university of groningen

faculty of mathematics and natural sciences

GPLVQ Prototype-based Probabilistic Classification

Master Thesis

June 2017

Student: M.C.P.M. Scheepens BSc

Primary supervisor: prof. dr. M. Biehl, University of Groningen

Secondary supervisor: dr. M.H.F. Wilkinson, University of Groningen

Contents

Chapter 1

Introduction

LVQ is a machine learning algorithm where data is represented by prototypes. GP is a statistical model which fits a distribution of functions around observed data points and is also seen as a machine learning algorithm. The goal of machine learning is to create models which can solve data problems and give meaningful output on data like class membership.

The most important aspect of LVQ is that it reduces the training data set to prototypes, where the distance to the prototypes determines the classification of novel data. GP on the other hand differs as it is probabilistic and does not rely on distance metrics. The relationship between the algorithms is shown in table 1.1.

It is hypothesized that a combination of LVQ with GP is possible and could introduce the concept of certainty to a given classification result. [1]

This thesis investigates how such a probabilistic prototype-based classifier can be created and how that classifiers performs for various datasets and in comparison to LVQ and GP.

Research questions

1. Can a mathematical framework be created which joins LVQ and GP?
2. (a) How does the newly created algorithm perform on various datasets?

   (b) How does the newly created algorithm perform in comparison with LVQ and GP?

	Non-probabilistic	Probabilistic
All data	Nearest neighbour	GP
Prototypes	LVQ	LVQ with GP

Table 1.1: Relationship of GPLVQ with already existing classifiers based on usage of data for classification

Chapter 2

Gaussian Processes

This chapter introduces the concept of learning with Gaussian processes.

2.1 Introduction

A Gaussian process is a statistical method which allows for supervised regression of data. Classification of data is also possible, but needs extra computations through approximations which are not Gaussian.

The Gaussian process is named after the Gaussian probability distribution, from which it is a generalization [2]. The Gaussian probability distribution is named after the German mathematician Carl Friedrich Gauß [3].

A Gaussian process describes a distribution over functions. Every point of a continuous input space is associated with a normally distributed random variable. The joint distribution of all these random variables is the distribution of the Gaussian process [2].

The theoretical foundation for Gaussian processes was already formulated in the 1940's with Wiener-Kolgomorov predictions and time series analysis [4]. In 1978 Gaussian processes were introduced to one-dimensional curve fitting by O'Hagan [4, 5]. Gaussian processes were popularized in machine learning in the 1990's by the upcoming of back-propagation in neural networks and the introduction of a Bayesian framework [6, 7]. Gaussian processes were developed independently in geo-spatial sciences in the 1950's with the name Kriging [8].

Gaussian processes as statistical methods have a direct relationship to Machine Learning algorithms. Neal showed that large Neural Networks converge eventually to Gaussian processes [7]. GP advocates sometimes claim that the equivalent model for a neural network as a Gaussian process would be easier to handle and interpret than the neural network [9], since the models are analytical tractable [10]. Furthermore, GPs perform well on binary classification problems and score results comparable to neural networks [6].

A major disadvantage of a Gaussian process is the computational effort. The implementation features inversions of matrices of the size $n \times n$ where n is the number of points where the Gaussian process is evaluated. Using standard linear algebra techniques, the inversion of this matrix has complexity $\mathcal{O}(n^3)$ [2].

Important real life applications of Gaussian processes are in geo-spatial sciences and in meteorology [8, 9].

2.2 Definition

"A Gaussian process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions" [2]. The Gaussian process is a generalization of the Gaussian distribution. The mean vector is replaced with a mean function and the covariance matrix is replaced with a covariance function. It is a stochastic process specified by its mean function m and covariance function k [11]:

$$m(\mathbf{x}) = \mathbb{E}\left[f(\mathbf{x})\right], \tag{2.1}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x} - m(\mathbf{x}))(f(\mathbf{x}' - m(\mathbf{x}'))\right], \tag{2.2}$$

where $\mathbf{x}$ is a input. The notation for a Gaussian process $\mathcal{GP}$ is

$$f \sim \mathcal{GP}\left(m, k\right), \tag{2.3}$$

where f is a function that is distributed as a Gaussian process where the Gaussian process is defined by the mean function m and the covariance function k [2].

2.2.1 Mean function

The mean function $m(\mathbf{x})$ (eq. 2.1) describes the mean value for every dimension. During preprocessing the input data is usually centred around the origin of the coordinate system. Because of this step in preprocessing, the mean function can be set to zero, or $m(\mathbf{x}) = \mathbf{0}$ throughout the rest of this thesis [12].

2.2.2 Covariance function

The covariance function describes the similarity of two data points as a scalar. The notion of similarity is essential in machine learning problems, since similar inputs usually yield similar outputs and the applied covariance function should reflect this [2, 11]. According to Rasmussen and Williams the covariance function "is the crucial ingredient in a Gaussian process predictor, as it encodes our assumptions about the function we wish to learn." [9].

Mathematically, the covariance function is defined on a pair of inputs and can be any positive semi-definite function $k(\mathbf{x}, \mathbf{x}')$ which satisfies [9]

$$\int k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d\mathbf{x} d\mathbf{x}' \geq 0 \text{ for all } f \in L_2(\mathbf{x}^{\mu}, \mathbf{x}). \tag{2.4}$$

The output is a scalar and one of its important properties is symmetry: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$.

A covariance function is stationary if it is a function of $\mathbf{x} - \mathbf{x}'$. A stationary covariance function is insensitive to movement of all inputs, thus it is invariant to translation of the data [6]. If the covariance function depends only on $|\mathbf{x} - \mathbf{x}'|$, it is called isotropic and is insensitive to all rigid motions on the input.

In literature, alternative names for the covariance function are *kernel* or *kernel function*. The squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-|\mathbf{x} - \mathbf{x}'|^2}{2\alpha^2}\right) \tag{2.5}$$

with α as length scale is a covariance function that satisfies the previous properties. The squared exponential covariance function is a widely used kernel in the machine learning field and is useful when the unknown function that GP tries to model is smooth [9]. The hyperparameter α is a length scale which indicates when the two inputs of the covariance function become uncorrelated [12].

2.2.3 Example

The definition of a Gaussian process may not be understood easily, therefore an example may help. A simple Gaussian process is defined with:

$$f \sim \mathcal{GP}(m, k), \text{ where } m(\mathbf{x}) = \frac{1}{4}\mathbf{x}^2 \text{ and } k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^2). \tag{2.6}$$

Now samples can be drawn from the function f at any number of locations n. Given the $\mathbf{x}$, the Gaussian process can now be evaluated using the previous formula which results in the following Gaussian distribution:

$$\mu_i = m(x_i) = \frac{1}{4}\mathbf{x}_i^2, \; i = 1, \ldots, n \tag{2.7}$$

$$\Sigma_{ij} = k(x_i, x_j) = \exp\left(-\frac{1}{2}(x_i - x_j)^2\right), \; i, j = 1, \ldots, n, \tag{2.8}$$

where μ denotes the mean rather than the mean function m and Σ the covariance rather than the covariance function k. It is possible to retrieve random vectors $\mathbf{f}$ from this distribution according to $f(x)$:

$$\mathbf{f} \sim \mathcal{N}(\mu, \Sigma) \tag{2.9}$$

In figure 2.1 a visualization is presented where three random samples for equation 2.9 are generated that satisfy equation 2.6. The presented functions are discrete, since the function were only evaluated for a finite number n.

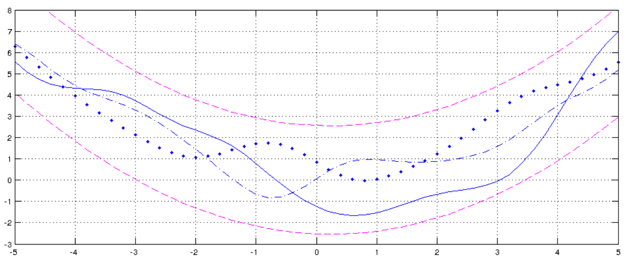


Figure 2.1: Three randomly drawn functions from equation 2.9 which satisfy equation 2.6. The pink dashed lines represent the 95% confidence interval of the Gaussian process.

2.3 Training

A Gaussian process is a supervised learning algorithm. Many machine learning algorithms build their classification model by iterating over the input and comparing the calculated training results to the training data. A Gaussian process on the other hand creates its model by fitting a Bayesian statistical model to the training data.

2.3.1 Prior

The prior describes all functions which satisfy equation 2.3 for the specified mean function, the covariance function and the hyperparameters.

If we assume, for example, a second order mean function $m(x) = ax^2 + bx + c$ and the squared exponential kernel (eq. 2.5), then the function space from equation 2.3 is made up of functions that resemble second order polynomials. The functions also contain some Gaussian noise and an added smooth function from the covariance function [2].

Two additional features are introduced. The *Gram matrix* or covariance matrix K is the composition of all covariance values for all inputs $\mathbf{x}_{1:n}$: $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ The observation noise σ_n^2 is a hyperparameter, it encodes the noise in the training data. The noise is assumed to be independent for every measurement and that it can be described with Gaussian noise. With these assumptions, the covariance k is slightly modified. With the new Gram matrix notation, the observation noise can be added easily: The Gram matrix K is replaced with $K + \sigma_n^2 I$.

The addition of observation noise is important. If the noise is $\sigma_n^2 = 0$, the Gaussian process will yield a mean predictive function which matches all training inputs exactly [2]. This is considered an example of overfitting the input data.

2.3.2 Posterior

The prior does not encode training data, it specifies properties of the function to be learned. The posterior takes the training input into account and will be used to make predictions. Consider the set of training values $\mathbf{f}$ and the set of test values $\mathbf{f}_*$. These have a joint Gaussian process:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu_*} \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^\top & \Sigma_{**} \end{bmatrix} \right), \tag{2.10}$$

with the training means $\boldsymbol{\mu} = m(x_i)$ for all $i = 1, \dots, n$ and $\boldsymbol{\mu_*}$ for the test means. Σ denotes the training set covariances, Σ_* the training-test set covariances and $\Sigma_* *$ the test set covariances.

The previous equation can be written as a conditional distribution of $\mathbf{f}_*$ given $\mathbf{f}$, for which $\mathbf{f}$ is already known:

$$\mathbf{f}_* | \mathbf{f} \sim \mathcal{N}\left(\boldsymbol{\mu}_* + \Sigma_*^\top \Sigma^{-1} (\mathbf{f} - \boldsymbol{\mu}), \Sigma_{**} - \Sigma_*^\top \Sigma^{-1} \Sigma_* \right). \tag{2.11}$$

From this distribution the following generalization for the Gaussian process framework can be made:

$$\begin{aligned} f|\mathcal{D} \sim \mathcal{GP}(m_\mathcal{D}, k_\mathcal{D}), \\ m_\mathcal{D}(x) = m(x) + \Sigma(X, x)^\top \Sigma^{-1} (\mathbf{f} - \mathbf{m}) \\ k_\mathcal{D}(x, x') = k(x, x') - \Sigma(X, x)^\top \Sigma^{-1} \Sigma(X, x') \end{aligned} \tag{2.12}$$

2.3.3 Marginal likelihood

In order to train the Gaussian process, the hyperparameters can be optimized. The probability of the data is calculated for the hyperparameters in the *marginal likelihood*:

$$L = \log p(\mathbf{x}|\mathbf{y}, \theta) = -\frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} (\mathbf{y} - \mu) \top \mathbf{K}^{-1} (\mathbf{y} - \mu) - \frac{n}{2} \log(2\pi), \tag{2.13}$$

where θ denotes the set of hyperparameters. By maximizing the gradient of the marginal likelihood the system is trained. The best solution between overfitting and underfitting is determined without any input or heuristics [2].

The posterior is the distribution of functions that is obtained after optimization of the marginal likelihood, therefore the posterior encodes the input data into the model.

2.3.4 Regression

The basic use case of Gaussian processes is to perform regression on a given dataset. With regression the aim is to give a numerical estimate for a new data point. This estimate is computed from the predictive distribution [9]:

$$p(y_x|\mathbf{x}_*, \mathbf{x}_{1:N}, y_{1:N}) \sim \mathcal{N}(\bar{y}_*, \mathbb{V}[y_*]), \tag{2.14}$$

where the mean function and variance function are

$$\bar{y}_* = \mathbf{k}_*^\top(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} \tag{2.15}$$

$$\mathbb{V}[y_*] = 1 - \mathbf{k}_*^\top(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*. \tag{2.16}$$

2.3.5 Classification

Gaussian processes are not fully suitable for classification in their standard set-up [4, 10, 13]. The goal of a Gaussian process classifier is to assign an estimate of the probability for class membership $P(y|\mathbf{x})$, where y is the class. It is necessary to add an activation function that squeezes the output between zero and one, e.g. a logistic function:

$$P(y|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}. \tag{2.17}$$

When the activation function is incorporated into the prior, so the outcome represents probabilities, it becomes impossible to solve the prior analytically. It is necessary to use Monte Carlo functions or a approximation like Laplacians approximation in order to make an approximation for the prediction [13, 14].

Chapter 3

LVQ

Learning Vector Quantization (LVQ) is a supervised machine learning technique that is used for classification. LVQ models the training data with prototypes, where the prototypes are a representation of the underlying data. In order to model data with greater variance, i.e. data with multiple clusters, more prototypes for a class can be introduced. The use of prototypes allows easily for classification of multi-class problems. [15] LVQ – as opposed to many classification algorithms – allows for straightforward interpretation of these prototypes. This facilitates discussions with domain experts when presenting the obtained classification framework.

LVQ1 is the first version of LVQ which was introduced by Kohonen in 1986 [16]. This is the simplest variant of the LVQ family. LVQ1 uses labeled data $\{\mathbf{x}^\mu, y^\mu\}_{\mu=1}^M$, with feature vectors $\mathbf{x}^\mu \in \mathbb{R}^N$ and labels $y^\mu \in \{1, 2, \ldots C\}$. The LVQ system itself is defined by the prototypes $W = \{\mathbf{w}^j, c(\mathbf{w}^j)\}_{j=1}^M$, where $\mathbf{w}^j \in \mathbb{R}^N$ and $c(\mathbf{w}^j) \in \{1, 2, \ldots C\}$, and the distance measure $d(\mathbf{x}, \mathbf{w})$. This distance measure usually employs the Euclidean distance or another form of the Minkowski distance.

The training of LVQ is fairly simple. At first the prototypes are initialized, often by assigning prototypes to some random data points or by placing the prototypes at the mean of a class. The next step is to train the system by presenting single examples. During the training the prototype with the smallest distance to the sample is updated. The update depends on the class-membership: if the class of the sample matches the class of the prototype, the prototype moves towards the sample, otherwise the prototype moves in the opposite direction [17, 18].

The learning procedure is:

1. Randomly select a training sample $(\mathbf{x}, y)$
2. Determine the winning prototype $\mathbf{w}^i$ with distance $d(\mathbf{w}^i, \mathbf{x}) = \min_j(d(\mathbf{w}^j, \mathbf{x}))$

3. Update $\mathbf{w}^i$:

$$\mathbf{w}^i \leftarrow \mathbf{w}^i + \alpha \cdot \eta \cdot (\mathbf{x} - \mathbf{w}^i) \tag{3.1}$$

   where the learning rate η controls the step size of the update and
   $\alpha = \begin{cases} +1 \text{ if classes of } \mathbf{w}^i \text{ and } (\mathbf{x}, y) \text{ match} \\ -1 \text{ if classes do not match} \end{cases}.$

After learning, a set of prototypes with class labels is available for classification. For a new datapoint the distances to all prototypes are calculated, and is followed by an assignment to the class of the nearest prototype.

Various extensions to the LVQ algorithm exist which are briefly introduced now. The simplest extension, **LVQ2**, updates the nearest correct prototype when the class of the nearest prototype did not match, so more information about the data point is added to the model. More complex extensions introduce a cost function to the update or modify the distance metric.

Generalized Learning Vector Quantization (**GLVQ**), which was proposed by Sato and Yamada, features a cost function [19]. The cost function is based on the distances to the closest correct and incorrect prototype. The relative distance from the data point to these two prototypes is the cost function:

$$E_{GLVQ} = \sum_{i=1}^{P} \Phi(\mu_i), \mu_i = \frac{d_J(\mathbf{x}^\mu) - d_K(\mathbf{x}^\mu)}{d_J(\mathbf{x}^\mu) + d_K(\mathbf{x}^\mu)} \tag{3.2}$$

where the index J denotes the closest correct prototype and K denotes the closest incorrect prototype. $d_j(\mathbf{x})$ is short notation for the squared Euclidian distance $d_j(\mathbf{x}) = d(\mathbf{w}_j, \mathbf{x}) = (\mathbf{w}_j - \mathbf{x})^2$.
In the cost function Φ denotes a sigmoidal or linear activation function and μ_i is a ratio in the range $[-1, 1]$. The ratio is negative if the closest prototype is from the same class as the data point and positive if the data point and the closest prototype are from different classes. Learning is achieved by minimization of E_{GLVQ}. It can achieved by various techniques, for example stochastic gradient descent [20].

Generalized Matrix Learning Vector Quantization (**GMLVQ**) is an extension of GLVQ where the Euclidian distance measure is replaced by a generalized distance measure [17]. The algorithm can now assign different weights to the input dimensions. This allows it to filter noisy or non-relevant inputs and emphasize input dimensions which contribute meaningful to the classification.

The new distance measure is:

$$d^\Lambda(\mathbf{x}, \mathbf{w}) = (\mathbf{x} - \mathbf{w})^\top \Lambda (\mathbf{x} - \mathbf{w}) \tag{3.3}$$

with Λ being a matrix of dimension $\mathbb{R}^{n \times n}$. The matrix can be seen as a measure for scaling and dependencies of features [21]. Furthermore Λ can be represented with

$$\Lambda = \Omega^\top \Omega \tag{3.4}$$

where $\Omega \in \mathbb{R}^{m \times n}$ and $m \leq n$. This substitution of Λ makes it becomes symmetric and positive semi-definite, which are necessary conditions for a valid Euclidean pseudo-metric. The matrix Ω also leads to a rewritten form of the distance measure (formula 3.3):

$$d^{\Lambda}(\mathbf{x}, \mathbf{w}) = [(\mathbf{x} - \mathbf{w})^{\top} \Omega^{\top}][\Omega(\mathbf{x} - \mathbf{w})] = [\Omega(\mathbf{x} - \mathbf{w})]^2. \tag{3.5}$$

Choosing $m < n$ allows the user to restrict the number of dimensions, i.e. linear combinations of the original features used in the classifier.

Since GMLVQ is an extension of GLVQ, the prototypes $\mathbf{w}$ and Ω can likewise be adapted to a given data set by optimizing the cost function E_{GMLVQ} for a given set of examples, e.g. by means of gradient descent techniques.

Chapter 4

GPLVQ

4.1 Introduction

In the previous chapters the classification techniques Gaussian Processes and Learning Vector Quantization were introduced. In this chapter these two techniques are combined into a new machine learning technique called Gaussian Process Learning Vector Quantization (GPLVQ). The update of prototypes in LVQ is replaced by a gradient on the likelihood and the distance measure is replaced with the covariance function [1].

Since GP is in its normal set-up a two-class algorithm, GPLVQ is initially not suited for multi-class problems. From LVQ the new algorithm inherits *underfitting*. If a class has multiple clusters, but not enough prototypes for representing these distinct clusters, a prototype from a neighbouring cluster may lead to incorrect classification.

4.2 Theory

LVQ uses prototypes to represent the training inputs. Classification of an input is decided by the smallest Euclidean distance between the test data point and the prototypes. GP uses all training inputs to model a normal distribution of functions. Classification is decided through a likelihood function on those functions for that given point.

The purpose of this thesis is to show whether merging these two algorithms creates a new algorithm that couples predictions to prototypes. For the combination the two algorithm, the update of the prototypes in LVQ is changed from a Minkowski metric to the gradient of the likelihood of GP. The prototype will be moved along the gradient during the update towards the data point if the labels agree and is repelled if the labels disagree. The model for the Gaussian Process is based on the prototypes from LVQ instead of the initial data points.

4.2.1 Training

The GPLVQ algorithm has two stages during its training phase. At first the LVQ prototypes are initialized and trained, then the GP is trained based on those prototypes.

The update function of LVQ features a Minkowski distance metric for calculation of the update steps [18]. Instead of the regular metric, GPLVQ will use the gradient of the likelihood function to determine the update. This update function reflects the use of the Gaussian Process.

When the label c of the nearest prototype $\mathbf{w}$ and the class label y of data point $\mathbf{x}$ agree, the prototype is moved along the gradient towards the prototype. If the labels are different, the prototype is moved in the opposite direction.

The update function assumes the use of a squared exponential kernel. The update function of GPLVQ is derived in appendix B fully, a brief overview is given here.

Update function

The first step is to evaluate the probability function for the data class given the position of the point and the nearest prototype with its label (see equation 2.22 from [9]):

$$p(y|\mathbf{x}, \mathbf{w}_{1:N}, c_{1:N}) \sim \mathcal{N}(\bar{c}, \mathbb{V}[y]). \tag{4.1}$$

The short notation is expanded to a long notation of the Gaussian distribution:

$$p(y|\mathbf{x}, \mathbf{w}_{1:N}, c_{1:N}) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\intercal \Sigma^{-1} (\mathbf{x}-\mu)\right). \tag{4.2}$$

The mean of the distribution μ is replaced with the single prototype $\mathbf{w}$ which needs to be updated:

$$p(y|\mathbf{x}, \mathbf{w}_{1:N}, c_{1:N}) = \frac{1}{\sqrt{(2\pi)^N |\mathbb{V}[c]|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{w})^\intercal \mathbb{V}[c]^{-1} (\mathbf{x}-\mathbf{w})\right). \tag{4.3}$$

The single prototype is compared with just one data point:

$$p(y|\mathbf{x}, \mathbf{w}, c) = \frac{1}{\sqrt{(2\pi)|\mathbb{V}[c]|}} \exp\left(-\frac{1}{2\mathbb{V}[c]}(\mathbf{x}-\mathbf{w})^\intercal (\mathbf{x}-\mathbf{w})\right). \tag{4.4}$$

	Non-probabilistic	Probabilistic
All data	Nearest neighbour	GP
Prototypes	LVQ	GPLVQ

Table 4.1: Relationship of GPLVQ with already existing classifiers based on usage of data for classification

In order to use this probability function in the update function of LVQ, which shows the direction and the magnitude of the update, the partial derivative with respect to the prototype is taken.

The gradient between a data point $\mathbf{x}$ and a prototype $\mathbf{w}$ is described by:

$$\nabla_{\mathbf{w}} p(\mathbf{x}, y|\mathbf{w}, c) = \frac{-2c(\mathbf{x}-\mathbf{w})k(\mathbf{x},\mathbf{w})\left(c\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2} - y\right)}{\alpha^2(1+\sigma_n^2)\left(1-\frac{k(\mathbf{x},\mathbf{w})^2}{1+\sigma_n^2}\right)} - \frac{(\mathbf{x}-\mathbf{w})k(\mathbf{x},\mathbf{w})^2\left(c\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2} - y\right)^2}{\alpha^2(1+\sigma_n^2)\left(1-\frac{k(\mathbf{x},\mathbf{w})^2}{1+\sigma_n^2}\right)^2} \tag{4.5}$$

$$= \frac{-2c(\mathbf{x}-\mathbf{w})k(\mathbf{x},\mathbf{w})\left(c\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2} - y\right)}{\alpha^2(1+\sigma_n^2)\left(\mathbb{V}[c]\right)} - \frac{(\mathbf{x}-\mathbf{w})k(\mathbf{x},\mathbf{w})^2\left(c\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2} - y\right)^2}{\alpha^2(1+\sigma_n^2)\left(\mathbb{V}[c]\right)^2} \tag{4.6}$$

The update function for GPLVQ is:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta \nabla_{\mathbf{w}} p(\mathbf{x}, y|\mathbf{w}, c). \tag{4.7}$$

The training of the GPLVQ network is started similarly to LVQ. A set of prototypes is initiated on the data. Since GP needs a sufficient number of inputs, it is recommended to initialize with several prototypes per class. The next step is running the newly obtained update function for this algorithm over the prototypes. In this thesis the prototypes are assigned randomly to existing data points. Other strategies for the initialization could be employed as well, but are not investigated.

4.2.2 Classification

The classification is eventually performed in the same way as with standard GP. The difference is that the prototypes are the training input for the algorithm instead of all available training points. Since GP is only capable of classifying two-class problems, this restriction also holds for GPLVQ. A strategy for classification of multi-class problems with k classes is by performing k instances of one-versus-rest classification.

Chapter 5

Experiments

5.1 Introduction

This chapter explores the results when GPLVQ is employed to classify simple and more advanced datasets. Furthermore the performance of LVQ and GP for the same datasets is measured in order to compare GPLVQ with its ancestor algorithms.

5.2 Artificial data

In this first experiment the classifier is applied to an artificial dataset to demonstrate its performance in a well-observed situation [1]. Furthermore, the effect of varying the hyperparameters, the scaling parameter α and the observation noise σ_n^2, is observed. The dataset comprises of two classes in a two dimensional space. 50 data points per class are generated around two intertwined spirals and are thus not linearly separable (see figure 5.1).

The algorithms ran with 10-fold cross-validation. For LVQ, six prototypes were assigned randomly to existing data points. 250 epochs with a learning rate of $\eta = 0.01$ was used. For GPLVQ the same number and procedure for the prototypes was used as with LVQ. GP was executed with identical hyperparameter α as for GPLVQ. The hyperparameter settings are discussed below.

The scaling parameter α has significant effect on classification. Results are presented in table 5.1. The error is high for a very low α, $\alpha = 0$ is mathematically not allowed. The error decreases quickly and has a minimum around $\epsilon \approx 0.12$ for $0.4 < \alpha < 3$. For greater α the error increases.

The observation noise parameter σ_n^2 has a smaller effect on classification than α. The classification results are shown in table 5.2. If the observation noise is set to zero, the

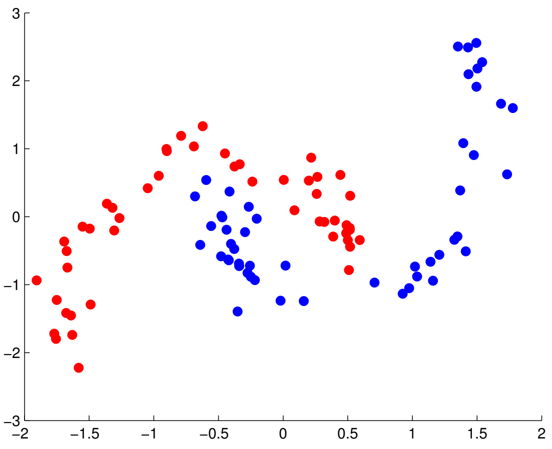


Figure 5.1: A plot of for the artificially created dataset with two spirals. The spirals are centred around the origin and are not linearly separable.

test error is at a maximum with 27%. When σ_n^2 is increased slightly to 0.01, the error has already dropped to 16% and it will continue to fall until it reaches its minimum at 0.5%. For higher values of σ_n^2, the error increases slightly and reaches a plateau at $\epsilon \approx 14\%$.

α	0.01	0.1	0.2	0.4	0.6	1.0	1.5	2.0	3.0	4.0	4.5	5.0
ϵ_{test}		0.23	0.17	0.11	0.12	0.12	0.11	0.12	0.12	0.15	0.18	0.26

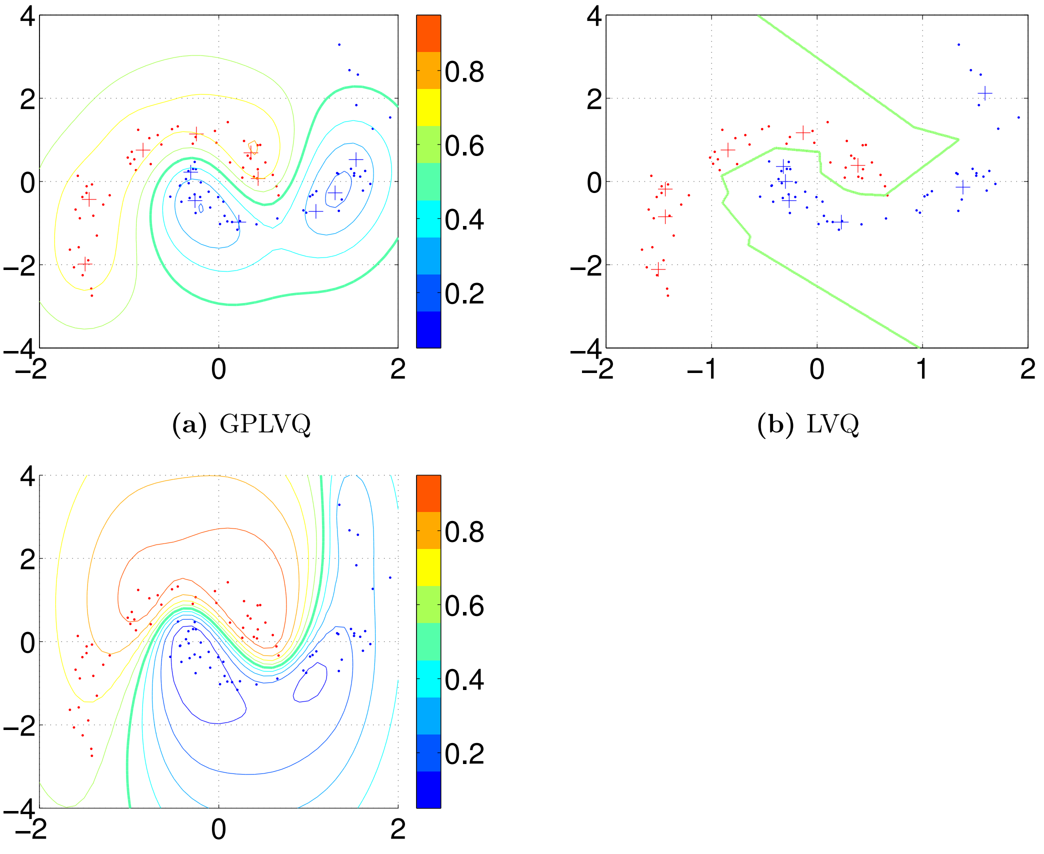
Table 5.1: Mean rate of misclassification for different values of the hyperparameter α on the artificial spirals dataset.

σ_n^2	0.0	0.01	0.1	0.2	0.3	0.5	1.0	2.0	5.0	10.0
ϵ_{test}	0.27	0.16	0.14	0.13	0.13	0.12	0.13	0.14	0.14	0.14

Table 5.2: Mean rate of misclassification for different values of the hyperparameter σ_n^2 (observation noise) on the artificial spirals dataset.

All three algorithms feature a decision boundary. For LVQ the decision boundary is at half the Euclidean distance between differing prototypes. Since GP and GPLVQ do not use distances but probabilities for the classification, a decision boundary has to be chosen. The decision boundary should represent the points were the probability that an input belongs to either of the two classes changes. These points are where $p_{A|x} = 1 - p_{B|x} = 0.5$ holds.

Examples for these decision boundaries are shown in figure 5.2. The decision boundary of LVQ is not smooth because of the distance metric. The decision boundary is basically a subset of the boundaries shown in a Voronoi diagram where the lines between classes are excluded. The decision boundaries of GP and GPLVQ on the other hand are smooth.



(a) GPLVQ

(b) LVQ

(c) GP

Figure 5.2: Prototypes and decision boundary for GPLVQ, LVQ and GP for identical data. The decision boundary for GP and GPLVQ is at $p(A|x) = 0.5$.

The decision boundary is derived from calculations of the covariance function. The covariance function features high order derivatives which makes the boundary between the two classes smooth. All three decision boundaries resemble a spiral which can be drawn between the two intertwined spirals.

There are some notable differences in classification results between GP and GPLVQ. The isolines for different p-values for GP and GPLVQ show that GPLVQ classifies with the same p-value in a smaller area. The comparison of the area where the variance is equal at σ_n^2=0.05 (see figure 5.3) also that this area is smaller for GPLVQ than for GP. This difference is caused by the smaller number of data points (prototypes) that are used in GPLVQ's GP than in the original GP. In figure 5.4 the behaviour for a large number of prototypes in GPLVQ is compared to GP. With 25 prototypes on a dataset of 45 training points the difference between the isolines of GP and GPLVQ becomes smaller. When 45 prototypes are chosen the difference in classification between the two methods becomes minimally.

This behaviour of GPLVQ for a very high number of prototypes is explainable: GPLVQ now mirrors GP. Many data points are replaced by a prototype, thus a prototype may represent one or two data points. GP is then executed on a set of prototypes which does not differ much from the initial dataset.

	ϵ_{train}	ϵ_{test}
LVQ	0.01	0.03
GP	0.04	0.01
GPLVQ	0.03	0.05

Table 5.3: Comparison of the mean misclassification rate for LVQ, GP and GPLVQ for the artificial spirals dataset.

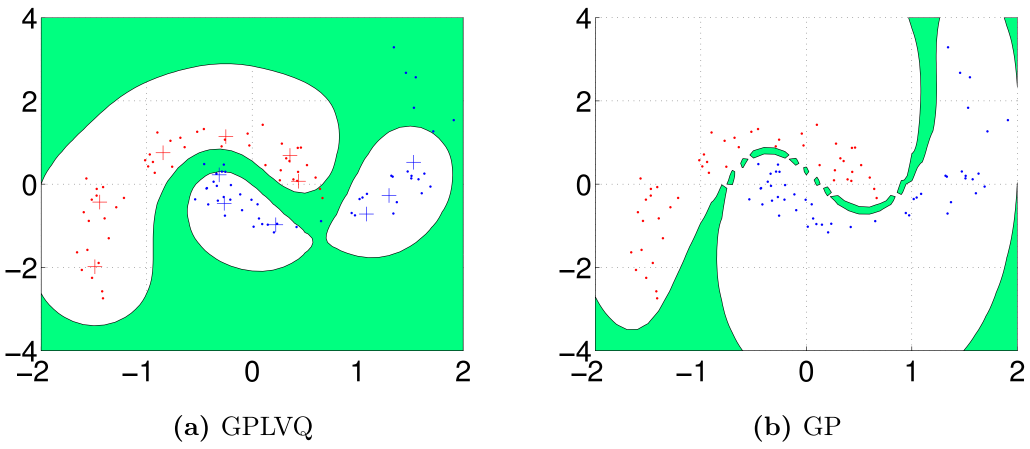


Figure 5.3: Area where the covariance is $= 0.05$ for GPLVQ and GP for identical data.

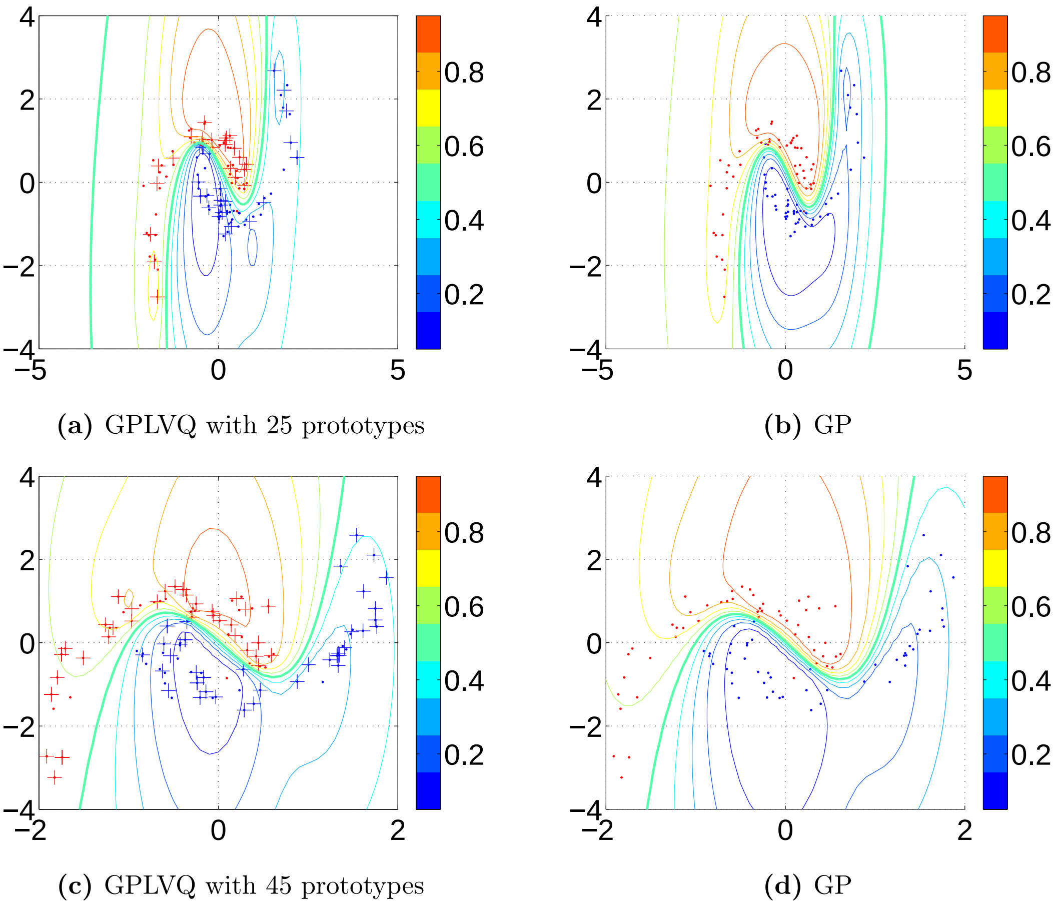


Figure 5.4: Visual comparison of probility field for GPLVQ with a high and a maximal number of prototypes to GP on identical data. The decision boundary for GP and GPLVQ is at $p = 0.5$.

5.3 Other datasets

5.3.1 Fisher's Iris dataset

The Iris dataset is a three-class, four-dimensional dataset on simple properties of three related *Iris* species [22]. Since the GPLVQ algorithm cannot handle multi-class problems, one class which formed its own cluster was excluded from the dataset. The two classes which are within the other cluster remained and are used for classification.

For LVQ and GPLVQ two prototypes were used. The results for variation of hyperparameter α are shown in table 5.4. The comparison chart for LVQ, GP and GPLVQ is presented in table 5.6.

The three algorithms performed similarly well on the dataset.

α	0.01	0.2	0.5	1	1.5	2	2.5	3	4	5	10
ϵ_{test}	0.16	0.12	0.10	0.08	0.07	0.07	0.07	0.07	0.08	0.07	0.07

Table 5.4: Mean rate of misclassification for different values of the hyperparameter α on Fisher's Iris dataset.

5.3.2 Adrenal Tumor dataset

The dataset has samples of two different types of adrenal tumors. The first class contains 102 samples of benign adrenocortical adenoma, the second class 45 samples of the malignant carcinoma. Every data point has 32 features [23].

GPLVQ does not classify the dataset well (table 5.5). For high or low values of the hyperparameter α, the classification rate is just as high as 61%. For $\alpha = 3$, the classification rate has its maximum at about 71%.

When the performance of GPLVQ is compared to LVQ and GP, LVQ and GP perform notably better than GPLVQ on the dataset (see 5.6), thus suggesting that the dataset is not particulary hard to classify. GP can classify about 90% of the test data points correctly, while LVQ manages about 89%. The very low error rate for the training for GP can be an indicator for overfitting on the dataset.

α	0.01	0.1	0.5	1	2	3	5	10	20
ϵ_{test}	0.38	0.40	0.38	0.38	0.29	0.23	0.28	0.29	.37

Table 5.5: Mean rate of misclassification for different values of the hyperparameter α on the Adrenal Tumor dataset.

	Fisher		Adrenal		MNIST	
	ϵ_{train}	ϵ_{test}	ϵ_{train}	ϵ_{test}	ϵ_{train}	ϵ_{test}
LVQ	0.05	0.06	0.09	0.11	0.01	0.01
GP	0.06	0.06	0.01	0.10	0.00	0.50
GPLVQ	0.06	0.07	0.22	0.23	0.49	0.50

Table 5.6: Comparison of the mean misclassification rate for LVQ, GP and GPLVQ for Fisher's Iris dataset, Adrenal Tumor dataset and MNIST dataset.

5.3.3 MNIST

The MNIST database contains a vast collection of handwritten digits. The database is advertised as being easy to use and set-up for testing classification algorithms [24]. Since GPLVQ can only handle two-class problems, the dataset used is a subset of the database. The dataset contains thousand randomly chosen samples from digits *two* and *nine*. A data point consists of 784 dimensions.

The classification of the two digits fails when using the regular set-up. As shown in table 5.7, the algorithm predicts the class in only half of the cases correctly, which means that it is not performing at all, since randomly choosing class membership would produce the same result. GP appears to struggle with the dataset, too. It performs very well on the training data, but can also not classify the test data. Since both algorithms struggle with the dataset, this is an indication that some property of the GP algorithm causes the problems with GPLVQ.

The dataset was modified to test whether the number of data points or the number of features for a data point influences the quality of the classification.

It was stated previously, that GP has a maximum of approximately 1000 data points that might be used for training [11]. The number of data points was chosen accordingly. If the number of data points is lowered from 1000 to 100 and even to 50, no improvements in classification are observed. The size of the dataset seems not to explain the poor classification performance.

The number of features is reduced through principal component analysis (PCA), the results are shown in table 5.8 and plotted in figure 5.5.

If the number of components is set to 40 or higher, GPLVQ and GP still have a classification error of $\epsilon_{test} \approx 50\%$. If the number of components is decreased further, classification results start to improve for GPLVQ and reach a minimum of $\epsilon_{test} = 5.9\%$ at three components. The test and train error develop very similarly for GPLVQ.
GP shows a very steep performance increase when moving from 30 to 25 components. The test error sinks from 43% to just 1.1%. The test error has a plateau from 25 components to 5 components and increases slightly for fewer components. The train error is very low for a high number of components, has a minimum of zero at thirty components and the increases for lower number of components. The train error has a maximum of

α	1	2	3	5	10	20	50
ϵ_{test}	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table 5.7: Mean rate of misclassification for different values of the hyperparameter α on the MNIST dataset.

# components		1	2	3	5	10	15	20	25	30	40	50	100
LVQ	ϵ_{train}	5.8	5.3	3.5	3.2	2.1	1.7	1.7	1.0	0.9	0.7	0.9	1.9
	ϵ_{test}	6.2	5.5	3.8	3.5	2.3	2.2	2.2	1.7	1.8	1.7	2.0	3.9
GP	ϵ_{train}	5.2	5.5	5.0	4.5	3.4	1.4	0.5	0.1	0.0	2.5	1.4	1.3
	ϵ_{test}	5.3	5.3	3.0	1.8	1.8	1.3	1.7	1.1	43	47	49	48
GPLVQ	ϵ_{train}	7.2	8.2	5.8	6.5	6.4	7.5	19	38	45	49	50	49
	ϵ_{test}	6.8	8.4	5.9	6.9	6.3	7.5	20	39	46	50	50	50

Table 5.8: Mean rate of misclassification for different number of PCA components on the MNIST dataset with $\alpha = 2$, $\sigma_n^2 = 0.3$, and 4 runs of ten-fold cross-validation

$\epsilon_{train} = 5.5\%$ at two components.
LVQ has a test error which is highest for just one component with $\epsilon_{test} = 6.2\%$ and decreases to 2.3% at ten components where it reaches a plateau. The train error behaves similarly but is slightly lower than the test error.

GP and LVQ both show a slight decrease in performance if the number of components becomes very low, this may be caused by discarding to much discriminative information with the principal components analysis.
All three algorithms benefit from the reduction of the number of components. The effect is not very strong for LVQ and only present for a higher number of components than with the other two algorithms. The performance of GPLVQ was improved strongly when the number of components was below 15. GP gains most from applying PCA to the input data. Where GP could not classify the raw data, it achieved the best test error observed for all three methods when using 25 components.

The results indicate that GP and GPLVQ both cannot classify high-dimensional and sparse data well. Since GPLVQ is based on GP, it can also be assumed that GPLVQ's bad performance was caused by the same problem that caused GP's bad performance.

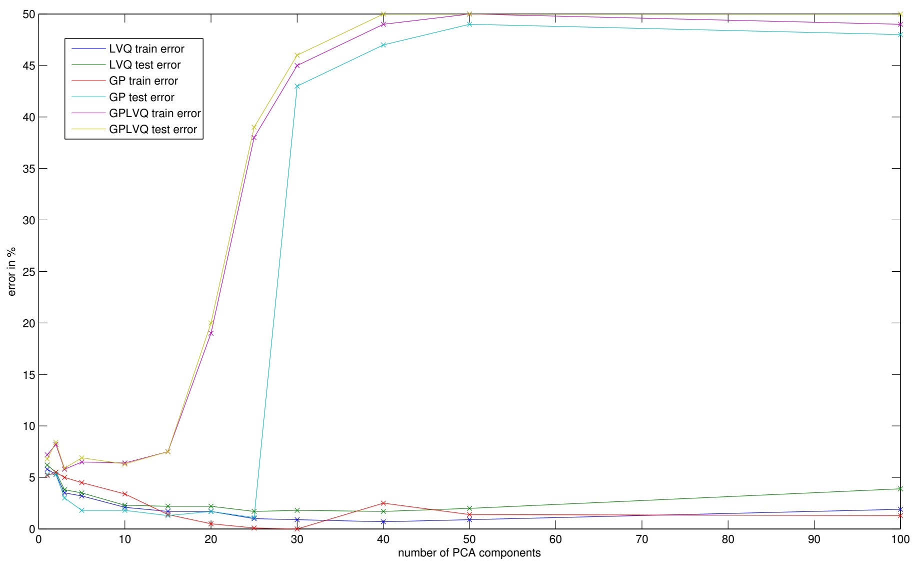


Figure 5.5: Plot of the mean rate of misclassification as presented in table 5.8)

Chapter 6

Conclusion and Outlook

The foundations of the GPLVQ algorithm, derived from the algorithms for LVQ and GP, were shown. The newly created algorithm tries to simplify datasets with many data points using LVQ combined with GP in a meaningful way. It inherits from GP that it is only able to classify two dimensional problems. It inherits from LVQ that underfitting by using a too small set of prototypes is still possible.

The performance of the new algorithm was tested. It was shown that GPLVQ performs well for simple datasets. However, GPLVQ struggled with the high dimensional inputs from the MNIST dataset. It was shown that reducing the complexity of the MNIST dataset through PCA yields good results. This is a hint that the dimensionality might be a problem.
Problems with a high dimensionality are described in literature. It is either suggested to tune the hyperparameters very precisely [9] or to deploy a GP variant, the Sparse Gaussian Process, which applies a dimensionality reduction [25, 26].

GPLVQ did not perform better than the ancestor algorithms LVQ and GP in any situation. For small datasets there is no advantage over GP if a probabilistic measure is needed. For advanced datasets GPLVQ showed major problems with high-dimensional data where LVQ classified the data well.

Since the new algorithm did not show any advantage, I suggest using GP or LVQ. During future research, one could investigate the decrease of performance of GPLVQ for high dimensional data further, e.g. by applying the Sparse Gaussian Process method. Another issue is that GPLVQ can only handle two class problems, this could be extended to multiclass problems.

Appendix A

List of symbols

α	Scaling parameter
η	Learning rate
μ	Mean of the Gaussian distribution
Σ	Variance of the Gaussian distribution
σ_n^2	Variance of the observation noise
θ	Set with all parameters of a GP
C	Number of different class labels y^μ
c	Corresponding class label for prototype $\mathbf{w}$
d	Distance metric
$\mathbb{E}$	Expectation of prediction
$\mathcal{GP}(m, k)$	Gaussian process with mean function m and covariance function k
J	Index for correct prototype $\mathbf{w}^J$
K	Index for incorrect prototype $\mathbf{w}^K$
$\mathbf{K}$	Gram matrix
$k(\mathbf{x}_i, \mathbf{x}_j)$	Covariance function evaluated at $\mathbf{x}_i$ and $\mathbf{x}_j$
M	Number of prototypes $\mathbf{w}^j$
$m(\mathbf{x}_i)$	Mean function for a Gaussian process
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean vector μ and covariance matrix Σ
P	Number of features $\mathbf{x}^\mu$
$\mathbb{V}$	Variance of prediction
W	Set of prototypes
$\mathbf{w}$	Prototype vector
$\mathbf{w}_k$	Correct prototype
$\mathbf{w}_j$	Incorrect prototype
$\mathbf{x}$	Feature vector
$\mathbf{x}^\mu$	Set of all feature vectors
y	Corresponding class label for feature vector $\mathbf{x}$
y^μ	Set of all class label for feature vectors $\mathbf{x}^\mu$

Appendix B

GPLVQ function

The covariance function for two points:

$$k(\mathbf{x}_1, \mathbf{x}_2) = e^{-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\alpha^2}} \tag{B.1}$$

Properties of the predictive distributions (see formulas 2.25 and 2.26 in [9]):

$$\bar{y}_* = \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \tag{B.2}$$

$$\mathbb{V}[y_*] = 1 - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_* \tag{B.3}$$

where $\mathbf{k}_* = \mathbf{k}(\mathbf{x}_*)$ is the vector of covariances between the test point and the n training points and where $\mathbf{K} = K(X, X)$ is the Gram matrix.

The Gaussian distribution in short and long notation:

$$p(y_* | \mathbf{x}_*, \mathbf{x}_{1:N}, c_{1:N}) \sim \mathcal{N}(\mu, \Sigma) \tag{B.4}$$

$$p(y_* | \mathbf{x}_*, \mathbf{x}_{1:N}, c_{1:N}) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left(-\frac{1}{2} (\mathbf{x}_* - \mu)^\top \Sigma^{-1} (\mathbf{x}_* - \mu) \right) \tag{B.5}$$

with the mean predicted label $\bar{y}_* = \mu = \mathbf{k}_*^\top \left(\mathbf{K} + \sigma_n^2 \mathbf{I} \right)^{-1} \mathbf{y}$, the actual label $\mathbf{x}_* = y$ and the variance of the prediction $\Sigma = 1 - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_*$

The derivation of the gradient function will be performed at first for a simplified situation where $\sigma_n^2 = 0$ and the dimensionality is 1. From that simplified situation higher dimensionalities and observation noise are reinstated.

$$p = \frac{1}{\sqrt{(2\pi)^N * \Sigma}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) \right) \tag{B.6}$$

Take the log of the previous equation

$$\log(p) = \log\left(\frac{1}{\sqrt{(2\pi)^N * \Sigma}}\right) - \left(\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right) \tag{B.7}$$

Replace $\mu = \mathbf{k}_*^\top \left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}$, $\mathbf{x}_* = y$ and $\Sigma = 1 - \mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*$

$$\begin{aligned}\log(p) = &\log\left(\frac{1}{\sqrt{(2\pi)^N}} \cdot \frac{1}{1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*}\right) - \left(\frac{1}{2}\right)\\ &\left[\left(\mathbf{k}_*^\top \left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right) - \mathbf{y}^*\right]^\top\\ &\left[1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*\right]^{-1}\\ &\left[\left(\mathbf{k}_*^\top \left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right) - \mathbf{y}^*\right]\end{aligned} \tag{B.8}$$

Set $\sigma_n^2 = 0$:

$$\begin{aligned}\log(p) = &\log\left(\frac{1}{\sqrt{(2\pi)^N}} \cdot \frac{1}{1-\mathbf{k}_*^\top(\mathbf{K}+0\mathbf{I})^{-1}\mathbf{k}_*}\right) - \left(\frac{1}{2}\right)\cdot\\ &\left[\left(\mathbf{k}_*^\top \left(\mathbf{K}+0\mathbf{I}\right)^{-1}\mathbf{y}\right) - \mathbf{y}^*\right]^\top\\ &\left[1-\mathbf{k}_*^\top(\mathbf{K}+0\mathbf{I})^{-1}\mathbf{k}_*\right]^{-1}\\ &\left[\left(\mathbf{k}_*^\top \left(\mathbf{K}+0\mathbf{I}\right)^{-1}\mathbf{y}\right) - \mathbf{y}^*\right]\end{aligned} \tag{B.9}$$

Simplify from the n dimensional to the $1D$ case.
$\mathbf{K} = K(X,X)$ is a $n \cdot n$ matrix with $k(x_i,x_j)$ as entries. $k(x,x) = 1$.
Switch to LVQ notation, i.e. the vector of labels $\mathbf{y}$ becomes the label of the prototype c. The vector with labels $\mathbf{y}^*$ becomes the label of the datapoint y. The covariance function changes from $\mathbf{k}_*$ to $k(\mathbf{x},\mathbf{w})$.

$$\begin{aligned}\log(p) = &\log\left(\frac{1}{\sqrt{2\pi}} \cdot \frac{1}{1-k^2(x,w)}\right) - \left(\frac{1}{2}\right)\\ &[c\cdot k(x,w)-y]^\top\left[1-k^2(x,w)\right]^{-1}[c\cdot k(x,w)-y]\end{aligned} \tag{B.10}$$

$$\log(p) = \log\left(\frac{1}{\sqrt{2\pi}} \cdot \frac{1}{1-k^2(x,w)}\right) - \left(\frac{1}{2}\right)\frac{[c\cdot k(x,w)-y]^2}{1-k^2(x,w)} \tag{B.11}$$

Constant term is discarded, since we want to calculate the derivative.

$$\log(p) = \log\left(\frac{1}{1-k^2(x,w)}\right) - \frac{1}{2}\frac{[c\cdot k(x,w)-y]^2}{1-k^2(x,w)} \tag{B.12}$$

Calculate the derivative of:

$$\frac{\partial \log(p)}{\partial w} = \frac{\partial}{\partial w}\left[\log\left(\frac{1}{1-k^2(x,w)}\right) - \frac{1}{2}\frac{[c\cdot k(x,w)-y]^2}{1-k^2(x,w)}\right] \tag{B.13}$$

The derivative of k with respect to w

$$\frac{\partial}{\partial w}k(x,w) = \frac{\partial}{\partial w}e^{-\frac{|x-w|^2}{2\alpha^2}} = \frac{x-w}{\alpha^2}k(x,w). \tag{B.14}$$

The partial derivatives of parts of the term are:

$$\frac{\partial}{\partial w}1-k^2(x,w) = -2\frac{(x-w)}{\alpha^2}k^2(x,w) \tag{B.15}$$

$$\frac{\partial}{\partial w}\frac{1}{1-k^2(x,w)} = 2\frac{(x-w)}{\alpha^2}\frac{k^2(x,w)}{(1-k^2(x,w))^2} \tag{B.16}$$

$$\frac{\partial}{\partial w}\log\left(\frac{1}{1-k^2(x,w)}\right) = -2\frac{(x-w)}{\alpha^2}\frac{k^2(x,w)}{1-k^2(x,w)} \tag{B.17}$$

$$\frac{\partial}{\partial w}\left(c\cdot k(x,w)-y\right)^2 = c\cdot 2\frac{(x-w)}{\alpha^2}k(x,w)\left(c\cdot k(x,w)-y\right) \tag{B.18}$$

Derivative of right term in the equation:

$$\frac{\partial}{\partial w}\frac{[c\cdot k(x,w)-y]^2}{1-k^2(x,w)} = \frac{2(x-w)k^2(x,w)\left(c\cdot k(x,w)-y\right)^2}{\alpha^2\left(1-k^2(x,w)\right)^2} + \frac{c\cdot 2(x-w)k(x,w)\left(c\cdot k(x,w)-y\right)}{\alpha^2\left(1-k^2(x,w)\right)} \tag{B.19}$$

$$\begin{aligned}\frac{\partial}{\partial w}\log p = & -\frac{2(x-w)k^2(x,w)}{\alpha^2(1-k^2(x,w))} \\ & -\frac{(x-w)k^2(x,w)\left(c\cdot k(x,w)-y\right)^2}{\alpha^2\left(1-k^2(x,w)\right)^2} - \frac{c\cdot(x-w)k(x,w)\left(c\cdot k(x,w)-y\right)}{\alpha^2\left(1-k^2(x,w)\right)}\end{aligned} \tag{B.20}$$

$$\frac{\partial}{\partial w}\log p = \frac{(x-w)k(x,w)}{\alpha^2(1-k^2(x,w))}\cdot \left(-2k(x,w) - \frac{k(x,w)\,(c\cdot k(x,w)-y)^2}{(1-k^2(x,w))} - c\cdot(c\cdot k(x,w)-y)\right) \tag{B.21}$$

Introduce the observation noise. See formula B.8.

$$\log(p) = \log\left(\frac{1}{\sqrt{(2\pi)^N}}\cdot\frac{1}{1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*}\right) - \left(\frac{1}{2}\right) \left[\left(\mathbf{k}_*^\top\left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]^\top \left[1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*\right]^{-1} \left[\left(\mathbf{k}_*^\top\left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right] \tag{B.22}$$

Assume 1D, remember that $\mathbf{K}=1$

$$\log(p) = \log\left(\frac{1}{\sqrt{(2\pi)^N}}\cdot\frac{1}{1-\mathbf{k}_*^\top(1+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*}\right) - \left(\frac{1}{2}\right) \left[\left(\mathbf{k}_*^\top\left(1+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]^\top \left[1-\mathbf{k}_*^\top(1+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*\right]^{-1} \left[\left(\mathbf{k}_*^\top\left(1+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right] \tag{B.23}$$

$$\log(p) = \log\left(\frac{1}{\sqrt{2\pi}}\cdot\frac{1}{1-\frac{k^2(x,w)}{1+\sigma_n^2}}\right) - \left(\frac{1}{2}\right) \left[c\cdot\frac{k(x,w)}{1+\sigma_n^2}-y\right]^\top\left[1-\frac{k^2(x,w)}{1+\sigma_n^2}\right]^{-1}\left[c\cdot\frac{k(x,w)}{1+\sigma_n^2}-y\right] \tag{B.24}$$

$$\log(p) = \log\left(\frac{1}{\sqrt{2\pi}}\cdot\frac{1}{1-\frac{k^2(x,w)}{1+\sigma_n^2}}\right) - \frac{1}{2}\frac{\left(c\cdot\frac{k(x,w)}{1+\sigma_n^2}-y\right)^2}{1-\frac{k^2(x,w)}{1+\sigma_n^2}} \tag{B.25}$$

Derivative of this term is:

$$\frac{\partial}{\partial w}\log p = -\frac{2(x-w)k^2(x,w)}{\alpha^2(1+\sigma_n^2)(1-\frac{k^2(x,w)}{1+\sigma_n^2})} \\ -\frac{(x-w)k^2(x,w)\left(c\cdot\frac{k(x,w)}{1+\sigma_n^2}-y\right)^2}{\alpha^2(1+\sigma_n^2)\left(1-\frac{k^2(x,w)}{1+\sigma_n^2}\right)^2} - \frac{c\cdot(x-w)k(x,w)\left(c\cdot\frac{k(x,w)}{1+\sigma_n^2}-y\right)}{\alpha^2(1+\sigma_n^2)\left(1-\frac{k^2(x,w)}{1+\sigma_n^2}\right)} \tag{B.26}$$

Return to higher dimensionalities than 1D.

$$\begin{aligned}\log(p) =& \log\left(\frac{1}{\sqrt{(2\pi)^N}}\cdot\frac{1}{1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*}\right)-\left(\frac{1}{2}\right)\\ &\left[\left(\mathbf{k}_*^\top\left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]^\top\\ &\left[1-\mathbf{k}_*^\top(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*\right]^{-1}\\ &\left[\left(\mathbf{k}_*^\top\left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]\end{aligned} \tag{B.27}$$

$K = K(X,X) = k(\mathbf{x},\mathbf{x}) = 1$ since the matrix X has just one entry $\mathbf{x}$

$$\begin{aligned}\log(p) =& \log\left(\frac{1}{\sqrt{(2\pi)^N}}\cdot\frac{1}{1-\mathbf{k}_*^\top(1+\sigma_n^2)^{-1}\mathbf{k}_*}\right)-\left(\frac{1}{2}\right)\\ &\left[\left(\mathbf{k}_*^\top\left(1+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]^\top\\ &\left[1-\mathbf{k}_*^\top(1+\sigma_n^2)^{-1}\mathbf{k}_*\right]^{-1}\\ &\left[\left(\mathbf{k}_*^\top\left(1+\sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y}\right)-\mathbf{y}^*\right]\end{aligned} \tag{B.28}$$

$$\log(p) = \log\left(\frac{1}{\sqrt{2\pi}}\cdot\frac{1}{1-\frac{k^2(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}}\right)-\frac{1}{2}\frac{\left(c\cdot\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}-y\right)^2}{1-\frac{k^2(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}} \tag{B.29}$$

Derivative of this term is:

$$\nabla_w p = \frac{(\mathbf{x}-\mathbf{w})k(\mathbf{x},\mathbf{w})}{\alpha^2(1+\sigma_n^2)(1-\frac{k^2(\mathbf{x},\mathbf{w})}{1+\sigma_n^2})}\cdot \\ \left(-2k(\mathbf{x},\mathbf{w})-\frac{k(\mathbf{x},\mathbf{w})\left(c\cdot\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}-y\right)^2}{\left(1-\frac{k^2(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}\right)}-c\cdot\left(c\cdot\frac{k(\mathbf{x},\mathbf{w})}{1+\sigma_n^2}-y\right)\right) \tag{B.30}$$

Appendix C

Implementation and Code

C.1 GP

Besides the very extensive book on Gaussian Processes by Rasmussen and Williams [2], a Matlab implementation called *Gaussian Processes for Machine Learning Toolbox* is also provided on the book's website: [27]. The toolbox features various mean functions, covariance functions and likelihood functions [28].

C.2 GPLVQ and derivatives

The implementation of GPLVQ was made with Matlab and is available from: [29].

The code can be started by running `run_gplvq` in Matlab. Without any parameters set, it will run a demo classification on the spirals dataset.

The code is designed for modularity. Every step in the process has its own function without any interaction to previous steps. The modular approach allows for fast and straight-forward replacement or extension of the algorithm. A new covariance function can be used or new rules for classification can be introduced.

For ease of implementation suitable Matlab functions were used where possible.

The visualization function uses PCA to map high dimensional data into 2D. This visualization is of limited use for high-dimensional datasets where dimensions can have high variance, but are omitted.

Bibliography

[1] J. Quinn, "Notes on combining LVQ and Gaussian processes for probabilistic prototype-based classification," 2013.

[2] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning*, pp. 63–71, Springer, 2004.

[3] N. N. Taleb, *The black swan: The impact of the highly improbable*, vol. 2. Random house, 2007.

[4] P. Boyle, *Gaussian processes for regression and optimisation.* PhD thesis, Victoria University of Wellington, 2007.

[5] A. O'Hagan and J. Kingman, "Curve fitting and optimal design for prediction," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–42, 1978.

[6] D. J. C. MacKay, "Introduction to Gaussian processes," *NATO ASI Series F Computer and Systems Sciences*, vol. 168, pp. 133–166, 1998.

[7] R. M. Neal, *Bayesian learning for neural networks.* PhD thesis, University of Toronto, 1995.

[8] D. G. Krige, "A statistical approach to some basic mine valuation problems on the witwatersrand," *Journal of the Southern African Institute of Mining and Metallurgy*, vol. 52, no. 6, pp. 119–139, 1951.

[9] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning. 2006*, vol. 38. MIT, 2006.

[10] C. K. I. Williams, "Prediction with Gaussian processes: From linear regression to linear prediction and beyond," in *Learning in graphical models*, pp. 599–621, Springer, 1997.

[11] C. K. I. Williams and C. E. Rasmussen, "Gaussian Processes for Regression," in *Advances in Neural Information Processing Systems 8*, MIT Press, 1996.

[12] M. Blum and M. A. Riedmiller, "Optimization of gaussian process hyperparameters using rprop.," in *ESANN*, 2013.

[13] R. M. Neal, "Monte carlo implementation of gaussian process models for bayesian regression and classification," *arXiv preprint physics/9701026*, 1997.

[14] C. K. I. Williams and D. Barber, "Bayesian classification with Gaussian processes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 12, pp. 1342–1351, 1998.

[15] A. Witoelar, M. Biehl, and B. Hammer, "Learning vector quantization: generalization ability and dynamics of competing prototypes," in *Dagstuhl Seminar Proceedings*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[16] T. Kohonen, "Improved versions of learning vector quantization," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 545–550, IEEE, 1990.

[17] P. Schneider, M. Biehl, and B. Hammer, "Relevance matrices in LVQ," *Similarity-based Clustering and its Application to Medicine and Biology*, no. 07131, 2007.

[18] E. T. Mwebaze, *Divergences for prototype-based classification and causal structure discovery.* PhD thesis, University of Groningen, 2014.

[19] A. Sato and K. Yamada, "Generalized learning vector quantization," *Advances in neural information processing systems*, pp. 423–429, 1996.

[20] P. Schneider, M. Biehl, and B. Hammer, "Adaptive relevance matrices in learning vector quantization," *Neural Computation*, vol. 21, no. 12, pp. 3532–3561, 2009.

[21] P. Schneider, M. Biehl, and B. Hammer, "Distance learning in discriminative vector quantization," *Neural Computation*, vol. 21, no. 10, pp. 2942–2969, 2009.

[22] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[23] M. Biehl, P. Schneider, D. Smith, H. Stiekema, A. Taylor, B. Hughes, C. Shackleton, P. Stewart, and W. Arlt, "Matrix relevance LVQ in steroid metabolomics based classification of adrenal tumors," in *20th european symposium on artificial neural networks (ESANN 2012)*, pp. 423–428, 2012.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, pp. 1257–1264, 2006.

[26] E. Snelson and Z. Ghahramani, "Variable noise and dimensionality reduction for sparse gaussian processes," *arXiv preprint arXiv:1206.6873*, 2012.

[27] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (GPML) toolbox." http://gaussianprocess.org/gpml/code/matlab/doc/.

[28] C. E. Rasmussen and H. Nickisch, “Gaussian processes for machine learning (GPML) toolbox,” *The Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, 2010.

[29] M. C. P. M. Scheepens, “GPLVQ code for thesis of Martien Scheepens.” `http://martien.home.fmf.nl/download/master/`.