

4.1.1 Symmetry for real-valued series

Each time series signal is transformed from the time domain representation $x \in \mathbb{R}^N$ to the frequency spectrum $X \in \mathbb{C}^N$ using formula (4.1). There is an important observation to take into account when considering frequency spectra obtained from a DFT on real-valued signals.

Theorem 4.1.1 *The frequency spectrum of a real-valued discrete time series obtained by a DFT is conjugate symmetric such that* $X(-\omega_k) = X(\omega_k)^*$.



(a) Time series example. **(b)** Corresponding symmetric power spectrum.

Figure 4.1: A plot of a time series (left) and the corresponding power spectrum (right). Note the symmetry in the spectrum.

Theorem (4.1.1) is important to take into account when considering the Fourier representation of real-valued time series: One half of the frequency spectra is redundant since it does not contain extra information and it can be easily reconstructed from the other half. For this reason the time series $x \in \mathbb{R}^N$ can be represented by complex coefficient vectors $X \in \mathbb{C}^{N/2+1}$ without losing any information.

4.1.2 Representations of vectors in Fourier space

As mentioned in chapter 1 complex-valued vectors could also be represented by concatenating the real parts and the imaginary parts to make them suitable for classification algorithms which are only defined for real-valued data. Therefore the frequency spectra as discussed above $X \in \mathbb{C}^{N/2+1}$ could be represented as follows:

$$\begin{bmatrix} \Re(X) \\ \Im(X) \end{bmatrix} \in \mathbb{R}^{N+2} = \begin{bmatrix} \Re(x_1) \\ \Re(x_2) \\ \dots \\ \Re(x_{N/2+1}) \\ \Im(x_1) \\ \Im(x_2) \\ \dots \\ \Im(x_{N/2+1}) \end{bmatrix} \in \mathbb{R}^{N+2} \tag{4.3}$$

In the experiments, chapter 5, the classification performance on this representation is obtained for comparison with the Wirtinger method.

4.2 Classifier in complex space and back transformation

One of the advantages of LVQ learning schemes is that the resulting classifier of LVQ learning is intuitive since the prototypes are in the same space as the data and in the case of GMLVQ the Λ matrix gives the relevance of the features for the classification which provides meaningful information about the classification problem besides the adaptation of the distance measure. If training is done on Fourier coefficients $\mathbb{C}^N$, the resulting prototypes are therefore $\mathbf{w}_i \in \mathbb{C}^N$ and the matrix $\Lambda \in \mathbb{C}^{N \times N}$. This gives an intuitive interpretation in the Fourier space: The prototypes represent the class-typical contributions of the frequency components and the relevance matrix indicates which frequency components are most discriminative. Since the data was originally in the time domain, it is necessary to transform the prototypes $\mathbf{w}_i \in \mathbb{C}^N$ and matrix $\Lambda \in \mathbb{C}^{N \times N}$ from the Fourier domain to the time domain for an intuitive interpretation in the time domain. The inverse Fourier transform takes a frequency spectrum and transforms it to the corresponding time domain signal.

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N}, n = 0, 1, 2, ..., N-1 \tag{4.4}$$

The inverse DFT can be understood as the projections of a signal x onto the N different complex sinusoids [12]. We see in (4.4) that $x(n)$ is defined as the average of the dot product between the frequency spectrum X and the nth sample of the complex sinusoids $e^{j2\pi nk/N}$. An intuitive understanding is: In order to get the nth time domain sample of x, we look at the nth sample of each complex sinusoid k, scale them by their contribution $X(k)$, sum these up and take the average.

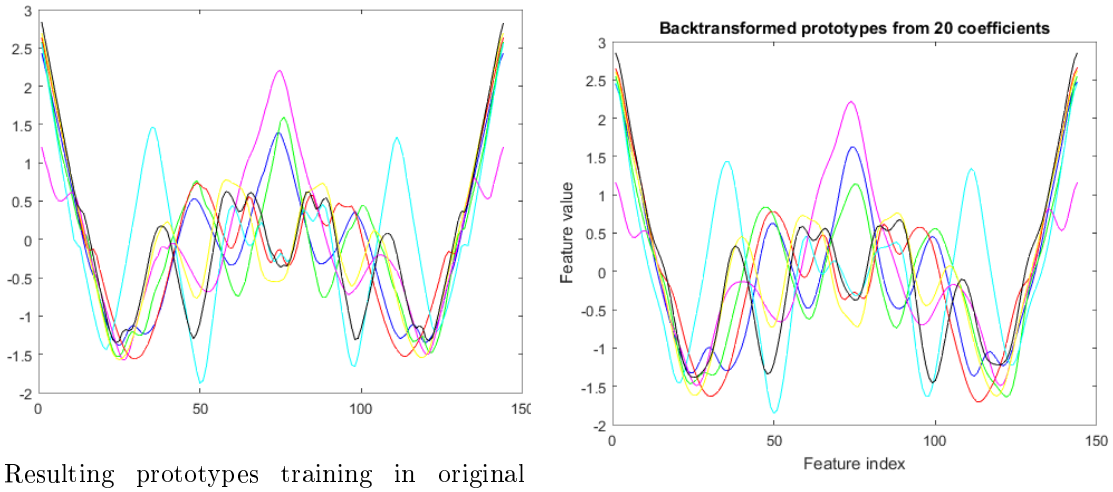
4.2.1 Truncation and reconstruction

As discussed the Fourier transform of a time series $\mathbf{x} \in \mathbb{R}^N$ yields a vector $\mathbf{X} \in \mathbb{C}^N$, this is because N Fourier sinusoids are compared to the signal $\mathbf{x}$. This follows directly from equation (4.1) in which k, the frequency number, ranges from $[0...N-1]$. Truncation at n frequencies refers to k ranging from $[0...n-1]$ where obviously $n < N$, yielding spectra $\mathbf{X} \in \mathbb{C}^n$ in which only the first n complex sinusoids are used.

As a side note: The inverse DFT as in equation (4.4) assumes a conjugate symmetric vector $\mathbf{X}$ and the implementation in Matlab requires $\mathbf{X}$ to be of length N in order to construct a real signal $\mathbf{x}$ of length N. Therefore zeroes need to be appended to a prototype $\mathbf{w} \in \mathbb{C}^n$ representing a spectrum truncated at n core lowest frequencies, and the conjugate symmetric part needs to be concatenated to yield $\mathbf{w} \in \mathbb{C}^N$.

4.2.2 Interpretation of the complex prototypes in complex and original space

After having trained the classifier in the Fourier space, each prototype $(W, c) \in \mathbb{C}^n \times C$ represents the typical frequency spectrum for its class c. Therefore the prototype (W, c) represents the class-typical contributions (magnitude and phase) of frequency components ω_k for class c. As already stated, the negative frequency components can be easily reconstructed and appended to the vectors yielding prototypes $(W, c) \in \mathbb{C}^N \times C$. If an inverse Fourier transform is applied to $(W, c) \in \mathbb{C}^N \times C$ as in (4.4), the time domain signal $(w, c) \in \mathbb{R}^N \times C$ is constructed using the typical complex sinusoids for the class y. This yields the class-typical prototypes in the time domain and can now be interpreted and compared to the time series in the dataset. In figure 4.2b is an example of this: On the left prototypes are shown that resulted from training on a dataset in the original space and on the right prototypes are shown that resulted from training in 20 coefficient Fourier space and then back transformed to original space.



(a) Resulting prototypes training in original space.

(b) Training on 20 coefficients, backtransformed.

Figure 4.2: Resulting prototypes from learning on a dataset consisting of 7 classes, one prototype per class. On the left are the resulting prototypes from learning in the original space of the time series. On the right are the resulting prototypes from learning in 20-coefficient Fourier space and then transformed back to the original space. The prototypes on the right are smoother than the ones on the left, for example observe the red prototype around features 50 and 100.

4.2.3 Interpretation of complex Λ_c in complex and original space

The matrix Λ_c is always a Hermitian positive definite matrix, by the fact that Λ is computed as $\Lambda_c = \Omega^H \Omega$. An Hermitian matrix has real diagonal values ($\Im(\Lambda_{ii}) = 0$) and always has real eigenvalues. The real values on the diagonal of Λ_c have the same interpretation as Λ obtained from training on real valued data: The values represent the discriminative power of the features for the classification problem, and therefore it is necessary that the most discriminative features have the highest impact on the distance. In the specific context of Fourier coefficients, the features are the different frequency components. In this sense if Λ_{ii} has a high value, then the frequency component ω_i was of high importance in the classification. That is for example in a two class problem where examples from one class may have a high presence of the ω_i frequency component while the ω_i frequency component was less present in examples from the other class. Likewise, if a frequency component ω_j has not much discriminative power, this can be because the component is equally present in both classes or the classes show the same range of presence of the frequency component among the examples, then a low relevance value for that component is the result.

In Λ trained on real valued data the off-diagonal elements give weights to the importance of correlations between pairs of features/variables. The off-diagonal elements in complex Λ are complex-valued. $\Lambda^{abs} = \|\Lambda\|$ yields a real-valued symmetric matrix in which $\Lambda^{abs}_{ij} = \Lambda^{abs}_{ji}, i \neq j$ gives an overall weight to the importance of the correlation between features i and j. This real representation can be used to graph the matrix visually, for example as in the GMLVQ toolbox [1]. The complex value of the matrix $\Lambda_{ij}, i \neq j$ itself specifies the correlation of the real and imaginary parts separately between features i and j.

Using an artificial dataset with examples $\mathbf{x} \in \mathbb{C}^2$ some of the above properties are illustrated, the example script used for this is in `testCorrelationsComplex`. The idea is to let the two classes be discriminated only by the difference between feature 1 and feature 2. The following cases are examined:

1. Importance correlation feature 1 and feature 2 based on absolute value.
2. Importance correlation feature 1 and feature 2 based on:
   (A) Real part only
   (B) Imaginary part only

Listing 4.1: Resulting $\mathbf{\Lambda}_C$ test 1

```
1  0.5286 + 0.0000i 0.1834 + 0.4643i
2  0.1834 - 0.4643i 0.4714 + 0.0000i
```

Listing 4.2: Resulting $\mathbf{\Lambda}_C$ test 2A

```
1   0.6441 + 0.0000i -0.4636 - 0.1194i
2  -0.4636 + 0.1194i 0.3559 + 0.0000i
```

Listing 4.3: Resulting $\mathbf{\Lambda}_C$ test 2B

```
1  0.4719 + 0.0000i 0.0393 + 0.4977i
2  0.0393 - 0.4977i 0.5281 + 0.0000i
```

From the above results clearly the matrix is able to detect case 1 by giving a high absolute weight to the correlation. The cases in which only the real- or imaginary parts are correlated are also reflected in a corresponding weight for real/imaginary value for cases 2A and 2B.

If there is a correlation between two features only in the respective real- or imaginary parts the concatenation method is suitable for this since it splits real- and imaginary parts into separate features.

4.2.3.1 Back transformation Λ_c to original space

For the interpretation of the relevance matrix Λ_c in the original space a back transformation needs to be formulated. The distance measure in the original space is defined as:

$$d^{\mathbf{\Lambda}}(\mathbf{x}, \mathbf{w}) = (\mathbf{x} - \mathbf{w})^T \mathbf{\Lambda} (\mathbf{x} - \mathbf{w}) \tag{4.5}$$

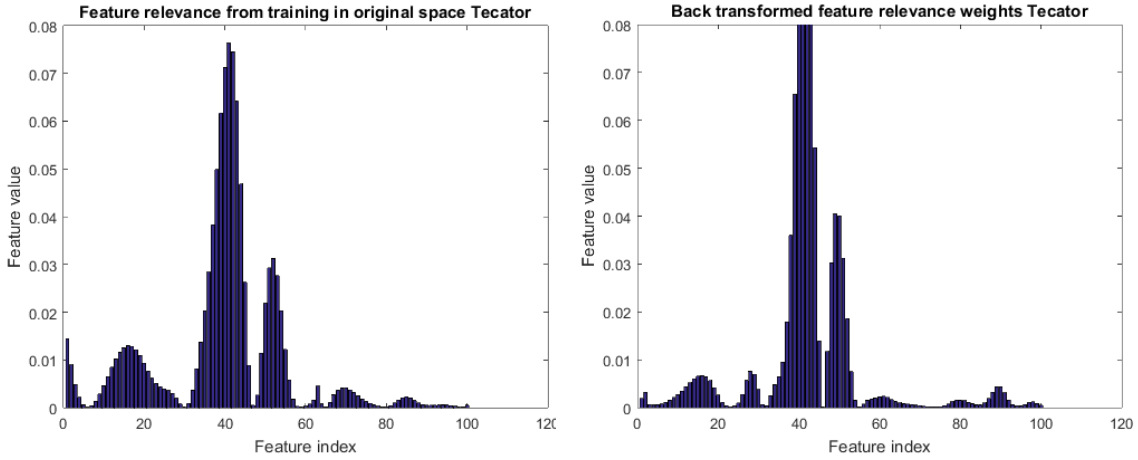
Recall that the distance measure in Fourier space is defined as:

$$d^{\mathbf{\Lambda}}(\mathbf{x}, \mathbf{w}) = (\mathbf{x}_C - \mathbf{w}_C)^H \mathbf{\Lambda}_C (\mathbf{x}_C - \mathbf{w}_C) \tag{4.6}$$

Since the vectors in the Fourier space are obtained from an application of the DFT matrix $\mathbf{F}$ in equation (4.1), equation (4.6) can be understood as: [7]

$$d^{\mathbf{\Lambda}}(\mathbf{x}_C, \mathbf{w}_C) = (\mathbf{x}_C - \mathbf{w}_C)^H \mathbf{F}^H \mathbf{\Lambda}_C \mathbf{F}(\mathbf{x} - \mathbf{w}) \tag{4.7}$$

And therefore this leads to $\mathbf{\Lambda} = \mathbf{F}^H \mathbf{\Lambda}_C \mathbf{F}$.



(a) Feature relevance from training in original space.

(b) Back transformed relevance from training in Fourier space.

Figure 4.3: Comparison between feature relevance as trained in original space and as trained in Fourier space and then back transformed to original space.

In figure 4.3 the feature relevance profile as resulted from training in original space and as resulted from training in Fourier space and applying a back transformation on the resulting matrix Λ_C are plotted. The features that are indicated as most relevant (i.e. the highest peaks) are very similar between the two figures.

5 Experiments

This chapter consists of experiments on several time series datasets primarily taken from the UCR repository [2]. The central theme in this chapter is to test the CGMLVQ method and to study the classification performance in the Fourier space.

5.1 Test strategy

For each data set the performance in the original domain is recorded, and this classifier's performance on the validation sets serves as a reference. This reference performance is compared to the resulting classifiers trained in the complex Fourier coefficient space truncated at n frequencies, which is obtained as described in chapter 4.

1. Performance of GMLVQ in the original feature space which serves as a reference. Since the data consists of vectors $\mathbf{x} \in \mathbb{R}^N$ the resulting prototypes are $\mathbf{w} \in \mathbb{R}^N$ and the matrix $\Lambda \in \mathbb{R}^{N \times N}$.

2. Training classifiers with the Wirtinger derived adaptation of GMLVQ following chapter 3 in the Fourier space of the time series, first on all the coefficients and then truncated at n coefficients. Vectors $\mathbf{x} \in \mathbb{R}^N$ are represented by vectors $\mathbf{X} \in \mathbb{C}^n$. The resulting prototypes are $\mathbf{W}_i \in \mathbb{C}^n$ and matrix $\Lambda \in \mathbb{C}^{n \times n}$.

   As discussed in chapter 4 complex coefficients can also be represented by the concatenation of the real- and the imaginary parts. This means that training on frequency spectra is done in the real space and it is interesting to compare the resulting performance to the performance obtained from training in the complex valued space. The data are then $\mathbf{x} \in \mathbb{R}^{2n}$ for n coefficients and the resulting prototypes are $w_i \in \mathbb{R}^{2n}$ and matrix $\Lambda \in \mathbb{R}^{(2n) \times (2n)}$.

3. The third strategy aims to examine the effect of the smoothing on the classification performance. To this end the data vectors $\mathbf{x} \in \mathbb{R}^N$ are transformed to Fourier space with truncation at n frequencies just like in the second strategy, but here in contrast the data is transformed back to the original domain with the iDFT as described in chapter 4 yielding smoothed vectors $\mathbf{x}_s \in \mathbb{R}^N$ and training and validation is then applied on this data. This will allow us to understand if a possible performance gain that might be observed in strategy 2 can only be explained by the fact that a smoothing was applied to the data, or that in addition the functional representation of the data is beneficial to classification performance.

For each of the tests, either cross validation is used (as in script `do_cross_benchmark`) and/or the specified splits in the UCR repository [2]. (`do_benchmark`).

5.1.1 Performance measurement of the classifiers

Two-class problems:
In a two-class problem we usually call the two classes "positive" and "negative". For each feature vector $\mathbf{x}$ in the validation set, a classifier's output is the prediction that $\mathbf{x}$ belongs to the positive- or the negative class. This predicted class is either the true class to which $\mathbf{x}$ belongs or the classifier has misclassified $\mathbf{x}$. A confusion matrix shows the classification results on the validation set by plotting true class membership against the predicted classes [3]. Obviously, since the true- and predicted class are equal on the diagonal, the diagonal elements represent the correct classifications and the off-diagonal elements represent misclassifications.

When we have a confusion matrix we can calculate the true- and false positive rates, tpr and fpr respectively [3]. When we have tpr and fpr, the point (fpr, trp) is said to be in ROC space. An example is shown in figure 5.1.

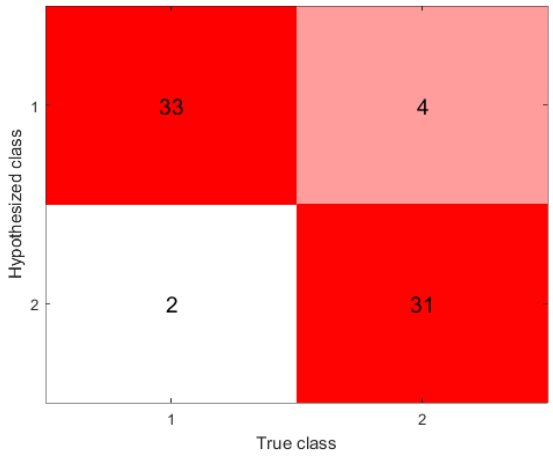


Figure 5.1: An example confusion matrix for a two class problem. TP: 33/35 = 0.94 FP: 4/35 = 0.11, yields the point (0.11, 0.94) in ROC space.

Note that the above is true for a discrete classifier, i.e. one that outputs only a class label [3]. Scoring classifiers output a probability which represents the degree to which a feature vector is a member of a class. A scoring classifier can be easily converted into a discrete binary classifier by applying a threshold. The classifier will then output positive if the probability is higher than the threshold, and negative if the probability is lower than the threshold. Different thresholds yield a different confusion matrix (a different classification of the examples in the validation set), and thus yield multiple points in ROC space. If we vary the threshold from $-\infty$ to $+\infty$ the so-called ROC curve is traced out.

However, note that GMLVQ is only a discrete classifier. In the GMLVQ toolbox [1] in the function `compute_costs` a distance-based score is calculated for each example $\mathbf{x}$. This

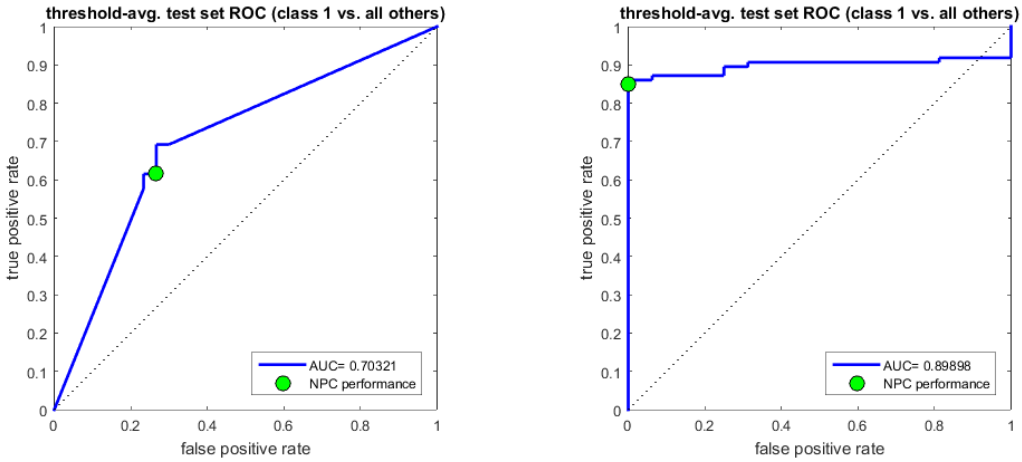
way GMLVQ becomes a sort of scoring classifier too. A choice for a score if $\mathbf{x}$ is positive is:

$$s = d_K - d_L \tag{5.1}$$

And if $\mathbf{x}$ is negative:

$$s = d_L - d_K \tag{5.2}$$

This way for each feature vector $\mathbf{x}$ in the validation set a score (or "probability") is calculated. In the function `compute_roc`, by varying the threshold and for each threshold calculating the corresponding trp and fpr different points in ROC space are obtained, this traces out an ROC curve and is a two-dimensional representation of classifier performance. To compare multiple classifiers a single scalar value would be handy to deal with [3]. Therefore the area under the ROC curve (AUC) can be calculated. An important statistical property of the AUC value is that the AUC value is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [3]. AUC will serve as a measure of classifier performance in the two-class experiments in this thesis. Performance measurement by ROC curve analysis is especially attractive since it is insensitive to changes in class distribution and class skew [3]. This follows directly from the fact that points in ROC space depend on tpr and fpr, which can both be calculated with a single column in the confusion matrix. Other metrics such as precision, accuracy and F-measure use values from both columns and are therefore sensitive to class distribution [3].



(a) A classifier with low performance.

(b) A classifier with good performance.

Figure 5.2: Two test set ROC plots of different classifiers resulting from cross validation, taking the mean of tpr and fpr at every threshold. The *green dot* represents NPC performance (Threshold = 0.5). Since the AUC of the right plot is higher than the AUC of the left plot, the right classifier's classification performance was better on the validation sets and we can expect better classification performance on new data.

Multiclass problems
For multiclass problems the performance of the classifiers are mainly compared by the confusion matrices.

5.2 Tecator

The tecator dataset consists of 215 labeled feature vectors $(\mathbf{x}, y) \in \mathbb{R}^{100} \times (1, 2)$. These are not time series, the dataset was used for its simplicity for first tests of the CGMLVQ method. Each feature vector describes the spectrometric curve which corresponds to the absorbance at 100 wavelengths (from 850mm to 1050mm) of finely chopped meat. The two classes are: Meat with small fat content (138 examples) and meat with large fat content (77 examples).

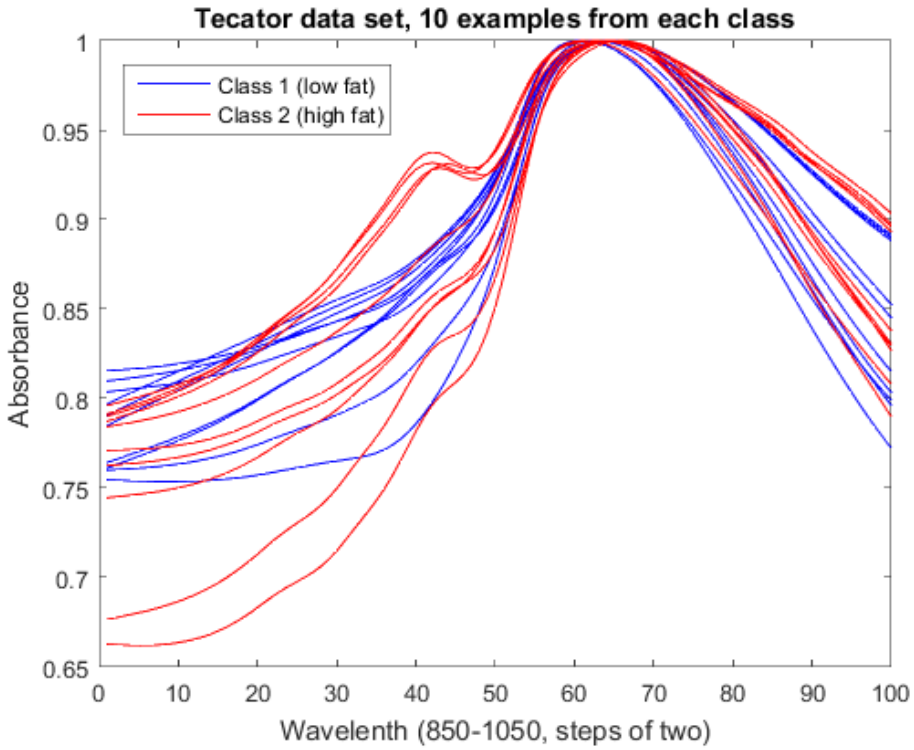
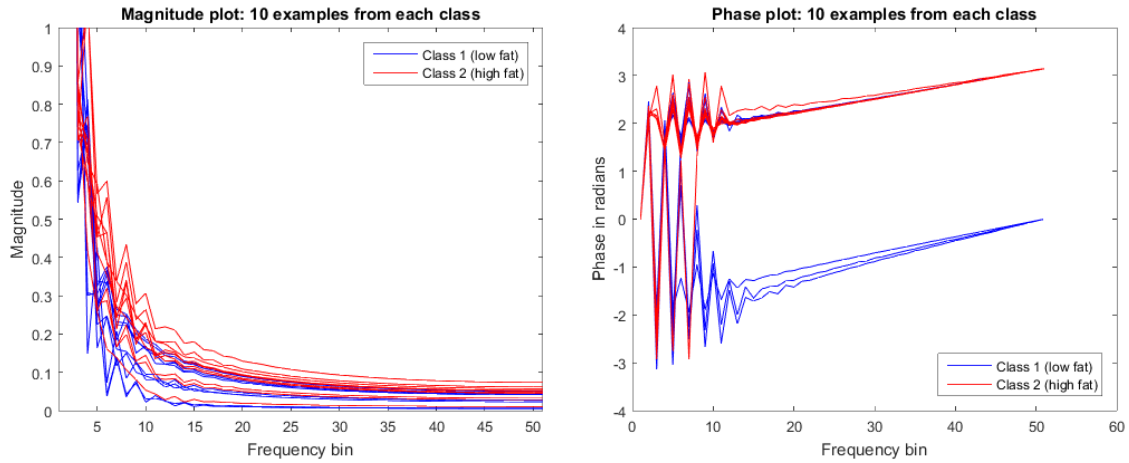


Figure 5.3: 10 examples taken from each class in the Tecator dataset

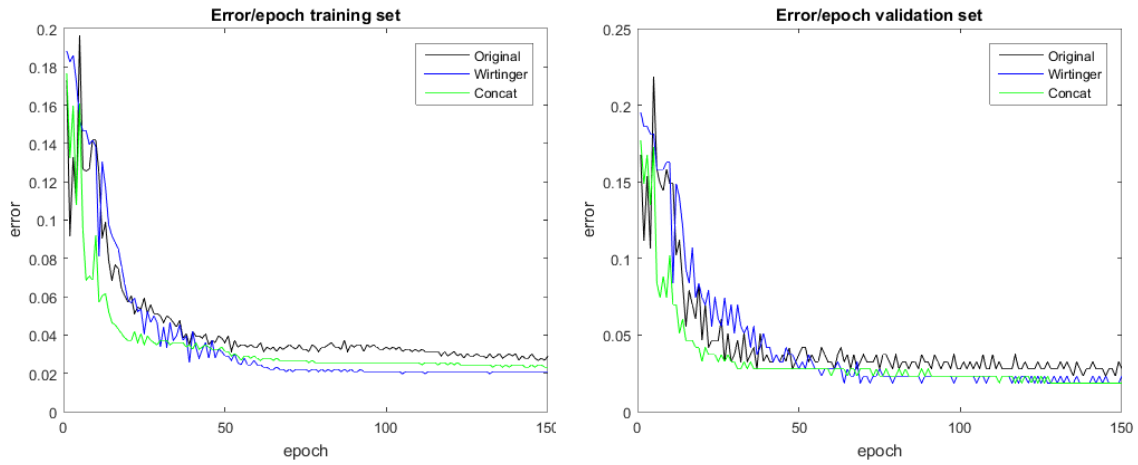
As can be observed from figure B.1, one particular difference between the two classes is in features 30-50 (Wave length 910-950nm). In class 2, the meat samples with high fat, a twist can be observed which is never present in these class 1 examples, the low fat-class.



(a) Magnitude plots of first 10 example of each class of the Tecator dataset

(b) Phase plots of first 10 example of each class of the Tecator dataset

Figure 5.4: Frequency spectra: 10 examples from each class



(a) Training error development.

(b) Validation error development.

Figure 5.5: Training curves of learning of the three different approaches (See legend). Learning in Fourier space achieves a slightly lower error.

Figure 5.5 shows the error percentages on the training and validation set while training in Original space, complex Fourier space and concatenated Fourier space. The Fourier space representations achieve a slightly lower error rate than the original space.

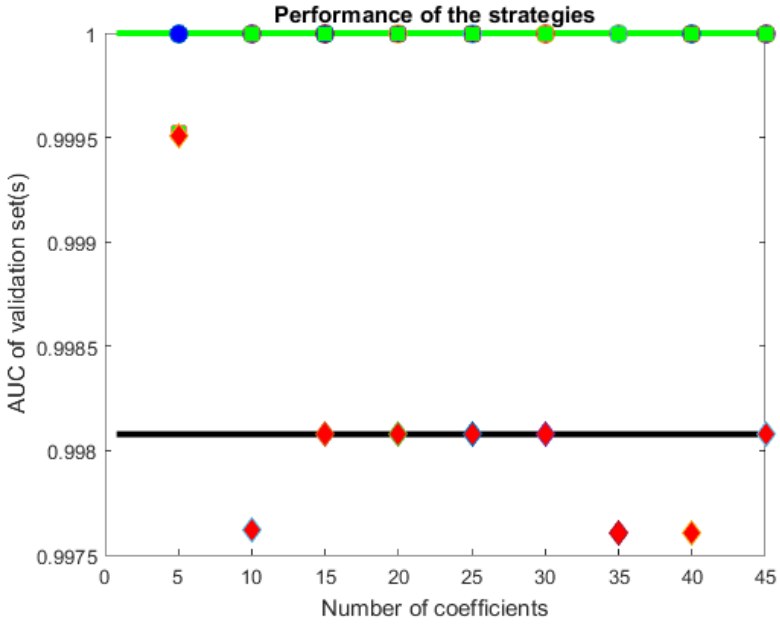
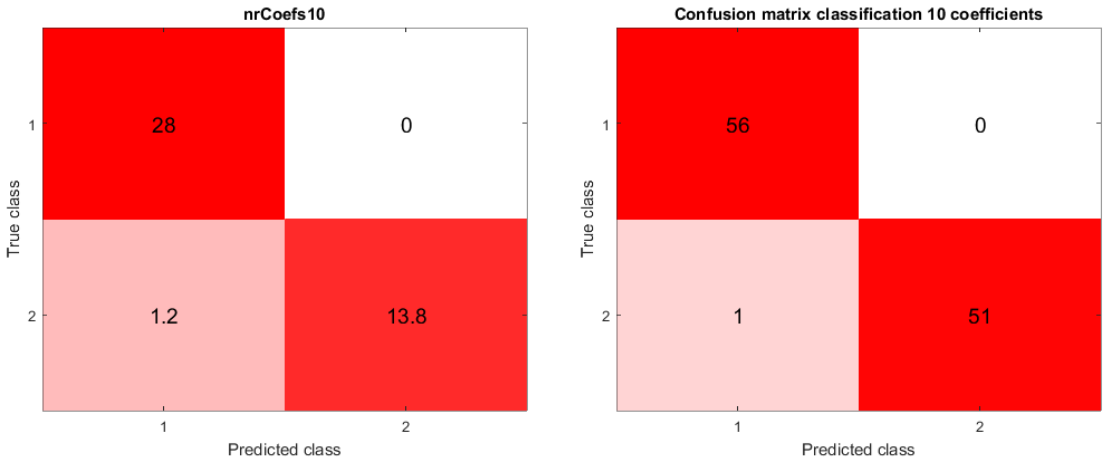


Figure 5.6: Comparison of the validation performance expressed as AUC values in dependence of the number of coefficients. The black, green, and blue solid lines represent the AUC value for the classification in original space, concatenated- and complex coefficient space respectively. Note that the blue solid line is behind the green line. *Blue circles* represent results achieved in complex Fourier coefficient space at n frequencies. *Green squares* represent results achieved in concatenated Fourier coefficient space at n frequencies. *Red diamonds* represent performance as achieved from smoothing of the data only.



(a) Cross validation 20% val. 5 runs

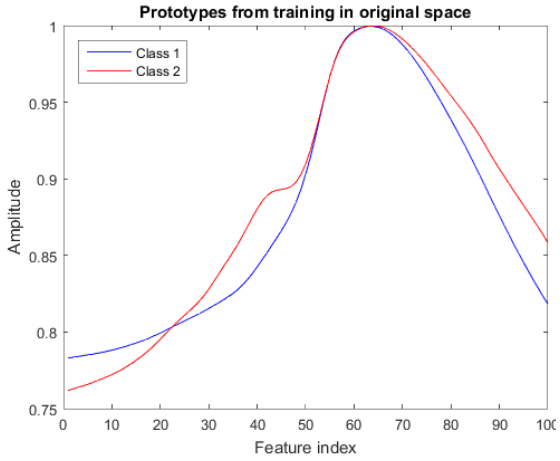
(b) One run 50% val.

Figure 5.7: Confusion matrices as obtained for two different test cases, both as a resulting from learning in 10-coefficient Fourier space. The results are similar for the cross validation with 20% of the examples dedicated to validation (left) and the pre-specified split with 50% dedicated to validation (right).

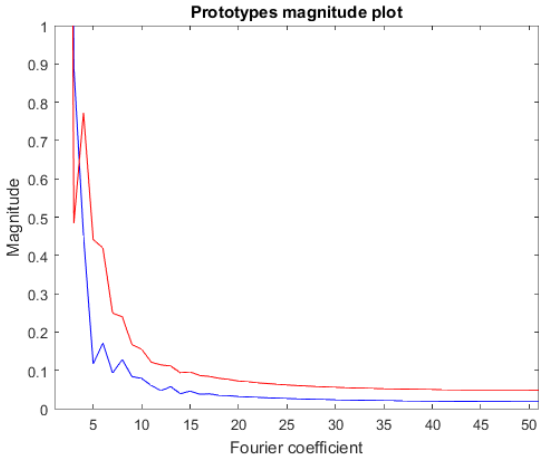
In figure 5.6 the results from cross validation are shown (mean of 5 runs) in which 20% of the vectors were taken for validation. The resulting classifiers for all the strategies perform

well on the validation sets as concluded from figure 5.6: The AUC of the classifier on the validation set in original space, concat space and complex space is consistently high. Although the differences are small, the AUC of the Wirtinger- and concatenation method is exactly 1 in all cases while the AUC of the classifier in the original space is a tiny bit lower.

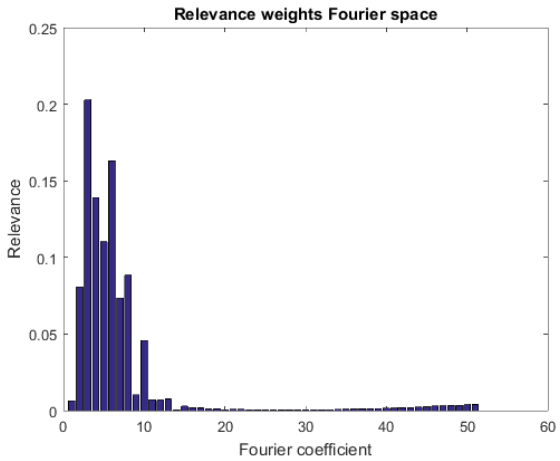
We can see that in the Fourier space with as few as 10 (or even 5) Fourier coefficients the examples can already be assigned with high accuracy into the correct classes. The corresponding classification in the case of 10 Fourier coefficients is shown in figure 5.7a, only small error is made in the classification of class 2 validation examples. In another test, the dataset was split in 107 training vectors and 108 validation vectors and training and validation was applied in the space of 10 Fourier coefficients. In figure 5.7b the corresponding confusion matrix is shown in which it can be seen that only one misclassification was made.



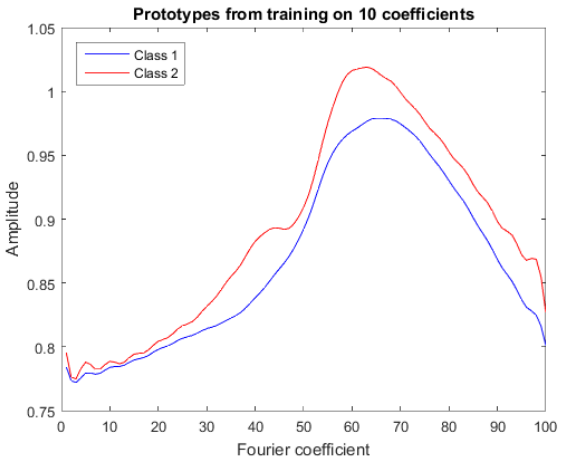
(a) Prototypes training in original space.



(b) Prototypes full Fourier space.



(c) Feature relevance in Fourier space.



(d) Backtransformed prototypes 10 coefficients.

Figure 5.8: The prototypes as resulted from training in original space and from training in coefficient space. Figure **c** shows the relevance profile in coefficient space, indicating that the the most relevant coefficients are among the first 10 coefficients.

In figure 5.8c the feature relevance in full Fourier space is plotted (diagonal $\mathbf{\Lambda}_c$ matrix),

in which it shows that the discriminative power is mainly in the 10 lower frequency coefficients.

Note that this dataset was chosen as a simple dataset to verify the correctness of the implementation of CGMLVQ based on Wirtinger derivatives. However, even for this dataset there are some benefits in the functional representation of the data. The clear benefit of the functional representation is the reduction of the dimensionality; The representation with 10 coefficients still gave at least as good performance while the dimensionality was reduced by 90% compared to the dimensionality in the original space. Although the results in figure 5.6 are close to each other, a consistent AUC value of 1 was measured in Fourier space, while slightly lower values were measured in original space and back transformed original space. This may indicate that the functional representation itself is beneficial for the classification, and not only the smoothing of the data that was introduced in the truncated Fourier versions.

5.3 Plane

The plane dataset [2] consists of 210 labeled feature vectors representing time series of the form $(\mathbf{x}, y) \in R^{144} \times \{1, 2, 3, 4, 5, 6, 7\}$. 105 examples are dedicated for training and 105 for validation in the prespecified split in the repository.

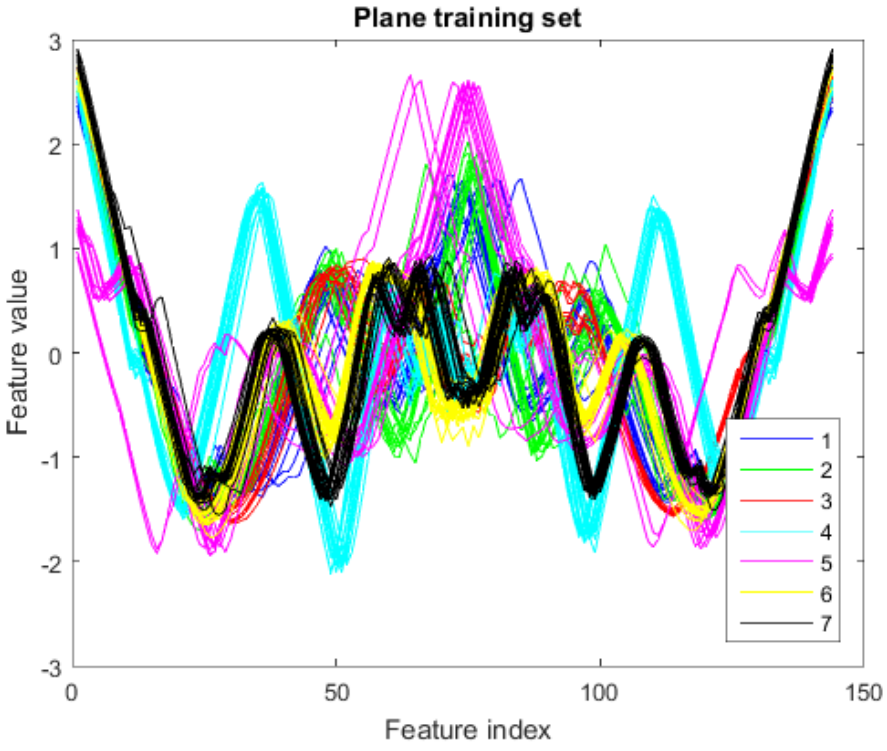


Figure 5.9: The feature vectors in the training set.

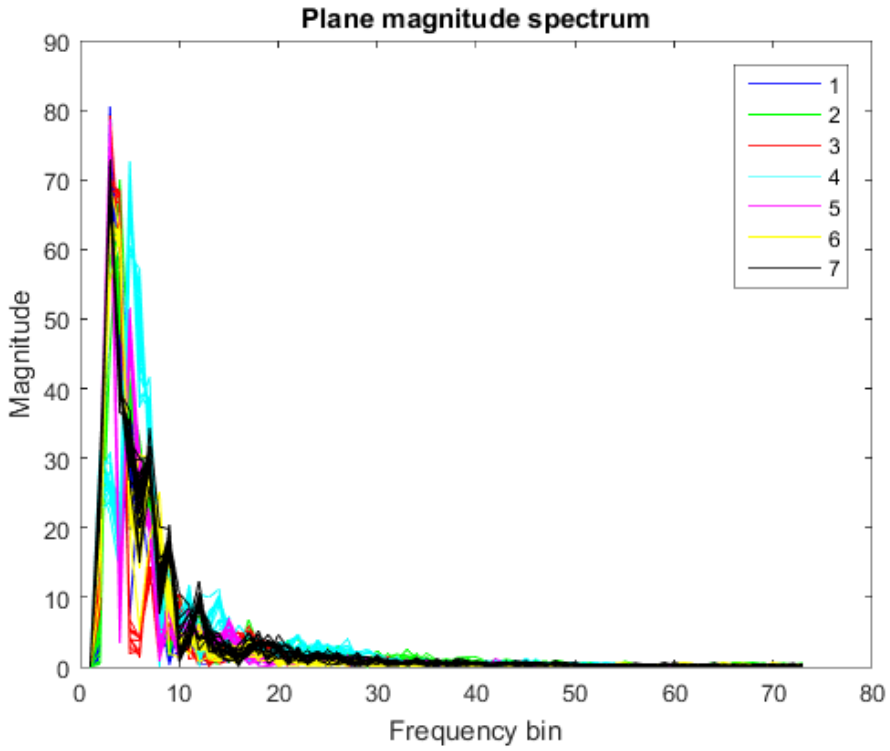


Figure 5.10: The magnitude plots of the features vectors in the training set.

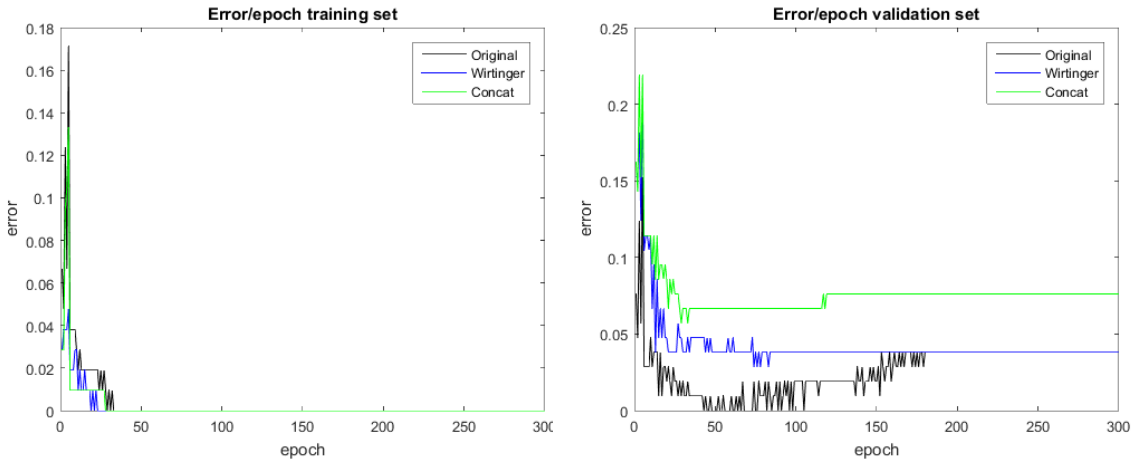
An analysis of figure 5.9 reveals that examples in the different classes have a periodic nature. For this reason the Fourier transform would make more sense as a functional representation of the discrete sampled time series. The difference between the classes is well observable; The amplitude of the different oscillations between the classes differs, as well as the frequency. However, the classes that resemble each other the most are class 6 and 7, represented by black and yellow. In figure 5.10 the amplitude plot is drawn. In this plot the characteristics of the classes can be observed; For instance the light blue peak is at a higher frequency than the pink peak. This is in accordance with figure 5.9 in which the light blue class has a higher frequency more present in it and the pink class a lower frequency. Similar argumentation is relevant to the rest of the spectra.

Along the lines of the first strategy the results that arise when performing GMLVQ training (one prototype per class) in the original space of the data is performed. In figure A.1 the training statistics per epoch (cost, error, etc) on the train set are shown. As becomes clear from figure 5.9 there are no outliers observed in the training set. This is in accordance with the results in figure A.1 in which the cost function achieves a value close to -1 and the error on the train set is 0; GMLVQ has been able to achieve a positioning of the prototypes and relevance matrix such that all examples in the training set are classified correctly.

The performance of the resulting classifier on the validation set is shown as a confusion matrix in figure 5.13a. A few errors have been made on the validation set in classifying examples that belong to class 2 and 3.

The training set statistics as arisen from GMLVQ on the full Fourier spectra of the data ($\mathbf{x} \in \mathbb{C}^{73}$) are shown in figure A.2. The results are similar to the results obtained from

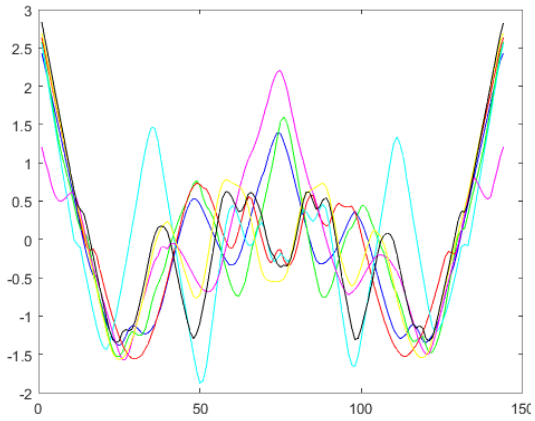
the training in the original space. Also by comparing the resulting prototypes in the real space (figure 5.12a) and those resulted in the coefficient space (figure 5.12c) it becomes apparent that they are almost identical. In figure 5.12d the Fourier coefficients most relevant in this classification problem are shown. The coefficients that represent the lower frequencies are among the most relevant coefficients in the spectrum. Figure 5.11 shows the development of the train- and validation error during training for the Wirtinger complex Fourier-, concatenation- and the original space approach. On the validation set classification in complex Fourier space and in original space reach the same error percentage. The concatenation approach reaches a higher error.



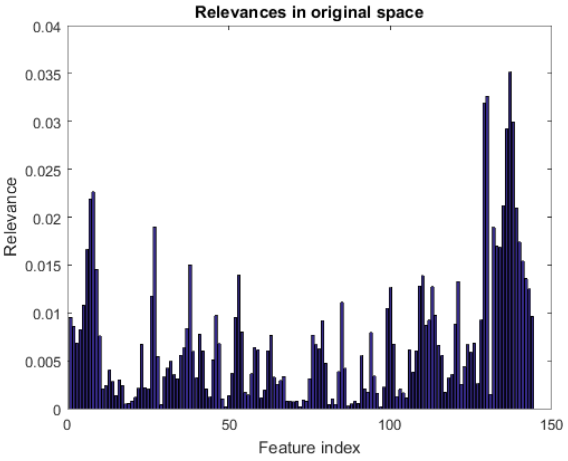
(a) Training error development.

(b) Validation error development.

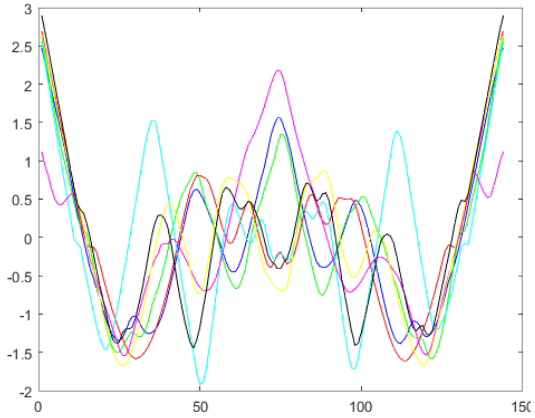
Figure 5.11: Training curves for learning in original space, complex Fourier space and concatenated Fourier space.



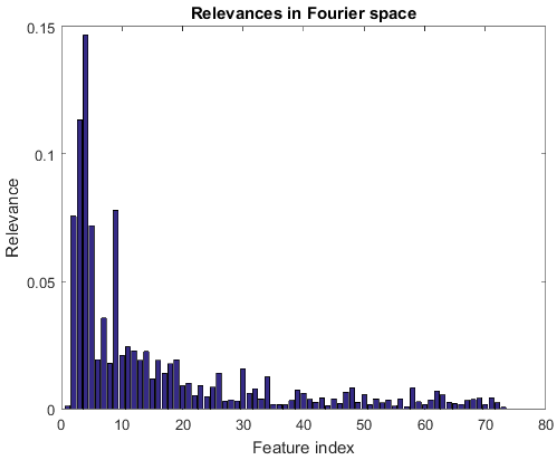
(a) Resulting prototypes training in real space.



(b) Relevances of the real dimensions.



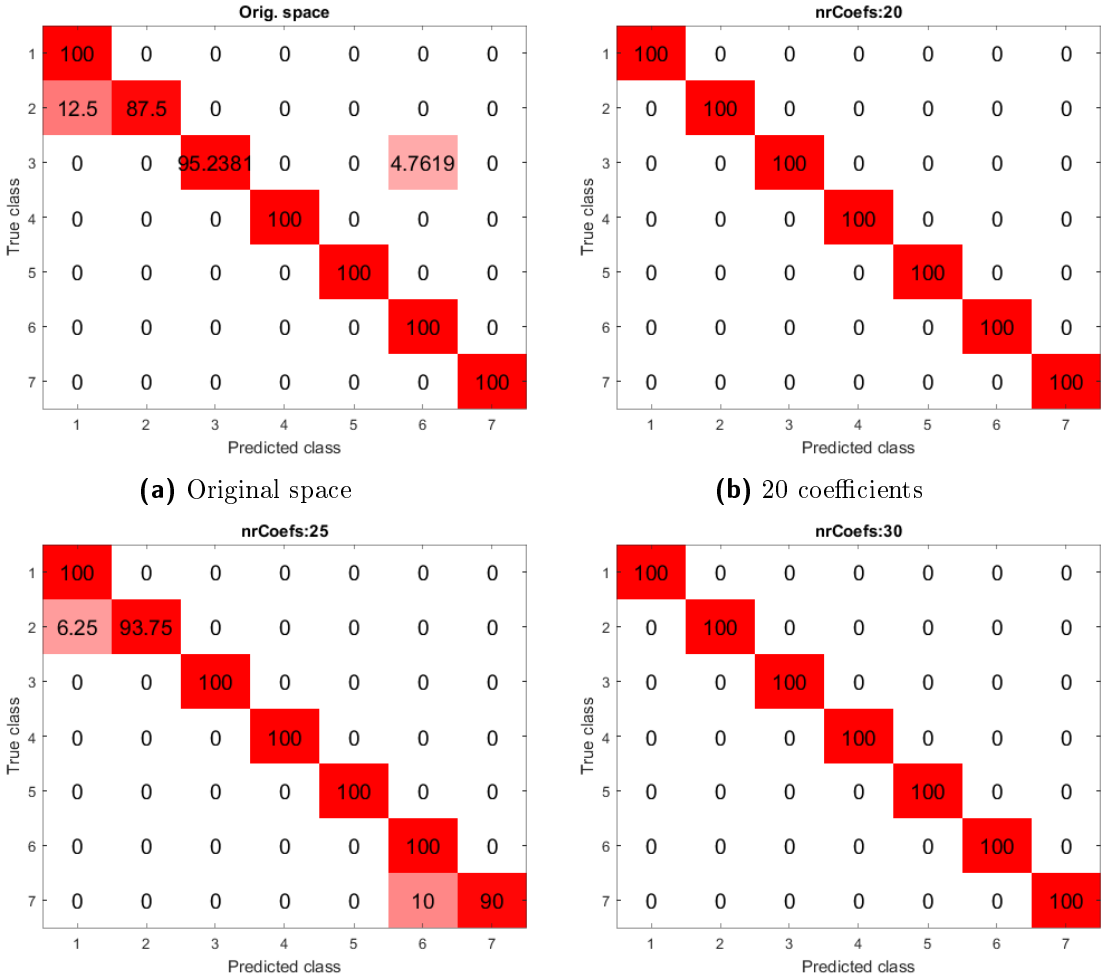
(c) Back transformed prototypes training in coefficient space.



(d) Relevances of the Fourier coefficients.

Figure 5.12: The prototypes and feature relevance of learning in original space (Fig. 5.12a, 5.12b) and coefficient space (Fig. 5.12c, 5.12d).

With 20 coefficients the classification performance on the validation set is better than the performance in the original space. Figure 5.13 shows the performance for different cases on the validation set by confusion matrices. In the confusion matrix from the original space, small error was made in the classification of class 2 and 3 on the validation examples (Figure 5.13a). In figure 5.13b in which spectra truncated at 20 frequencies were used, it turns out that the classification of the validation set went without error. As was shown in figure 5.12d On this dataset the main discriminative features are in the lower 20 coefficients and this may explain the good performance with as few as 20 coefficients (which is 14% of the original dimensionality) compared to the performance in the original space.



(a) Original space

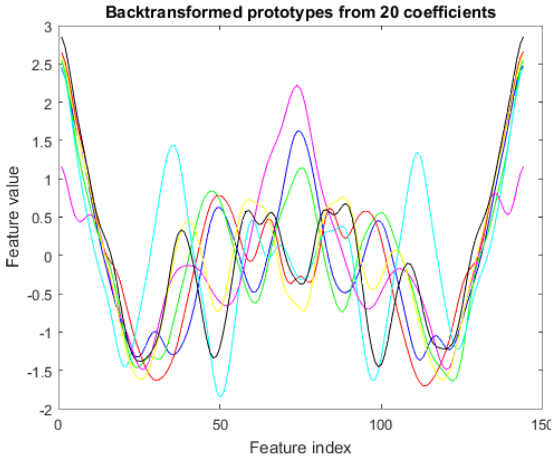
(b) 20 coefficients

(c) 25 coefficients

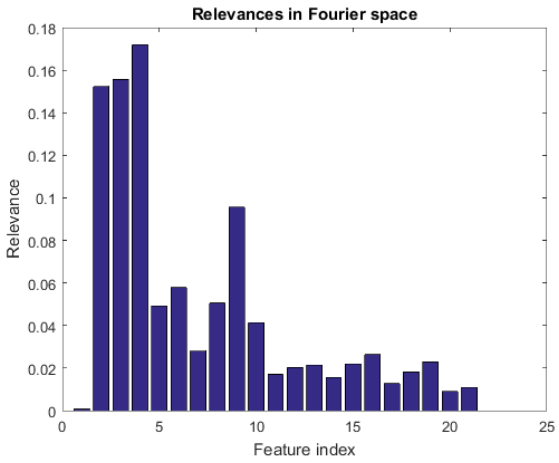
(d) 30 coefficients

Figure 5.13: Confusion matrices in percentages as obtained for learning in original space (Fig. 5.13a) and Fourier space truncated at different numbers of coefficients (Fig. 5.13b, 5.13c, 5.13d).

In figure 5.14a the back transformed prototypes for the case $n = 20$ are displayed. Since most information was in the lower frequencies, they are very similar to the prototypes in original space (figure 5.12a). The difference being that the prototypes in 20 coefficient space are slightly smoother than the prototypes in the original space because of the omission of the higher frequencies.



(a) The prototypes which resulted from training on 20 Fourier coefficients back transformed.



(b) Feature relevance 20 coefficients.

Figure 5.14: Prototypes and feature relevance as obtained for learning in 20-Fourier coefficient space. The prototypes are transformed back to the original space and a comparison with fig. 5.12a shows that the two resemble each other closely. The prototypes in 20-Fourier coefficient space are smoother than in original space.

The plane dataset turned out to be an appropriate dataset for classification in the Fourier space. This is supported by the experiment but also the observation that the data has a periodic nature. In the tests it was shown that with only 20 Fourier coefficients (A reduction of 86% of the dimensionality in the time domain) a better classification performance in the validation was obtained than the performance in the original space. The prototypes that resulted from the back transformation of the 20 coefficient-spectra to the time domain closely resembled the prototypes that arose from training in the time domain.

5.4 MALLAT

The MALLAT dataset [2] consists of 2400 labeled feature vectors $(\mathbf{x}, y) \in \mathbb{R}^{1024} \times \{1, 2, 3, 4, 5, 6, 7, 8\}$ representing time series in eight different classes. In the prespecified split in the UCR repository 55 of the vectors are assigned for training and 2345 for validation.

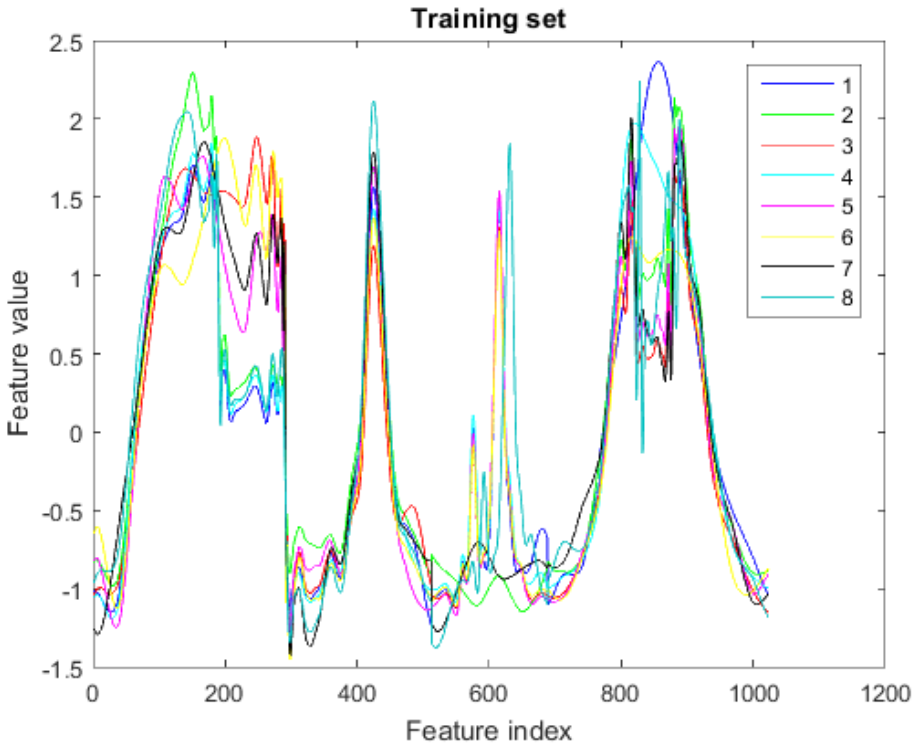


Figure 5.15: Training examples MALLAT

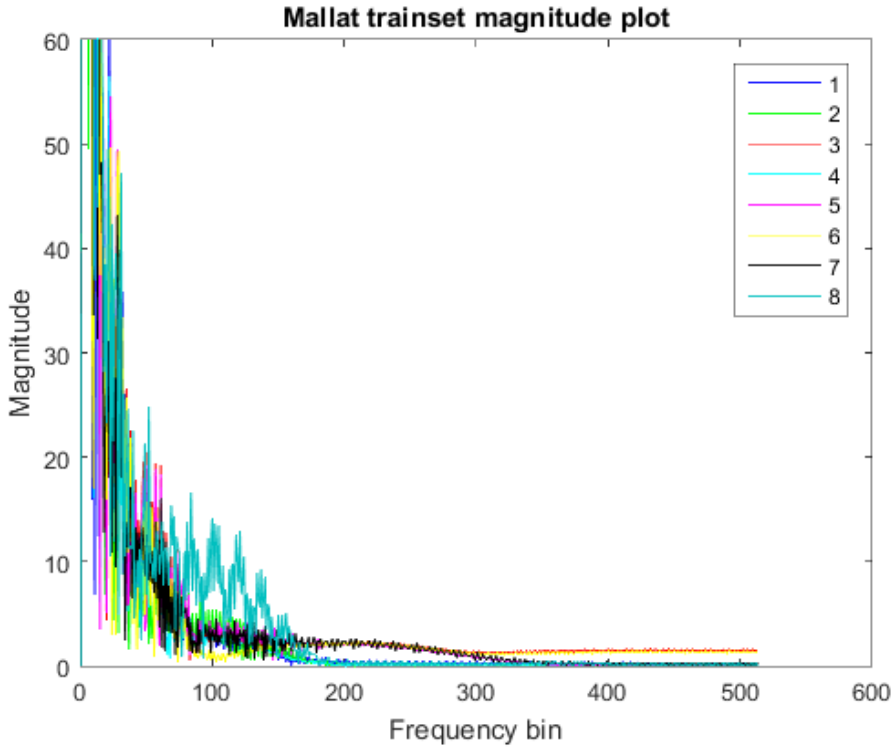
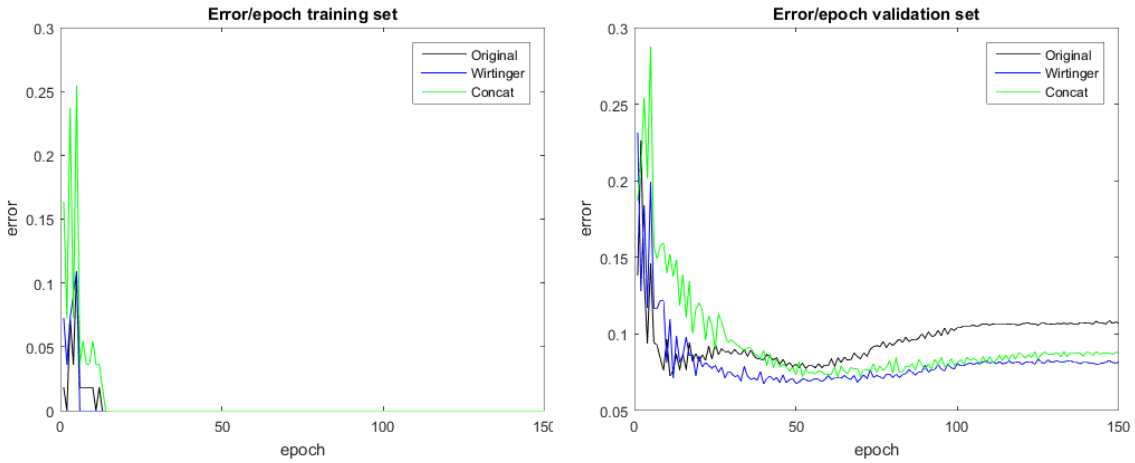


Figure 5.16: Training examples MALLAT: Magnitude plots.

In figure 5.15 the examples show a presence of low- and high frequency oscillations.



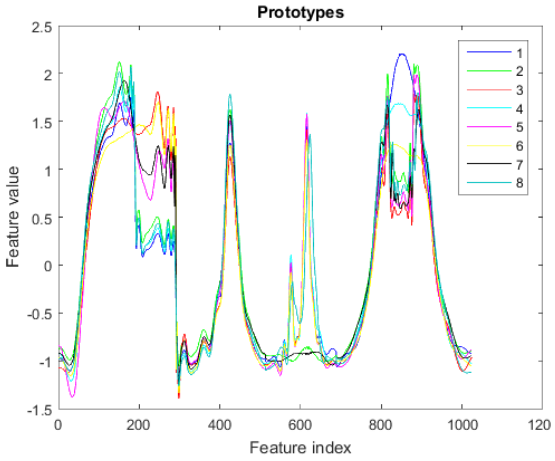
(a) Training error development.

(b) Validation error development.

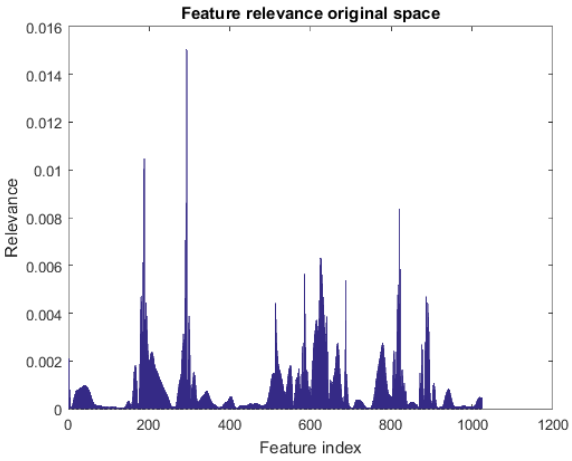
Figure 5.17: Training curves for the three methods (see legend). Overfitting occurs in the original space after 50 epochs as the validation error increases. In coefficient space, the overfitting does not happen (or far-less), the result is that both Fourier space representations achieve a lower error.)

Figure 5.17 shows the development of the training- and validation error during training. On the train set all the methods achieve zero error. On the validation set (fig. 5.17b) the Wirtinger method achieves the lowest error, followed by the concatenation method. The method in the original space achieves the highest error. After 50 epochs the validation error in the original space increases, indicating a presence of overfitting that arises in original space. As was mentioned in the introduction, in recent research it was claimed that training in coefficient space can help to reduce overfitting effects. Here we indeed see that the overfitting effect is far-less present in coefficient space, in accordance with the claim in recent research.

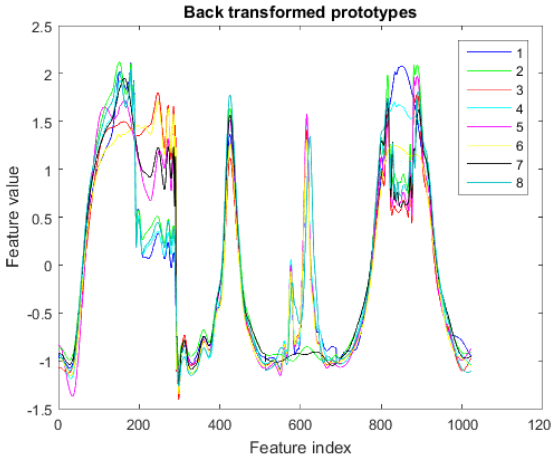
In figure 5.18 we see the classifiers (prototypes and diagonal of matrix) of the classifier in original space (figure 5.18a and 5.18b) and in Fourier space (figure 5.18c and 5.18d). In figure 5.18d the information in the high frequency coefficients appears to have low relevance in the classification. The highest peaks are the in the first 0-80 Fourier coefficients.



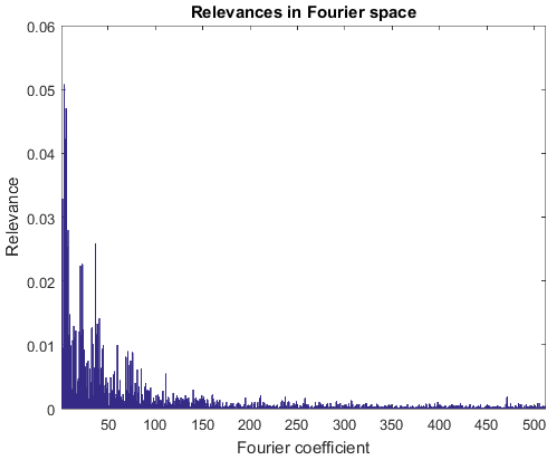
(a) Prototypes in original space.



(b) Feature relevance original space.



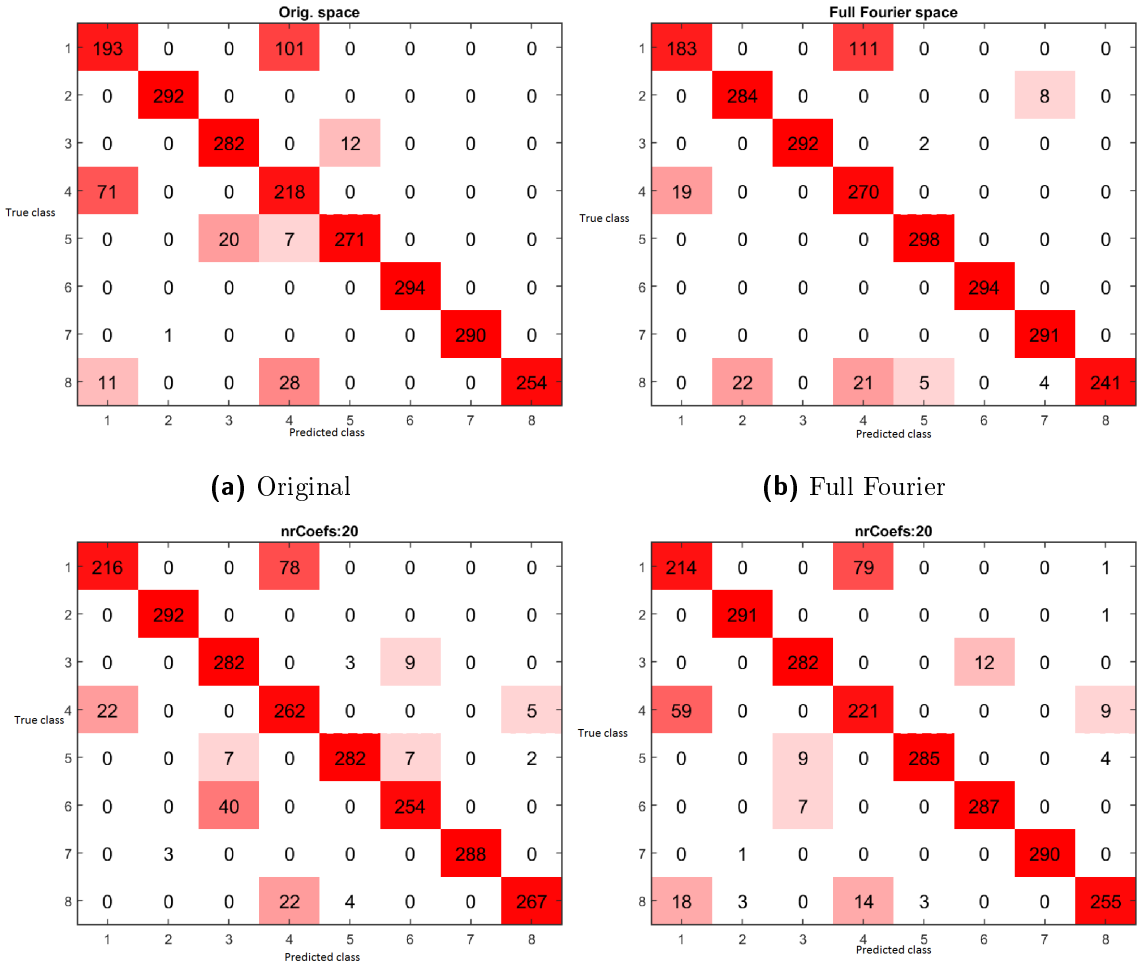
(c) Prototypes backtransformed from Fourier space.



(d) Feature relevance Fourier space.

Figure 5.18: The prototypes and feature relevance of learning in original space (Fig. 5.18a, 5.18b) and coefficient space (Fig. 5.18c, 5.18d).

From the confusion matrices in figure 5.19 which indicate the exact classifications of the examples in the validation set we see that the performance of the approximation with 20 Fourier coefficients (2% of the original dimensionality) is better than the performance in original space: 202 misclassifications in 20 coefficient Fourier space vs. 251 misclassifications in original space. When the data was transformed from an approximation with 20 coefficients to the original space and training was applied in original space, 220 misclassifications were recorded (fig. 5.19d). Since this performance is better than the raw data in original space but worse than performance in coefficient space, both smoothing and the functional representation, namely the Fourier representation of the time series, have a positive effect on performance.



(a) Original

(b) Full Fourier

(c) 20 comp. coefficients

(d) 20 coefficients back transformed

Figure 5.19: Confusion matrices as obtained for learning in original space (Fig. 5.19a), Full Fourier space (Fig. 5.19b), 20-coefficient Fourier space (Fig. 5.19c), original space after a back transformation from 20-coefficient Fourier space (Fig. 5.19d).

5.5 Symbols

The dataset Symbols [2] consists of 1020 labeled feature vectors $(\mathbf{x}, y) \in \mathbb{R}^{398} \times (1, 2, 3, 4, 5, 6)$. 25 of the feature vectors are specified as training examples and 995 as validation examples.

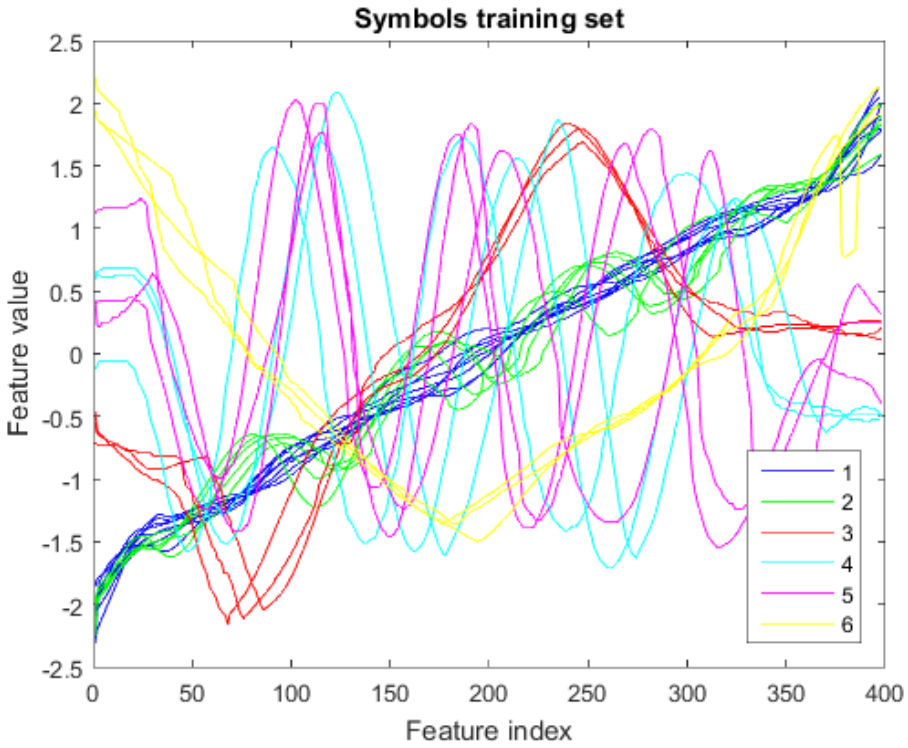


Figure 5.20: The training examples of the "Symbols" dataset.

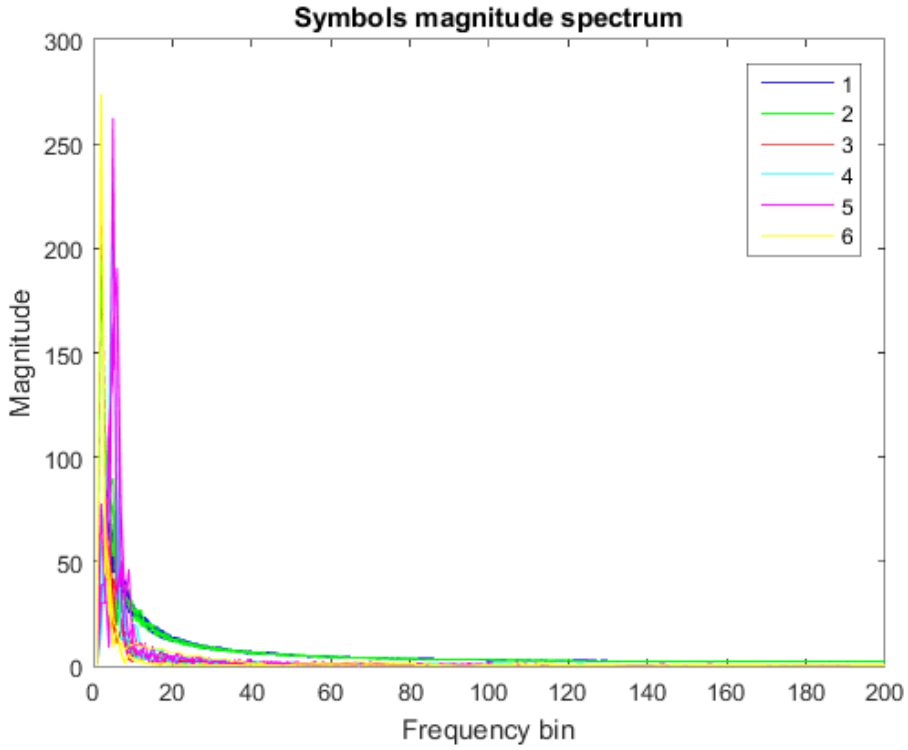


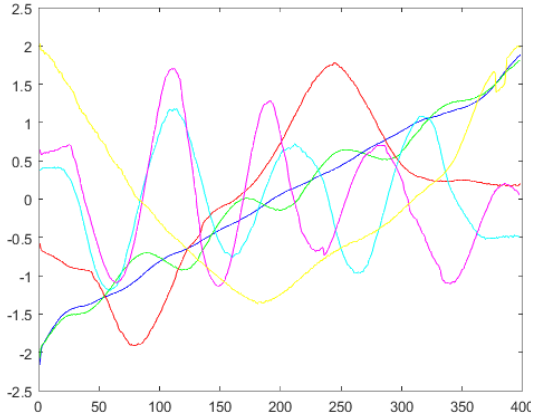
Figure 5.21: Magnitude plots of the training examples in the "Symbols" dataset.

In figure 5.20 the training set is displayed. The time series show periodic behavior, especially the classes 4, 5 and 6. The 4th and 5th classes show around four oscillations, the 6th class half an oscillation. The time series of the classes 1 and 2 lie close to each other, but the examples from class 2 show oscillations of a higher amplitude. Both classes 1 and 2 have higher frequency components in them compared to the other classes. In figure 5.21 the characteristics can be observed; A large yellow peak representing the low frequency component in examples of class 6 and a few frequencies higher a large pink peak representing three or four oscillations in class 5. We also observe the considerable amplitudes of class 1 and 2 at the higher frequencies.

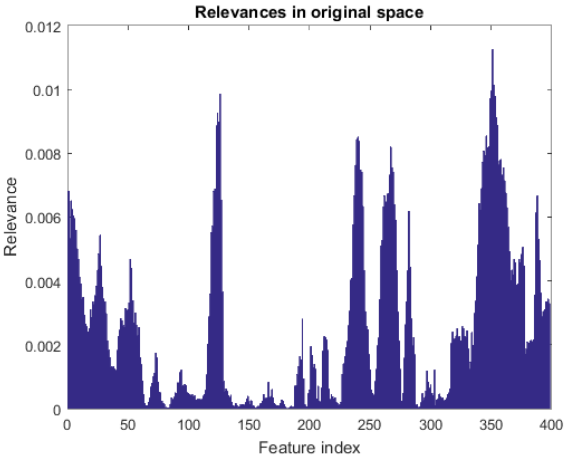
The two pairs of classes that would be most difficult to classify is the pair (1,2) and the pair (4,5). Both of these classes show a considerable overlap.

First the classifier has been trained on the training set in the original space of the data. The results in figure A.5 show that the cost function per example approaches -1 and 0 training error is achieved. Figure 5.25a shows the validation performance of the resulting classifier on the 995 validation examples. We see that the classifier has a low error rate in classifying examples labeled (1,3). Indeed the errors made by the classifier are higher on validation examples labeled (2,4,5). More than half of the examples labeled 2 has been classified as 1.

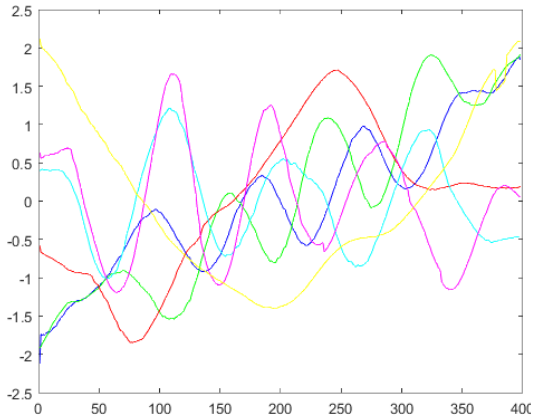
In figure A.5 the training set performance is displayed when the dataset is represented with all Fourier coefficients. ($\mathbf{x} \in \mathbb{C}^{200}$). Similar results are achieved compared to training in the original space. However, there are differences in the validation as seen in figure 5.25b. Classification of class 6 examples achieves a significantly better result in the Fourier space in which it went almost without error. Also classification of class 4 examples has improved by 30%.



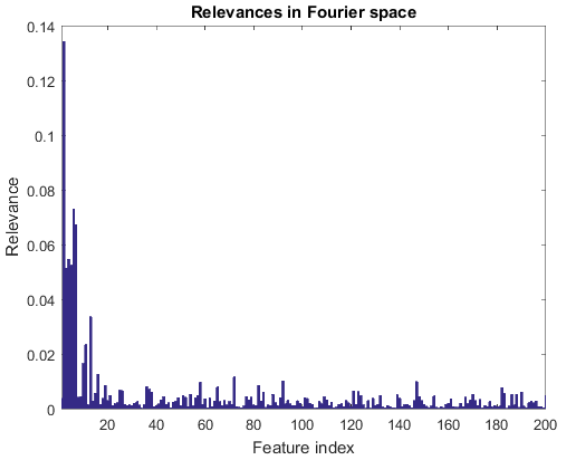
(a) Resulting prototypes training in real space.



(b) Relevances of the real dimensions.



(c) Back transformed prototypes training in coefficient space.



(d) Relevances of the Fourier coefficients.

Figure 5.22: The prototypes and feature relevance of learning in original space (Fig. 5.22a, 5.22b) and coefficient space (Fig. 5.22c, 5.22d).

From figure 5.22d it can be seen that the lower coefficients (below the 20th coefficient) contain peaks indicating high relevance.

As we have seen, the performance in the coefficient space is better than the performance in the real space. This is mainly because the examples of class 4 and 6 are much better classified in the coefficient space than in the real space. In figure 5.25c which shows the results of the 20 coefficients case classifier performance is similar to the performance in the full Fourier coefficient space. Therefore performance does not decrease much when approximating the time series with 20 Fourier coefficients. For these reasons the benefits of the Fourier representation for this dataset are a better classification rate and the opportunity to reduce the dimensionality. However, the classification performance of class 2 and 5 examples remains poor.

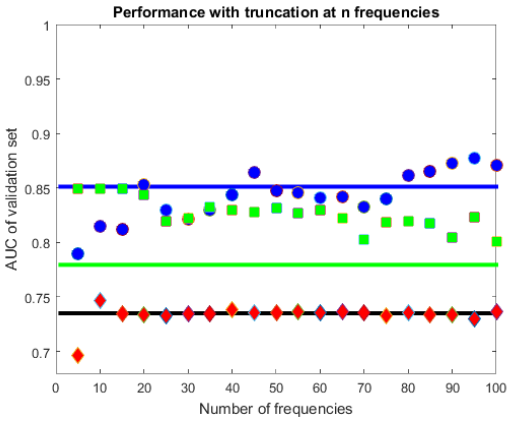


Figure 5.23: Comparison of the validation performance expressed as AUC values in dependence of the number of coefficients (Class 1 vs others). The black, green, and blue solid lines represent the AUC value for the classification in original space, concatenated- and complex coefficient space respectively. *Blue circles* represent results achieved in complex Fourier coefficient space at n frequencies. *Green squares* represent results achieved in concatenated Fourier coefficient space at n frequencies. *Red diamonds* represent performance as achieved from smoothing of the data only.

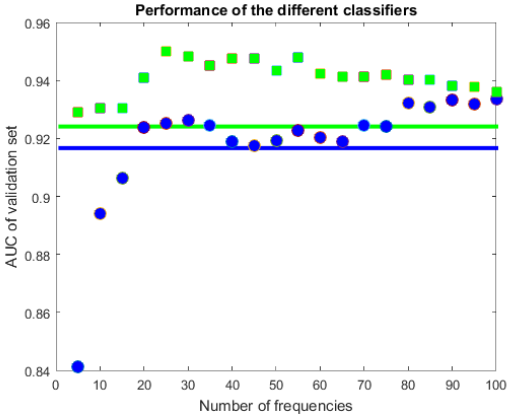
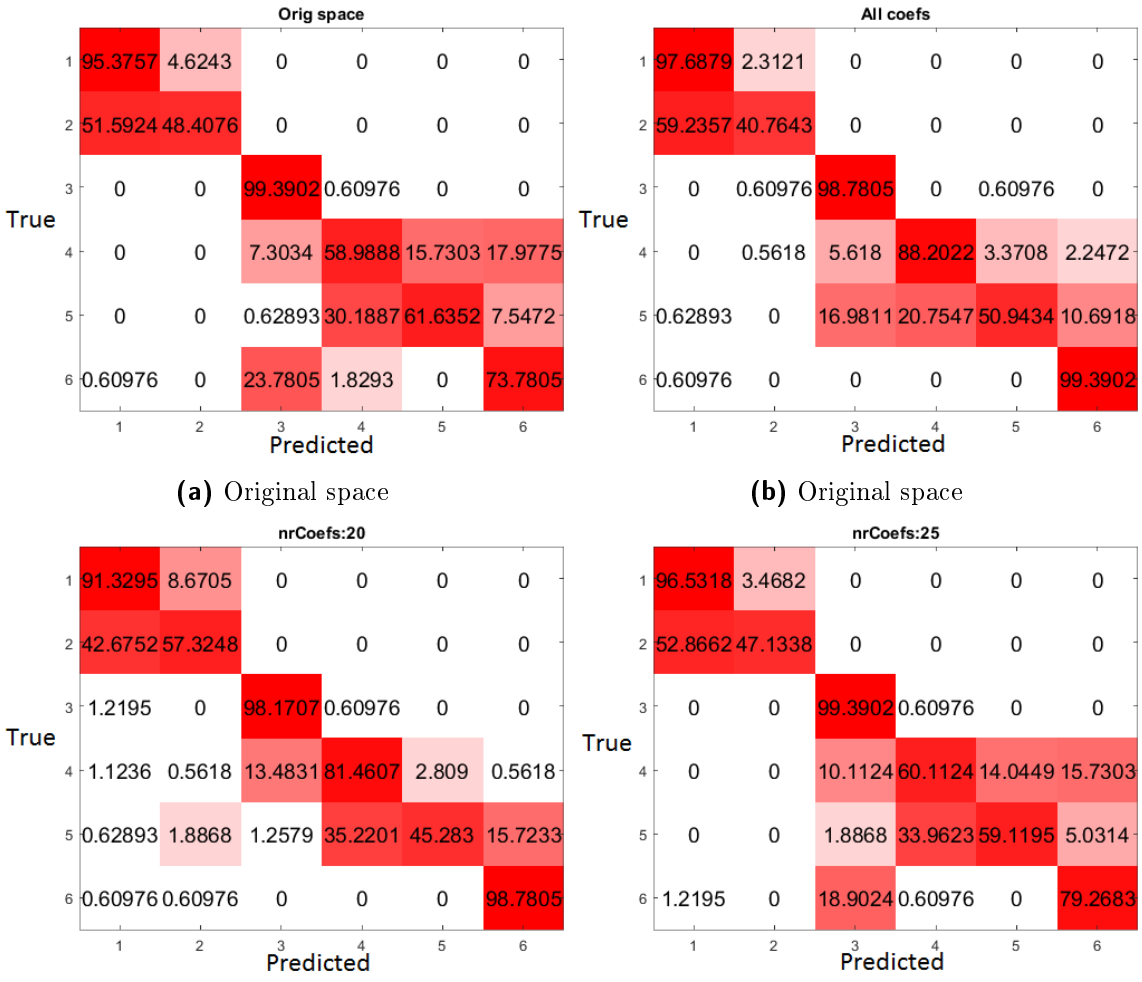


Figure 5.24: Mean of 5 runs, AUC values on validation set (Class 1 vs others, 30% of the data). *Solid blue line*: AUC full complex Fourier space. *Blue circles*: AUC complex Fourier space truncated at n frequencies. *Solid green line*: AUC full concatenated Fourier space. *Green circles*: AUC concatenated Fourier space truncated at n frequencies.

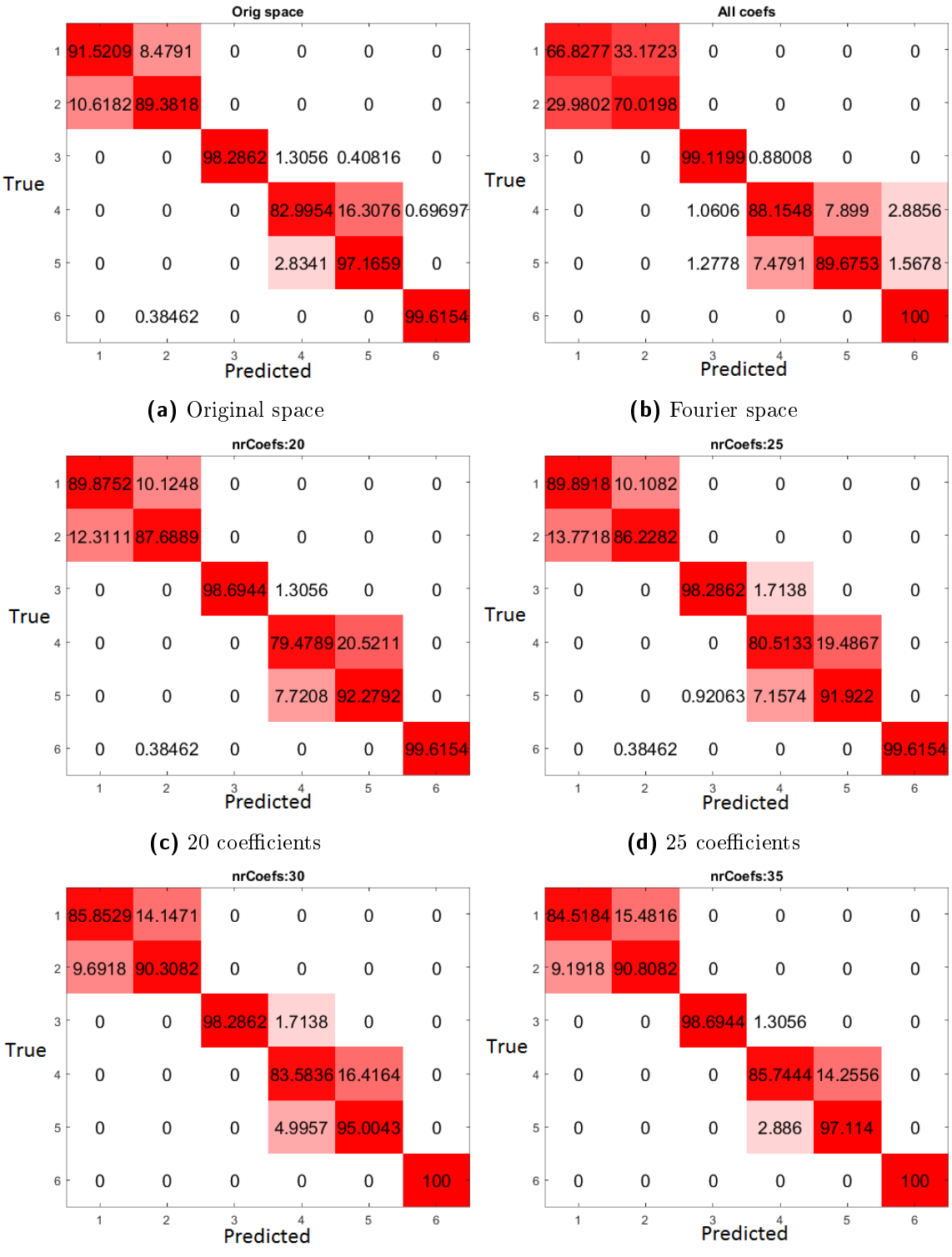


(a) Original space **(b)** Original space

(c) 20 coefficients **(d)** 25 coefficients

Figure 5.25: Confusion matrices as obtained for learning on the prespecified training set.

As discussed earlier, class 2 is overlaps with class 3 and class 5 overlaps with class 4. The prespecified training set is probably too small to learn the differences between the classes correctly. To test this assumption an experiment has been performed in which 70% of the data is for training and 30% for validation. The tests results are the mean of 5 runs. As shown in 5.26 the classification accuracy for the problematic classes 2 and 5 has increased significantly in all cases. Also the classification in the original space has a much better accuracy, this leads to the assumption that the classifier in the original space needed more training examples for the same level of accuracy than the classifier in the Fourier space. In case the time series are approximated with 20 coefficients (5% of the dimensionality in the original domain) the classifier is almost as accurate as the classification in the original domain. (Figure 5.26c).



(a) Original space

(b) Fourier space

(c) 20 coefficients

(d) 25 coefficients

(e) 30 coefficients

(f) 35 coefficients

Figure 5.26: Mean of the confusion matrices as obtained from 5 runs on a variable validation set consisting of 30% of the data, training on the other 70%. The performance in 20-coefficient Fourier space (Fig. 5.26c) is close to the performance in the original space (Fig. 5.26a).

The analysis showed that for the dataset Symbols there are advantages of representing the time series examples in the Fourier space. This became apparent by analysis of the performance on the prespecified train/validation set, and also by analysis of the results of the cross validation. In the first classification of specific classes became more accurate in the frequency domain, also on as few as 20 coefficients. In the cross validation it turned out that using 20 Fourier coefficients (5% of the dimensionality in the original space) gave a performance close to the performance as observed in the original domain.

6 Final thoughts

The classification of time series in Fourier space has proven to be beneficial compared to original space, especially in certain cases. In time series with a periodic nature the Fourier functional representation is suitable because of the sinusoidal basis functions and proved to be beneficial to classification results. The same or better performance was achieved with a Fourier approximation truncated at n coefficients in all experiments. In line with previous research, training in Fourier coefficient space also reduced overfitting effects (MALLAT experiment). The performance gain in Fourier space can be explained by the appropriate functional representation in case of periodic time series data. Note that the truncation at n coefficients yielded a reduction in dimensionality sometimes as high as 80-90%. This leads to a huge speed-up in the training process since the number of free parameters is reduced quadratically (Quadratic because of the $\mathbf{\Omega}$ matrix). Note that in some experiments, for example the plane dataset, the performance in original space and truncated Fourier space was similar. Still, using the Fourier transform as a means for dimensionality reduction is a significant benefit while keeping the same or similar performance.

In Fourier space, the complex Wirtinger method showed a similar performance in the experiments than the conventional concatenation method on the datasets which were tested. However there were benefits of Wirtinger GMLVQ: The number of free parameters is reduced and the method respects the complex nature of the data. Prototypes are in the same complex space as the data and the matrix accounts for the relevance of the complex dimensions. The method can therefore also account for correlations between the complex dimensions. The concatenation method may be more appropriate if the correlations are mainly between real- and imaginary parts.

For future study, note that in this thesis cross validation tests were performed in Fourier space truncated at n frequencies for different n and the results were interpreted. Generally, the performance was robust for different n and one specific n was chosen to discuss in more detail. This n generally satisfied the criteria of good performance and significantly lower dimensionality. A systematic approach for identifying the optimal number of coefficients could be an interesting continuation for future research. Note that this suggestion was made in [9], and such a method/approach will also be of great value for time series classification in Fourier space.

A Appendix: Learning results

A.1 Plane

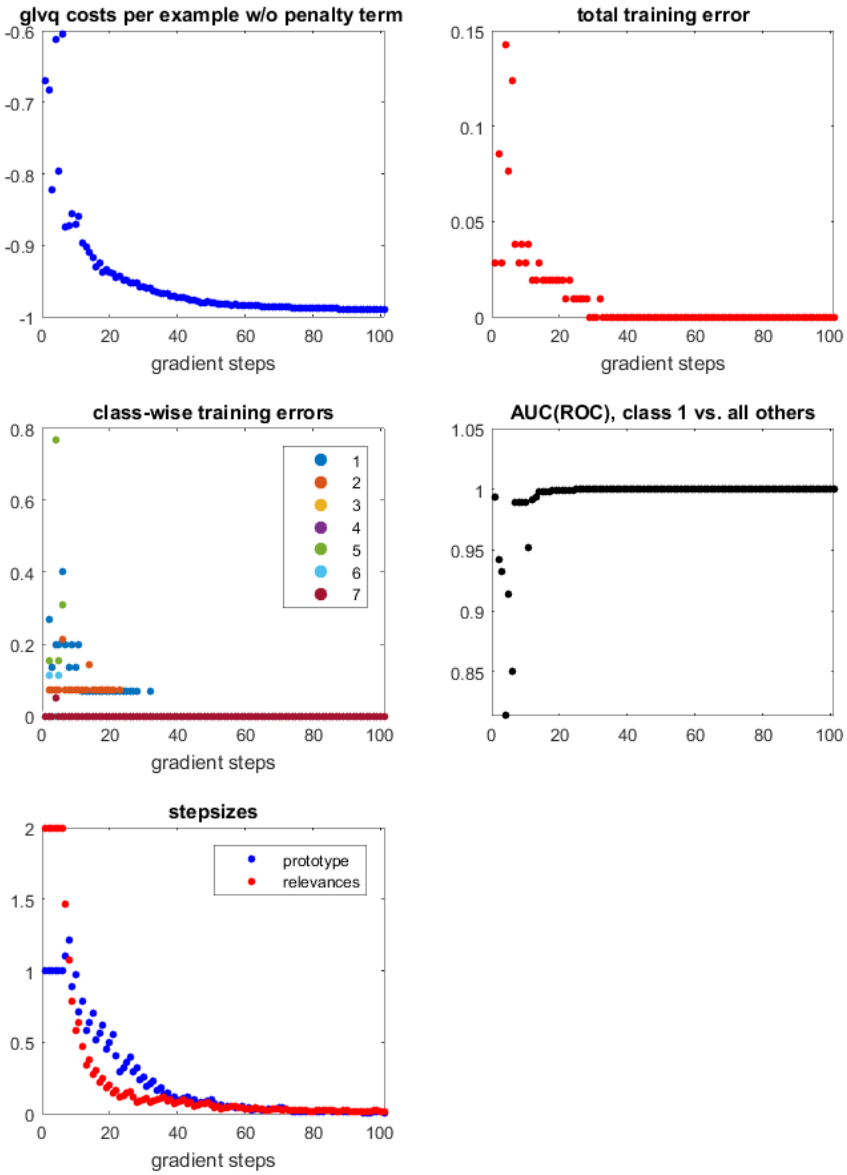


Figure A.1: Learning in original space.

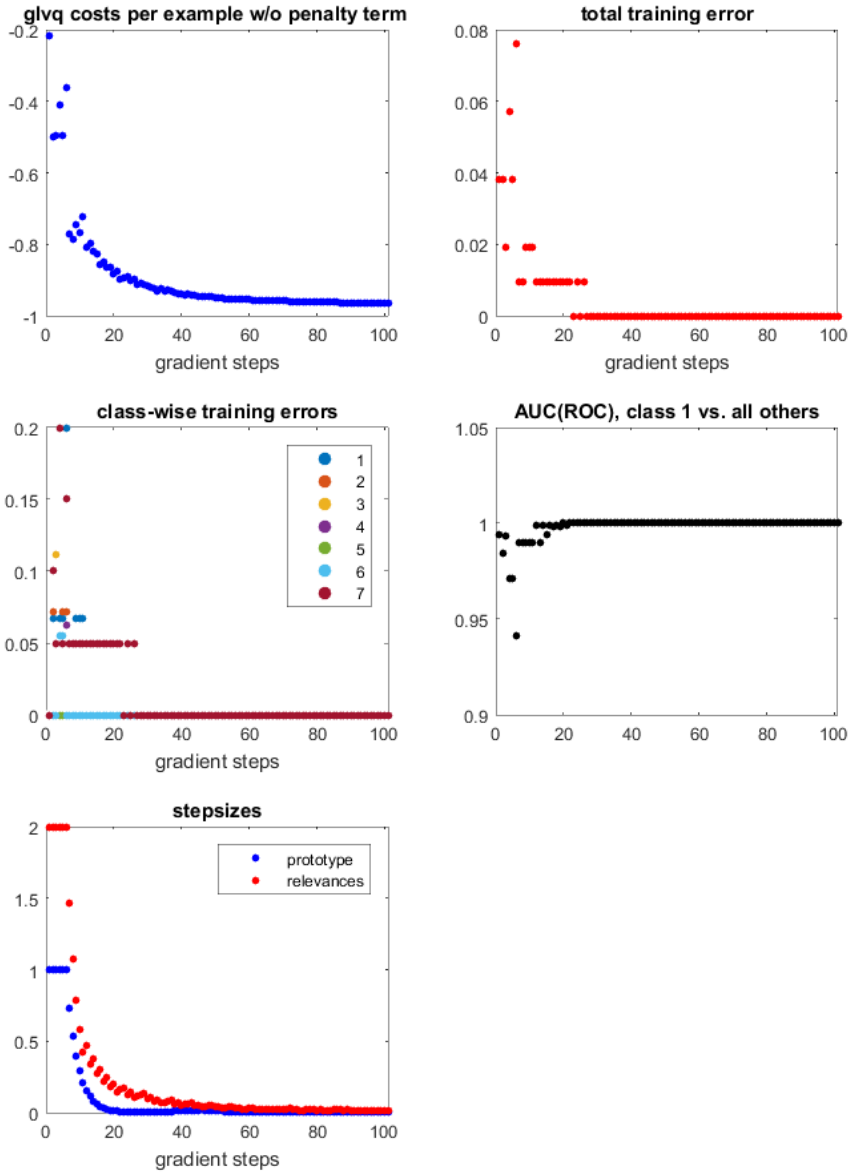


Figure A.2: Learning in Fourier space.

A.2 Mallat

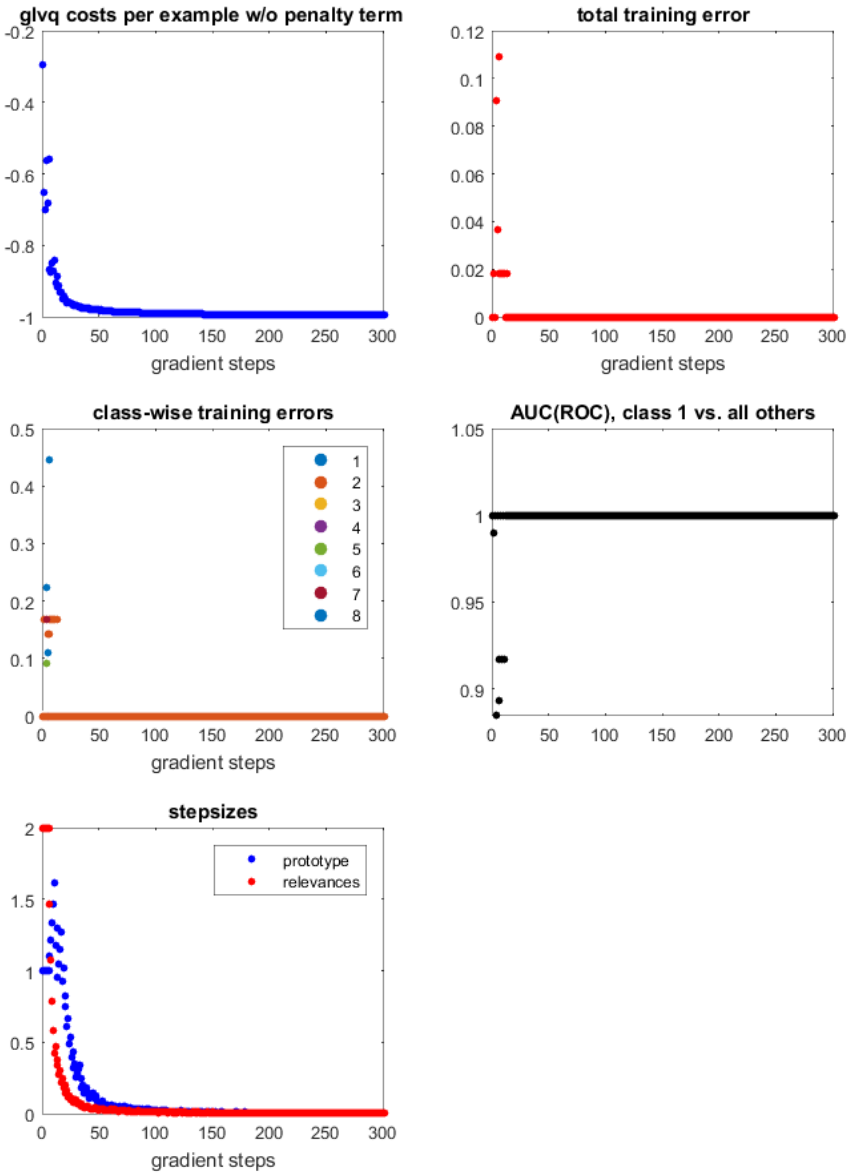


Figure A.3: Learning in original space.

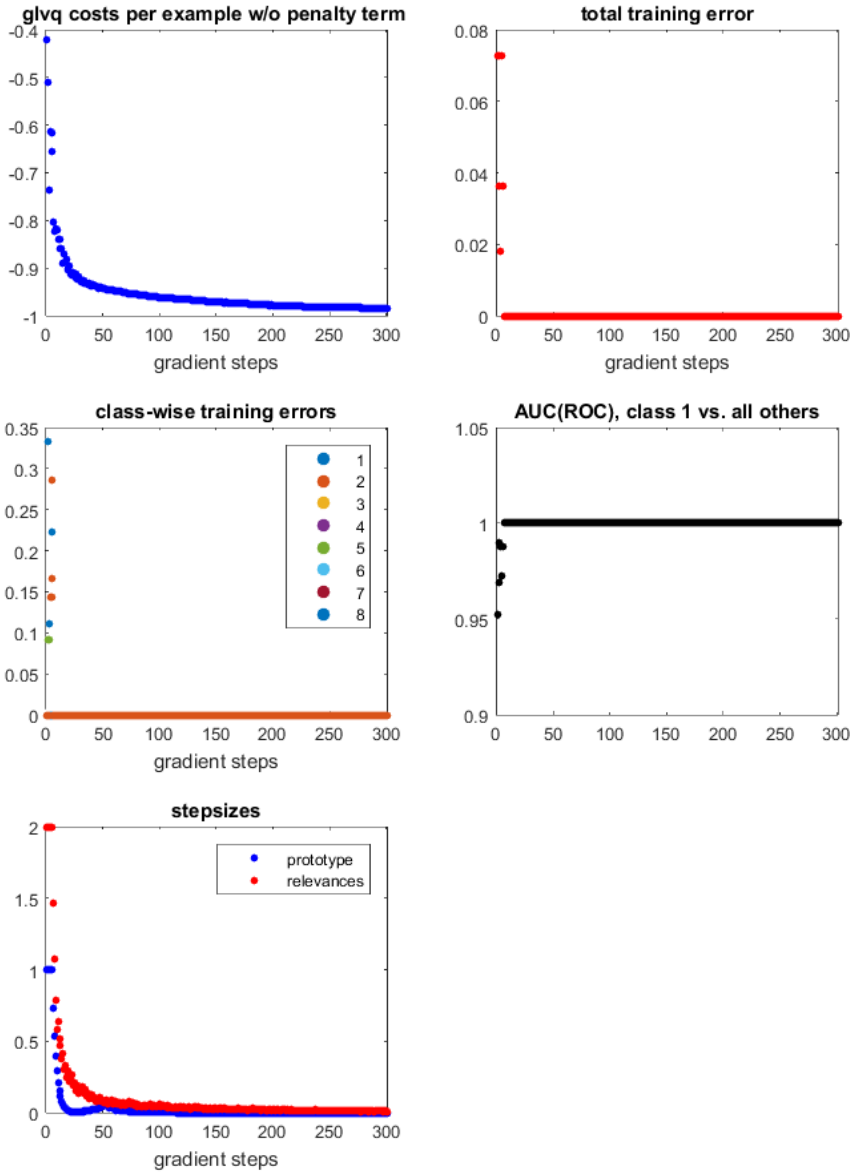


Figure A.4: Learning in Fourier space.

A.3 Symbols

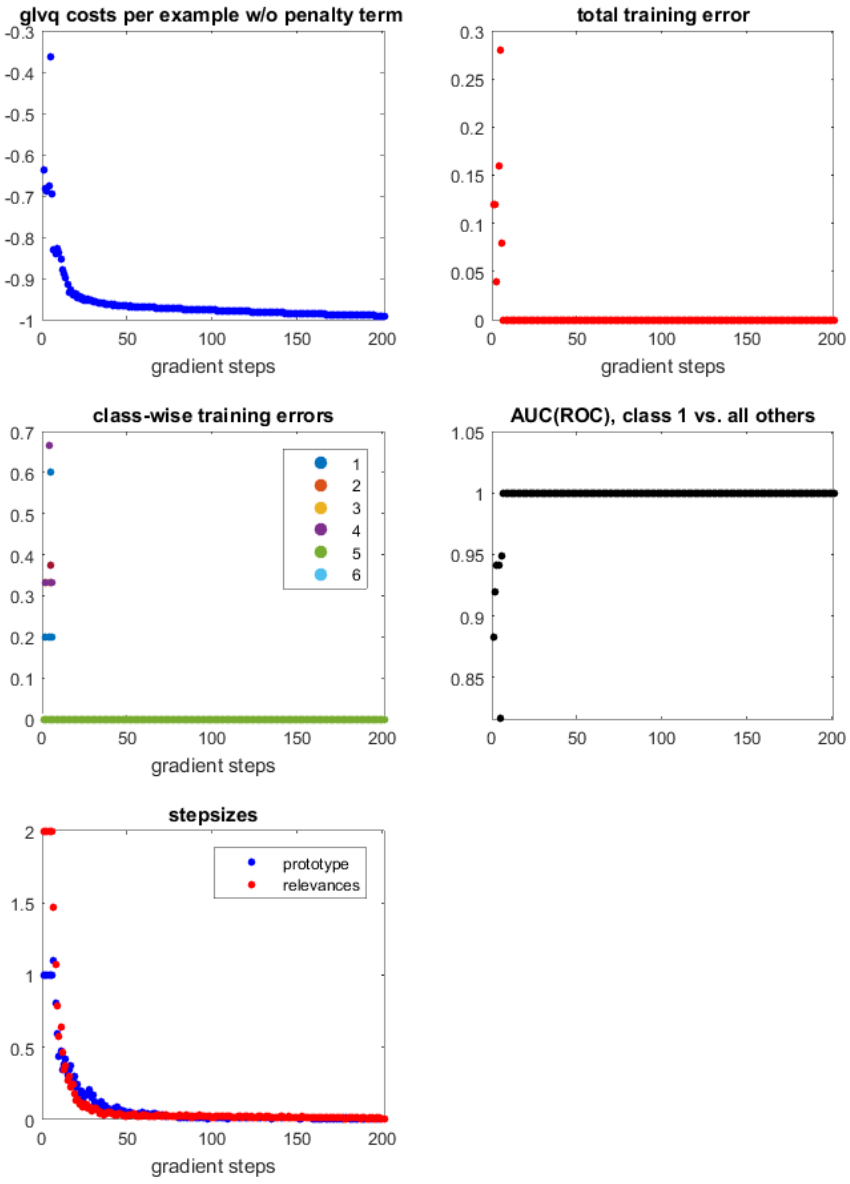


Figure A.5: Learning in original space.

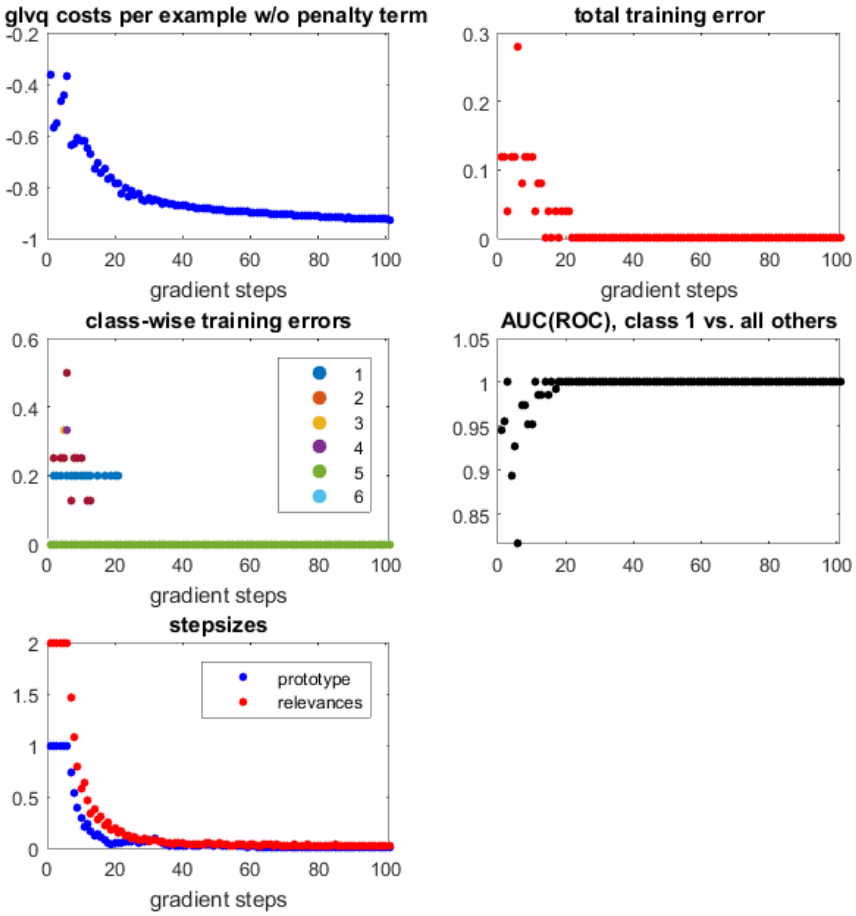


Figure A.6: Learning in Fourier space.

B Appendix: Functions

This document describes the most important functions that were created for the project. For the benchmark tests, the following folders need to be added to PATH in Matlab: benchmarks, display, fourier.

B.1 UCR dataset struct

st = LoadData(name)

Function loads a dataset with **name** from the UCR repository, provided that the repository (UCR_TS_Archive_2015) is available in the root of the project.

- **name**: The name of the UCR dataset.
- **st**: A struct representing the dataset.
  - **tra**: The training set.
  - **val**: The validation set.
  - **lbltra**: The labels of the examples in the training set.
  - **lblval**: The labels of the examples in the validation set.
  - **fvec**: The concatenation of training- and validation set (See function B.3).
  - **lbl**: The concatenation of the labels of the training- and validation set (See function B.3).

B.2 Fourier

Y = Fourier(x, r)

Wrapper around **fft** to obtain Fourier series of **x** at truncated at **r** coefficients. Ignores the symmetric part of the spectrum.

- **x**: Input vector (representing for example a time series).
- **r**: The desired number of coefficients to keep.
- **Y**: One-sided Fourier domain representation of **x**.

y = iFourier(X, L)

Wrapper around **ifft** to retrieve the original time domain signal **y**, given Fourier coefficients **X** and the length **L** of the original time domain signal.

- **X**: Vector of Fourier coefficients.
- **L**: The length of the original time series.
- **y**: The original time series.

st = takeFourierAt(st,r,type)

Takes a struct **st** as returned from **loadData(name)** (Function B.1) and attaches three additional structs to **st**:

- **FourierComp**: Complex Fourier representation (of training- and validation set) truncated at **r** coefficients.
- **FourierConc**: Concatenated Fourier representation (of training- and validation set) truncated at **r** coefficients.
- **back**: Original domain (of training- and validation set) after a truncation in Fourier space truncated at **r** coefficients.

Use **type** = 1 for an UCR struct as returned from the function B.1. Use **type** = 2 for a struct which does not have a prespecified training/validation split.

B.3 Benchmark

st = do_benchmark(st,epochs,plbl,maxF,name)

Function for benchmarking a dataset with a specified train/validation set. First learning in the original space is performed. Then learning in complex- and concatenated Fourier space is performed. Last, learning in truncated complex- and concatenated Fourier space is performed, and original space after a transform to truncated Fourier space and back transform to original space. The benchmark results, **benchout**, are attached to the struct

st. The graphical display of the results (AUC- and confusion matrices) are plotted and saved in the project root in the folder **name**.

- **st**: Struct as obtained from function B.1 to be benchmark tested.
- **epochs**: Number of epochs that should be performed in one test.
- **plbl**: The prototype labels.
- **maxF**: The maximum number of coefficients to be considered.
- **name**: The name of the folder to store the results in.

As an example, say we want to test the strategies on the **Plane** dataset from the UCR repository. First we load the dataset:

```
>> plane = loadData('Plane')

plane =

   lbltra: [105x1 double]
   lblval: [105x1 double]
     tra: [105x144 double]
     val: [105x144 double]
    fvec: [210x144 double]
     lbl: [210x1 double]
```

Then we run the `do_benchmark` function, indicating that all the results should be saved in the folder **Plane**:

```
>> plane = do_benchmark(plane,10,1:7,10,'Plane')

plane =

       lbltra: [105x1 double]
       lblval: [105x1 double]
         tra: [105x144 double]
         val: [105x144 double]
        fvec: [210x144 double]
         lbl: [210x1 double]
   FourierComp: [1x1 struct]
   FourierConc: [1x1 struct]
        back: [1x1 struct]
     benchout: [1x1 struct]
```

As can be seen, the struct now contains the two Fourier representations, the smoothed version and the results of the benchmark in **benchout**. All the figures are plotted by default, and stored in the chosen folder (in this case **Plane**).

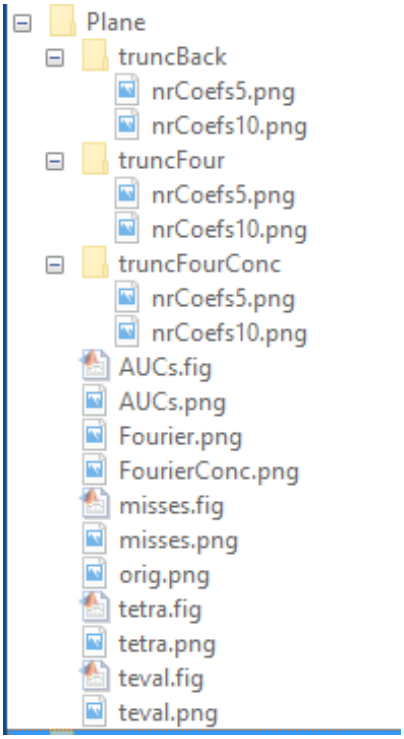


Figure B.1: Benchmark results including AUC curve that shows the AUC values for the different strategies, the confusion matrices as obtained in the different strategies and several error curves.

st = do_cross_benchmark(st,nruns,epochs,prctg,plbl,maxF,name)

Similar to the function **do_benchmark** (Function B.3). However, this function performs a cross validation of **nruns** using **prct**% of the examples for validation. The mean of the AUCs and the confusion matrices are plotted and saved in the project root in the folder **name**. In case the struct **st** represents a dataset from the UCR repository, the training- and validation set are simply concatenated and cross validation is performed on this concatenated set.

- **st**: Struct as obtained from function B.1 to be benchmark tested.
- **epochs**: Number of epochs that should be performed in one test.
- **plbl**: The prototype labels.
- **maxF**: The maximum number of coefficients to be considered.
- **name**: The name of the folder to store the results in.

C Appendix

Additional experiment on Wine dataset.

C.1 Wine

The dataset Wine from the UCR repository has 111 labeled feature vectors $(\mathbf{x}, y) \in \mathbb{R}^{243} \times (1, 2)$.

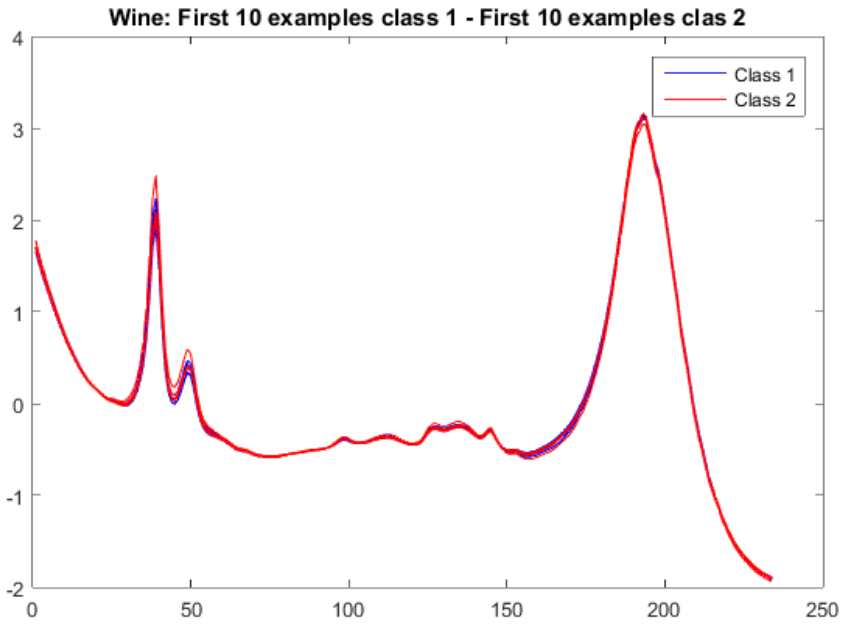
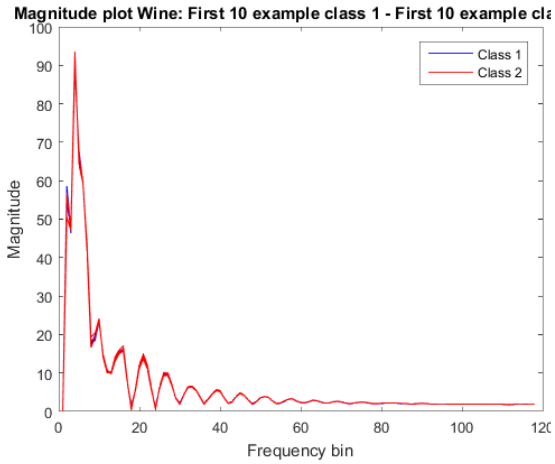
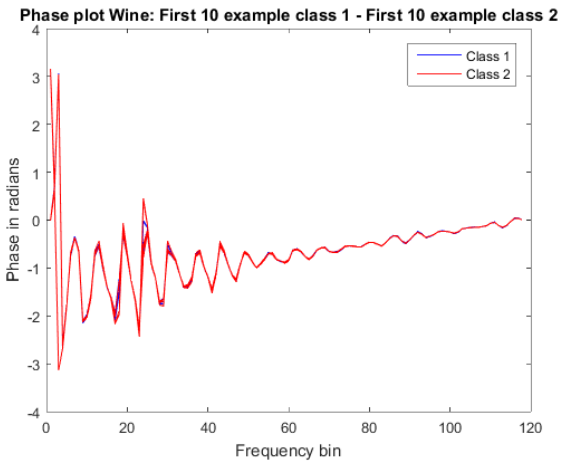


Figure C.1: 10 examples from each class of the Wine dataset in the original real-valued domain

The examples show small variations between the two classes in the original space and in Fourier space.

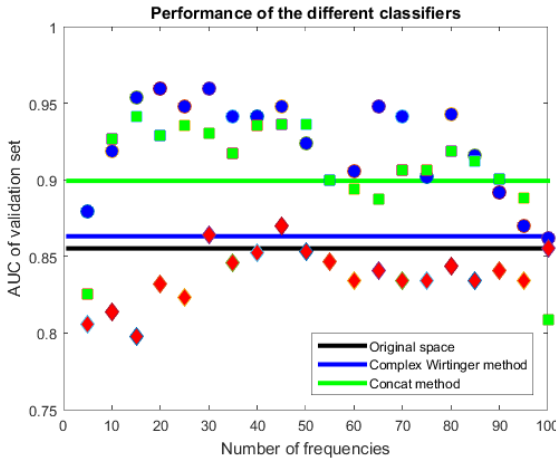


(a) Magnitude plots of first 10 example of each class of the Wine dataset

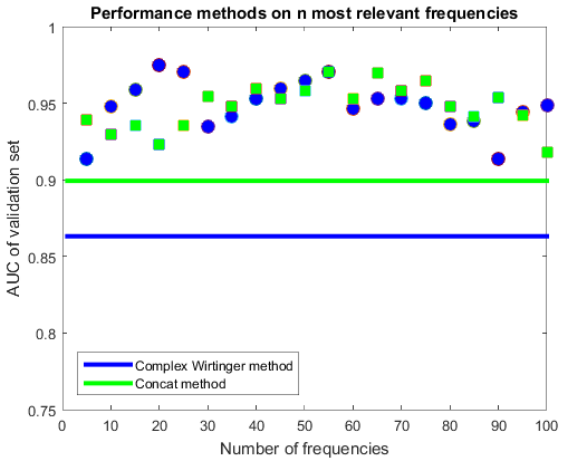


(b) Phase plots of first 10 example of each class of the Wine dataset

Figure C.2: Frequency spectra: 10 examples from each class



(a) In this approach the higher frequencies are gradually cut off.



(b) In this approach the n most relevant frequencies are kept.

Figure C.3: Benchmark results of the different strategies. The black solid line is the performance when the classifiers were trained in the original space. The results of the Wirtinger GMLVQ are in blue, with the solid line indicating the performance on all the Fourier coefficients. The results of the concatenation representation are in green, the green line indicates the performance on all the Fourier coefficients. The diamonds in red is the performance on the backtransformed data to the original space after smoothing/truncation in coefficient space.

In figure C.3 the benchmark results on the Wine dataset are shown. All the classifiers in the Fourier space show a better performance than than the classifiers in the original domain. From figure C.3a it appears that good classifiers are obtained when trained on 20 coefficients. In general, the performance of training on the complex coefficients and the performance of training on the concatenated real and imaginary parts seem to lie close to each other, but we observe that in the range [20...80] the training on the complex coefficients

performs slightly better. It also appears that there is a negative effect on performance the more higher frequency coefficients are in the spectra. The red diamonds show the third scenario in which the data is smoothed only. We see that in this case the AUC values do not exceed the AUC value from training in the original space. The improvement as seen in training in Fourier coefficient space can therefore not be explained as an effect of the smoothing only.

In figure C.3b GMLVQ is applied on the n most relevant coefficients. In this case we also observe good performance at 20 coefficients. No big performance loss is observed when considering more coefficients. There are some higher coefficients that are important for the classification and there are some middle-high coefficients that are not important.

Resulting diagonal of Lambda matrix after 400 epochs training on all Fourier coefficients (Wine dataset) (low to high frequencies)

0.0006	0.0662	0.0583	0.0036	0.0205	0.0310	0.0488	0.0139	0.0067	0.0873	0.0074	0.0002
0.0322	0.0146	0.0210	0.0288	0.0099	0.0075	0.0188	0.0158	0.0026	0.0008	0.0051	0.0258
0.0005	0.0089	0.0074	0.0030	0.0016	0.0122	0.0022	0.0089	0.0222	0.0026	0.0042	0.0006
0.0106	0.0066	0.0074	0.0039	0.0022	0.0020	0.0055	0.0055	0.0066	0.0025	0.0048	0.0220
0.0163	0.0020	0.0002	0.0027	0.0048	0.0056	0.0018	0.0005	0.0016	0.0002	0.0172	0.0133
0.0074	0.0110	0.0070	0.0142	0.0023	0.0094	0.0093	0.0211	0.0029	0.0070	0.0050	0.0012
0.0027	0.0019	0.0116	0.0001	0.0061	0.0007	0.0009	0.0017	0.0050	0.0014	0.0005	0.0022
0.0124	0.0011	0.0075	0.0018	0.0002	0.0024	0.0010	0.0040	0.0000	0.0043	0.0006	0.0037
0.0002	0.0001	0.0042	0.0008	0.0020	0.0021	0.0013	0.0042	0.0028	0.0017	0.0071	0.0080
0.0031	0.0034	0.0030	0.0047	0.0018	0.0009	0.0211	0.0032	0.0028	0.0024		

Figure C.4: The relevances of the Fourier coefficients as learned by GMLVQ after training on 400 epochs.

From figure C.4 we see that the fourth highest coefficient turns out to be quite relevant for the classification. This coefficient was not considered in the tests in figure C.3a, but it was considered in the tests in figure C.3b. It also turns out that some middle-high frequencies are of very low relevance: The 26th highest frequency has no relevance, and in general frequencies in that area are of low importance. This explains the results from figure C.3 since these frequencies were considered in figure C.3a, yielding a decrease in performance. Because of their low relevance, these frequencies were not considered in figure C.3b, this might explain the stable performance.

For this dataset, which does not seem to be a dataset which is particularly suitable for a Fourier representation, performance gains have been obtained. On 20-Fourier coefficients, the performance was better than the performance in the original space while reducing the dimensionality with 91%.

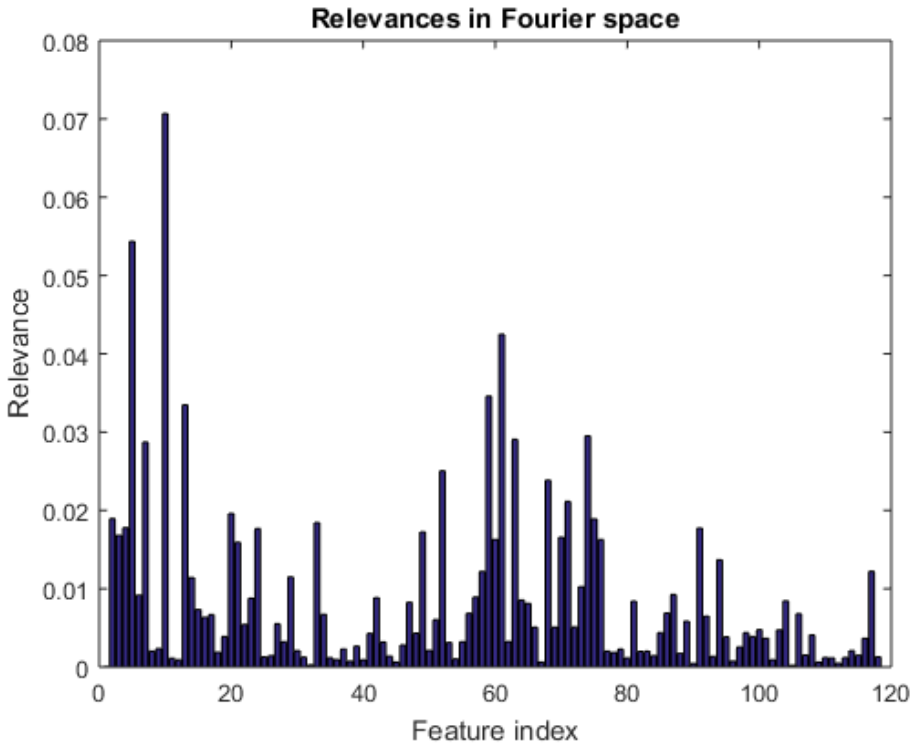
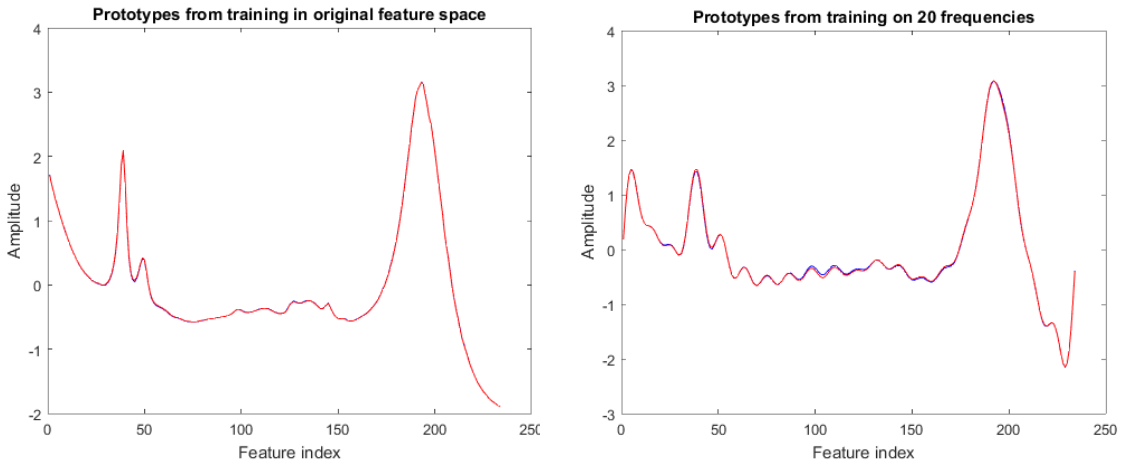


Figure C.5: The diagonal of Omega.



(a) Prototypes that resulted from training in the original space.

(b) Prototypes that resulted from training on 20 Fourier coefficients

Bibliography

[1] M. Biehl, *GMLVQ toolbox.* `http://www.cs.rug.nl/~biehl/gmlvq`, 2016. [Version 2.2].

[2] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, *The ucr time series classification archive*, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[3] T. Fawcett, *An introduction to ROC analysis*, Pattern Recogn. Lett., 27 (2006), pp. 861–874.

[4] M. Gay, *Sparsity and random projections in time-frequency-based communication systems, theory and algorithm design for the ofdm physical layer*, PhD thesis, University of Groningen, in press., (2016), pp. 139–145.

[5] M. Gay, M. Kaden, M. Biehl, A. Lampe, and T. Villmann, *Complex Variants of GLVQ Based on Wirtinger's Calculus*, in Advances in Self-Organizing Maps and Learning Vector Quantization, E. Merényi, M. Mendenhall, and P. O'Driscoll, eds., vol. 428 of Advances in Intelligent Systems and Computing, Springer, 1 2016, pp. 293–303.

[6] B. Hammer, M. Strickert, and T. Villmann, *On the Generalization Ability of GRLVQ Networks*, Neural Processing Letters, 21 (2005), pp. 109–120.

[7] F. Melchert, *Document not yet published. Section used on the backtransformation of relevances in coefficient space.*, 2016.

[8] F. Melchert, U. Seiffert, and M. Biehl, *Polynomial Approximation of Spectral Data in LVQ and Relevance Learning*, in Workshop on New Challenges in Neural Computation 2015, B. Hammer, T. Martinetz, and T. Villmann, eds., Machine Learning Reports, Bielefeld University, 10 2015, pp. 25–32. ISSN:1865-3960 http://www.techfak.uni-bielefeld.de/fschleif/mlr/mlr032015.pdf.

[9] F. Melchert, U. Seiffert, and M. Biehl, *Functional Representation of Prototypes in LVQ and Relevance Learning*, in Advances in Self-Organizing Maps and Learning Vector Quantization: Proceedings of the 11th International Workshop WSOM 2016, Houston, Texas, USA, January 6-8, 2016, E. Merényi, J. M. Mendenhall, and P. O'Driscoll, eds., Springer International Publishing, Cham, 2016, pp. 317–327.

[10] A. Sato and K. Yamada, *Generalized learning vector quantization*, in Advances in Neural Information Processing Systems 8, D. S. Touretzky and M. E. Hasselmo, eds., MIT Press, 1996, pp. 423–429.

[11] P. Schneider, M. Biehl, and B. Hammer, *Adaptive relevance matrices in learning vector quantization*, Neural computation, 21 (2009), pp. 3532–3561.

[12] J. O. Smith, *Mathematics of the Discrete Fourier Transform (DFT)*, W3K Publishing, http://www.w3k.org/books/, 2007.