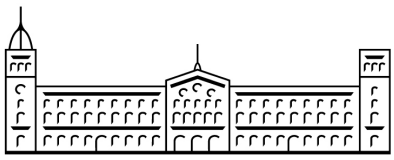


Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Gaussian Processes for Machine Learning

Autor: Gerard Martínez Canelles

Director: Dr. Jordi Vitrià Marca
Realitzat a: Mathematics and Computer Science

Barcelona, 28th June, 2017

The capacity of a parametric model, the amount of information that the model can represent, is bounded, even if the amount of observed data becomes unbounded.

-Zoubin Ghahramani [2013]

Acknowledgements

I would like to thank my advisor, Jordi Vitrià Marca, for so much encouragement, support and feedback. Jordi was patient while I spent the first few months chasing half-baked ideas, and then gently suggested a series of notions which worked. It was wonderful working with someone who is always willing to help and whose dedication to science is inspiring.

Contents

Abstract

The main focus of this project is to present clearly and concisely an overview of the main ideas of Gaussian processes for regression in a machine learning context. The introductory chapters contain core material and give some theoretical analysis of how to construct Gaussian processes models, the way to express many types of structure through kernels and how adding and multiplying different kernels combines their properties. Moreover explanatory examples are also covered in order to gain a deeper understanding of the material. Finally we provide an alternative approach from a Machine Learning complex problem carried out by a prestigious European Bank in endeavours to achieve a reliable method to predict customer expenses and consequently to forecast the balance of bank accounts. The aim was to analyze deeply this problem using Gaussian Processes for Regression from two different perspectives. Firstly, all the bank accounts studied as a global entity and then clustering the accounts regarding its similarity allowing a more flexible and adaptive approach.

In this work we will be concerned with supervised learning, which is the problem of learning input-output mapping from empirical data. Since we will work with continuous outputs this problem is known as regression.

In general we denote the input as **x**, and the output (or target) as y. The input is usually represented as a vector **x** as there are in general many input variables and the target y in regression problems, as mentioned before, must be continuous. Thus, if we have a dataset $\mathcal{D}$ of n observations, we write $\mathcal{D} = \{(x_i, y_i)|i = 1, ..., n\}$ or $\mathcal{D} = \{X, y\}$.

Given this training data we wish to make predictions for new inputs x_* that we have not seen in the training set. Hence, we need to move from the finite training data $\mathcal{D}$ to a function f that makes predictions for all possible inputs values. To do this we must make assumptions about the characteristics of the underlying function, as otherwise any function which is consistent with the training data would be equally valid. In that context, one particularly elegant way to learn functions is by giving a prior probability to every possible function, where higher probabilities are given to functions that we consider to be more likely.

In other words, we assume that $y_i = f(x_i)$, for some unknown function f, possibly corrupted by noise. Then we infer a distribution over functions given the data, $p(f|X, y)$, and we use this to make predictions given new inputs, i.e., to compute

$$p(y_*|x_*, X, y) = \int p(y_*|f, x_*)p(f|X, y)df$$

Thus, we will discuss a way to perform Bayesian inference over functions themselves. However this approach apppears to have serious problems, in that surely there are an uncountably infinite set of possible functions, and how are we going to compute with this set in finite time? This is where the Gaussian process, which is a generalitzation of the Gaussian probability distribution, comes to our rescue.

A GP defines a prior over functions, which can be converted into a posterior over functions once we have seen some data. Although it might seem difficult to represent a distribution over a function, it turns out that we only need to be able to define a distribution over the function's values at a finite, but arbitrary, set of points, say $x_1, \ldots, x_N$. A GP assumes that $p(f(x_1), ..., f(x_N))$ is jointly Gaussian, with some mean $\mu(x)$ and covariance $\Sigma(x)$ given by $\Sigma_{ij} = k(x_i, x_j)$, where k is a positive definite kernel function.

Actually, one of the main attractions of the Gaussian process framework is precisely that unites a sophisticated and consistent view with computational tractability. Therefore many models that are commonly employed in both machine learning and statistics are in fact special cases or restricted kinds of Gaussian processes.

To sum up, Gaussian Processes provide a principled, practical, probabilistic approach to learning in kernel machines and in some sense bring together work in the statistics and machine learning communities. The main goal of this work is to present clearly and concisely an overview of the main ideas of Gaussian processes in a machine learning context and to apply them in a complex forecasting problem consisting of predicting the movements of a bank account (expenses) in order to determine whether or not this bank

account will be in red numbers only having access to a small sample of historical values.

The main interest of being able to answer that question is that quite recently a leading bank company has been allocating a huge amount of resources and efforts trying to solve that riddle and creating a tool able to warn customers in case a major expense is likely to occur. Actually, they have already launched a preliminary version that has been tested in their employees. However, they adressed the problem in a different way since they faced that time series prediction problem using Long Short-Term Memory (LSTM) Networks, which is as type of recurrent neural network used in deep learning capable of succesfully training very large architectures.

As matter of fact, the responsibles of that project tried to provide a solution using Gaussian Processes, which at the end did not perform as well as LSTM Networks, but the differences were not very large. Thus, another added incentive is to reduce the gap between both approaches.

The work has a natural split into two parts, with the chapters up to and including chapter 3 covering core material, such as definition of Gaussian Processes for Regression, detailed analysis of kernel functions, model selection and a practical example. The remaining sections covers the Bank account forecasting problem.

The programming language used all along this work is Python 2.7. and some machine learning libraries such as scikit-learn (provided by Google), GPy (University of Sheffield) and PyGPs (University of Bremen). Source code to produce all figures and examples is available at http://www.github.com/gerardmartinezcanelles.

Symbols and Notation

Matrices are always capitalized. A subscript aterisk, such as in X_*, indicates reference to a *test set* quantity. A superscript asterisk denotes complex conjugate.

Symbol	Meaning
$\backslash$	left matrix divide: $A \setminus \boldsymbol{b}$ is the vector $\mathbf{x}$ which solves $A\boldsymbol{x} = \boldsymbol{b}$
$\lvert K \rvert$	determinant of K matrix
$\boldsymbol{y}^T$	the transpose of vector $\boldsymbol{y}$
$\propto$	proportional to; e.g. $p(x\lvert y) \propto f(x,y)$ means that $p(x\lvert y)$ is equal to $f(x,y)$ times a factor which is independent of x
$\sim$	distributed according to; example: $x \sim \mathcal{N}(\mu, \sigma^2)$
C	number of classes in a classification problem
cholesky(A)	Cholesky decomposition: L is a lower triangular matrix such that $LL^T = A$
$cov(\boldsymbol{f}_*)$	Gaussian process posterior covariance
D	dimension of input space $\mathcal{X}$
$\mathcal{D}$	data set: $\mathcal{D} = \{(x_i, y_i) \lvert i = 1, ..., n\}$
diag$(\boldsymbol{w})$	(vector argument) a diagonal matrix containing the elements of vector $\mathbf{w}$
diag(W)	(matrix argument) a diagonal matrix containing the elements of vector W
δ_{pq}	Kronecker delta, $\delta_{pq} = 1$ iff $p = q$ and 0 otherwise
$f(\boldsymbol{x})$ or $\boldsymbol{f}$	Gaussian process latent function values, $\boldsymbol{f} = (f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_n}))^T$
$\boldsymbol{f}_*$	Gaussian process (posterior) prediction (random variable)
$\bar{\boldsymbol{f}}_*$	Gaussian process posterior mean
GP	Gaussian process: $f \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'}))$, the function f is distributed as a Gaussian process with mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$
GPR	Gaussian process for Regression
I or I_n	the identity matrix (of size n)
$J_v(z)$	Bessel function of the first kind
$k(\boldsymbol{x}, \boldsymbol{x'})$	covariance (or kernel) function evaluated at $\mathbf{x}$ and $\mathbf{x'}$
$K(X, X')$	$n \times n$ covariance (or Gram) matrix
K_*	$n \times n_*$ matrix $K(X, X_*)$, the covariance between training and test cases
$\boldsymbol{k(x_*)}$	vector, short for $K(X, \boldsymbol{x}_*)$, when there is only a single test case
K_f or K	covariance matrix for the (noise free) $\mathbf{f}$ values
K_y	covariance matrix for the (noisy) $\mathbf{y}$ values; for independent homoscedastic noise, $K_y = K_f + \sigma_n^2 I$
$\mathcal{L}(a, b)$	loss function; the loss of predicting b, when a is true; note argument order
$\log(z)$	natural logarithm (base e)
ℓ or ℓ_d	characteristic length-scale (for input dimension d)
$m(\boldsymbol{x})$	the mean function of a Gaussian process
$\mathcal{N}(\mu, \sum)$	multivariate Gaussian (normal) distribution with mean vector μ and covariance matrix $\sum$
$\mathcal{N}(x)$	short for unit Gaussian $\boldsymbol{x} \sim \mathcal{N}(0, I)$
n and n_*	number of training (and test) cases
N	dimension of feature space
N_H	number of hidden units in a neural network

Symbol	Meaning
$y\|x$ and $p(y\|x)$	conditional random variable y given x and its probability(density)
σ_f^2	variance of the (noise free) signal
σ_n^2	noise variance
$\boldsymbol{\theta}$	vector of hyperparameters (parameters of the covariance function)
$tr(A)$	trace of (square) matrix A
$\mathcal{X}$	input space and also the index for the stochastic process
X	$D \times n$ matrix of the training inputs $\{\boldsymbol{x}_i\}_{i=1}^n$: the design matrix
X_*	matrix of test inputs
x_i	the ith training input
$\mathcal{O}(\cdot)$	The big-O asymptotic complexity of an algorithm.
SE	The squared-exponential kernel, also known as the radial-basis function (RBF) kernel, the Gaussian kernel, or the exponentiated quadratic.
RQ	The rational-quadratic kernel.
Per	The periodic kernel.
Lin	The linear kernel.
WN	The white-noise kernel.
C	The constant kernel.

Chapter 1

Gaussian Process for Regression

In this first chapter Gaussian process methods for regression problems is described. There are several ways to interpret Gaussian process (GP) regression models and one of them is *function-space view* which defines a distribution over functions and inference taking place directly in the space of functions. This approach will be discussed in section *1.1*. Another interesting and equivalent view of GP, which may be more familiar and accessible, is the *weight-space view* where GP are presented as a Bayesian analysis of the standard linear regression model [1]. This first chapter also includes a section where we discuss how to incorporate non zero-mean functions into the models and in the last section an easy example is shown.

1.1 Function-space View

Gaussian processes are a simple and general class of models of functions, meaning, we use them to describe a distribution over functions with a continuous domain. Formally:

Definition 1.1. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.[2]

[1] This particular view is extensively explained in *Carl E. Rasmussen and Christopher K.I. Williams. Gaussian Processes for Machine Learning, volume 38*

[2] In probability theory and statistics, Gaussian processes are usually defined as a real-valued stochastic process $\{X_t, t \in T\}$, where T is an index set and all the finite-dimensional distributions have a multivariate normal distribution. That is, for any choice of distinct values $t_1, \ldots, t_k \in T$, the random vector $X = (X_{t_1}, \ldots, X_{t_k})$ has a multivariate normal distribution. Where a multivariate Gaussian (or Normal) distribution has a joint probability density given by

$$p(x|m, \Sigma) = (2\pi)^{-M/2}|\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right) \tag{1.1}$$

where m is the mean vector (of length M) and Σ is the (symmetric, positive definite) covariance matrix (of size $M \times M$).

A Gaussian process is completely specified by its mean function and covariance function, defining both of them as:

$$m(x) = \mathbf{E}[\mathbf{f}(\mathbf{x})] \tag{1.2}$$

$$k(x, x') = \mathbf{E}[(\mathbf{f}(\mathbf{x}) - \mathbf{m}(\mathbf{x}))(\mathbf{f}(\mathbf{x}') - \mathbf{m}(\mathbf{x}'))] \tag{1.3}$$

and will write the Gaussian process as

$$f(x) \sim GP(m(x), k(x, x')) \tag{1.4}$$

It is common practice to assume that the mean function is simply zero everywhere, although this is not necessary as we will see since we can incorporate non-zero mean functions into the models.

Note that in our case the random variables represent the value of the funtion $f(x)$ at location $\mathbf{x}$. On the other hand, it may happen that the index set of the random variables is time, in other words, Gaussian processes can be defined over time. Indeed, many of the examples discussed in that work are time series.

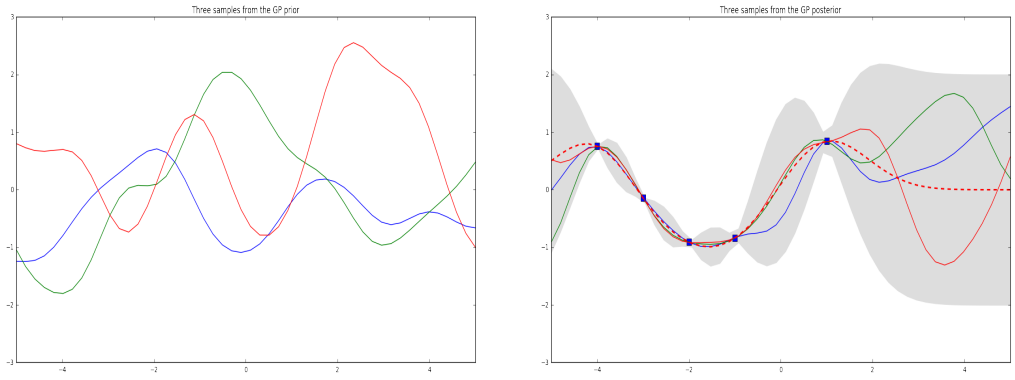
After accounting for the mean, the kind of structure that can be captured by a GP model is entirely determined by its covariance function, also known as kernel. The kernel specifies how the model generalizes, or extrapolates to new data. There are many possible choices of covariance function, and we can specify a wide range of models just by specifying the kernel of a GP. Actually, one of the main difficulties in using GPs is constructing a kernel which represents the particular structure present in the data being modelled. An example of a covariance function is the *Squared Exponential* (SE) which specifies the covariance between pairs of random variables.

$$Cov(f(x_p), f(x_q)) = k(x_p, x_q) = exp(-\frac{1}{2}|x_p - x_q|^2) \tag{1.5}$$

Note that the covariance between the outputs is written as a function of the inputs. As said before, the specification of the covariance function implies a distribution over functions. In the example in Figure 1.1 we have drawn samples from the distribution of functions evaluated at any number of input points X after determining the covariance function using SE kernel. More precisely we have generated a random Gaussian vector with this covariance matrix

$$f \sim \mathcal{N}(0, K(X, X))$$

and plotted the generated values as a function of the inputs. The mechanism used to generate multivariate Gaussian samples is explained in detail in the Github.



(a) prior (b) posterior

Figure 1.1: Panel (a) shows three functions drawn at random from a GP prior, in other words, a GP not coinditioned on any datapoints. Panel (b) The posterior after conditioning on five noise free observations . The shaded area represents the 95% confidence region, corresponding to the pointwise mean plus and minus two times the standard desviation for each input value.

1.1.1 Prediction with Noise-free Observations

Taking into account the information provided by the Figure 1.1 to get the posterior distribution over functions, intuitively, one may think of generating functions from the prior and rejecting the ones that disagree with the noise free observations. Even though this strategy would not be computationally very efficient. In probabilistic terms conditioning the joint Gaussian prior distribution on the observations is extremely simple.

Therefore let $\{(x_i, f_i)|i = 1, ..., n\}$ be the noise free observations, n the number of traning points and n_* be the number of test points. Expressing the idea mentioned in the paragraph above in a more formal way and keeping in mind that the joint distribution of the training outputs $\mathbf{f}$, and the test outputs $\boldsymbol{f}_*$, according to the prior is

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \quad \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{1.6}$$

where $K(X, X_*)$ denotes the $n \times n_*$ matrix of the covariances evaluated at all pairs of training and test points, and similarly for the other entries $K(X, X), K(X_*, X_*)$ and $K(X_*, X)$. The noise free-predictive distribution is given by

$$\boldsymbol{f}_*|X_*, X, \quad \boldsymbol{f} \sim \mathcal{N}(K(X_*, X)K(X, X)^{-1}\boldsymbol{f}, \quad K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)) \tag{1.7}$$

Function values $\boldsymbol{f}_*$ can be sampled from the joint posterior distribution by evaluating the mean and covariance matrix from equation (1.7). Figure 1.1b shows the results of these computations given the five datapoint marked with blue dots.

1.1.2 Prediction using Noisy Observations

Although there are some situations where it is reasonable that the observations are noise-free (e.g. computer simulations), it is uncommon to have acces to function values themselves.

Thus, we tend to consider models that incorporate a noise term ε giving rise to $y = f(x) + \varepsilon$ expression. Some of the assumptions of the noise term include additive independent identically Gaussian distribution and variance σ_n^2 so $\varepsilon_n \sim \mathcal{N}(0, \sigma_n^2)$.

Indeed, despite the fact that more complicated noise models with non-trivial covariance structure can also be handled, we will also assume the property of independence between noise terms. To sum up, all the assumptions mentioned before cause that the prior on the noisy observations becomes

$$cov(y_p, y_q) = k(x_p, x_q) + \sigma_n^2 \delta_{pq} \quad or \quad cov(y) = K(X, X) + \sigma_n^2 I \tag{1.8}$$

where δ_{pq} is a Kronecker delta which is one iff $p = q$ and zero otherwise. Taking into consideration the noise term and introducing it in equation (1.6) we can write the joint distribution of the observed target values and the function values at the test locations under the prior as

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f_*} \end{bmatrix} \sim \mathcal{N}\left(0, \quad \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{1.9}$$

Thus, the key predictive equations for Gaussian process regression, regarding the conclusions obtained in (1.7), are

$$\boldsymbol{f_*}|X, \boldsymbol{y}, X_* \sim \mathcal{N}(\bar{\boldsymbol{f}}_*, cov(\boldsymbol{f_*})) \tag{1.10}$$

where

$$\bar{\boldsymbol{f}}_* \triangleq \mathbf{E}[\boldsymbol{f_*}|X, \boldsymbol{y}, X_*] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\boldsymbol{y} \tag{1.11}$$

$$cov(\boldsymbol{f_*}) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_*) \tag{1.12}$$

Before considering one particular case of GP, we should examine in more detail expression for the variance given in equation (1.12). First, note that the variance only depends on the inputs, and not on the observed targets. Secondly, spliting (1.12) term by term we realize that the variance is the difference between two terms: the simple prior covariance $K(X_*, X_*)$ minus the information the observation gives us about the function. Moreover, the equation holds unchanged when X_* denotes multiple test inputs.

Now let's evaluate the case when the test set consists of a single point x_*. In that context the predictive distributions obtained, after adapting equations (1.11) and (1.12) to that particular case, reduce to

$$\bar{f}_* = \boldsymbol{k_*^T}[K(X, X) + \sigma_n^2 I]^{-1}\boldsymbol{y} \tag{1.13}$$

$$\mathbf{V}[(f_*)] = k(x_*, x_*) - \boldsymbol{k}_*^T[K(X,X) + \sigma^2 I]^{-1}\boldsymbol{k}_* \tag{1.14}$$

where $\boldsymbol{k}_* = \boldsymbol{k}(\boldsymbol{x}_*)$ denotes the vector of covariances between the test point and the n training points. Since equation (1.13) is a linear combination of observations $\mathbf{y}$ is sometimes referred to as a linear predictor, thus it can be rewritted as a linear combination of n kernels functions, each one centered on a training point, by writing

$$\bar{f}(x_*) = \sum_{i=1}^{n} \alpha_i k(x_i, x_*) \tag{1.15}$$

where $\alpha_i = ((K(X,X) + \sigma_n^2 I)^{-1})\boldsymbol{y}$. Even though the GP defines a joint Gaussian distribution over all of the y variables, for making predictions at $\boldsymbol{x}_*$ we only care about the (n+1)-dimensional distribution defined by the n training points and the test point. Coinditioning this (n+1)-dimensional distribution on the observations gives us the desired result since a Gaussian distribution is marginalized by just taking the relevant block of the joint covariance matrix.

Before concluding that section, I would like to introduce the concept of the *marginal likelihood* or $p(y|X)$. The marginal likelihood is the integral of the likelihood times the prior

$$p(y|X) = \int p(y|f, X)p(f|X)df \tag{1.16}$$

The term marginal likelihood refers to the marginalization over the function values $\mathbf{f}$. Under the Gaussian process model the prior is Gaussian, $\boldsymbol{f}|X \sim \mathcal{N}(0,\ K(X,X))$, or

$$logp(f|X) = -\frac{1}{2}\boldsymbol{f}^T K(X,X)^{-1}\boldsymbol{f} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{1.17}$$

and the likelihood is a factorized Gaussian $\boldsymbol{y}|\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma_n^2 I)$ so knowing that the product of two Gaussians gives another (un-normalized) Gaussian [3] and the normalizing constant also looks itself like a Gaussian[4] , the log marginal likelihood can be written as

$$logp(y|X) = -\frac{1}{2}\boldsymbol{y}^T(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|(K(X,X) + \sigma_n^2 I)| - \frac{n}{2}\log 2\pi \tag{1.18}$$

Although this result can also be obtained directly by observing that.

$$y \sim \mathcal{N}(0,\ K(X,X) + \sigma_n^2 I)$$

A practical implementation of Gaussian process regression is shown below and the complete python code is provided in the Github. The algorithm uses Cholesky decomposition since it is numerically more stable and faster. Computing the matrix inverse in equations in a conventional way takes $\mathcal{O}(n^3)$ time, making exact inference prohibitively slow for

[3] $\mathcal{N}(x|a,A)\mathcal{N}(x|b,B) = Z^{-1}\mathcal{N}(x|c,C)$ where $c = C(A^{-1}a + B^{-1}b)$ and $C = (A^{-1} + B^{-1})^{-1}$

[4] $Z^{-1} = (2\pi)^{-D/2}|A+B|^{-1/2}\exp(-\frac{1}{2}(a-b)^T(A+B)^{-1}(a-b))$

more than a few thousand datapoints. However the computational complexity for the Cholesky decomposition in line 2 is $\mathcal{O}(n^3/6)$. We also have to consider $\mathcal{O}(n^2/2)$ complexity for solving triangular systems in line 3 and (for each test case) in line 5. The algorithm returns the predictive mean and variance for noise free data test, adding the noise variance σ_n^2 to the predictive variance of $\boldsymbol{f}_*$ allows us to compute the predictive distribution for noisy test data $\boldsymbol{y}_*$.

input: X(inputs), **y** (targets), k (covariance function), σ_n^2 (noise level), x_* (test input)
2: $L := cholesky(K + \sigma_n^2 I)$
$\boldsymbol{\alpha} := L^T \setminus (L \setminus y)$
4: $\bar{f}(x_*) := \boldsymbol{k}_*^T \boldsymbol{\alpha}$
$\boldsymbol{v} := L \setminus \boldsymbol{k}_*$
6: $\mathcal{V}[f_*] := k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{v}^T \boldsymbol{v}$
$\log p(\boldsymbol{y}|X) := -\frac{1}{2}\boldsymbol{y}^T \alpha - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$
8: **return**: $\bar{f}(x_*)$ (mean), $\mathcal{V}[f_*]$ (variance), $\log p(y|X)$ (log marginal likelihood)

1.1.3 Non-zero mean Functions

There are several reasons why it is quite common to consider GPs with a zero mean function such as interpretability of the model or convenience of expressing prior information.

Note that this is not necessarily a drastic limitation since the mean of the posterior process is not confined to be zero. Moreover, the use of explicit fixed (deterministic) mean function $\boldsymbol{m(x)}$ is a way to specify a non-zero mean over functions. In other words, we can simply apply the zero mean GP to the difference between the observations and the fixed mean function. With

$$f(x) \sim GP(m(x), k(x, x')) \tag{1.19}$$

the predictive mean becomes

$$\bar{f}_* = m(X_*) + K(X_*, X)K_y^{-1}(y - m(X)), \tag{1.20}$$

where $K_y = K + \sigma_n^2 I$ and the predictive variance remains unchanged from eq. (1.12).

1.1.4 Example

After providing a theoretical explanation of how Gaussian Processes regression works it would be interesting to show a step by step example in order to make the forecasting process more understandable.
Let's consider six noisy data points (error bars are indicated with vertical lines) and we are interested in estimating a seventh at x_*.

For example $(x_i, y_i) = [(-2.5, -0.6), (-1.50, -0.1), (-0.5, 0.3), (0.75, 0.45), (1.95, 0.6), (2.8, 0.75)]$ for $i = \{1, \ldots, 6\}$ and $x_* = 3.2$.

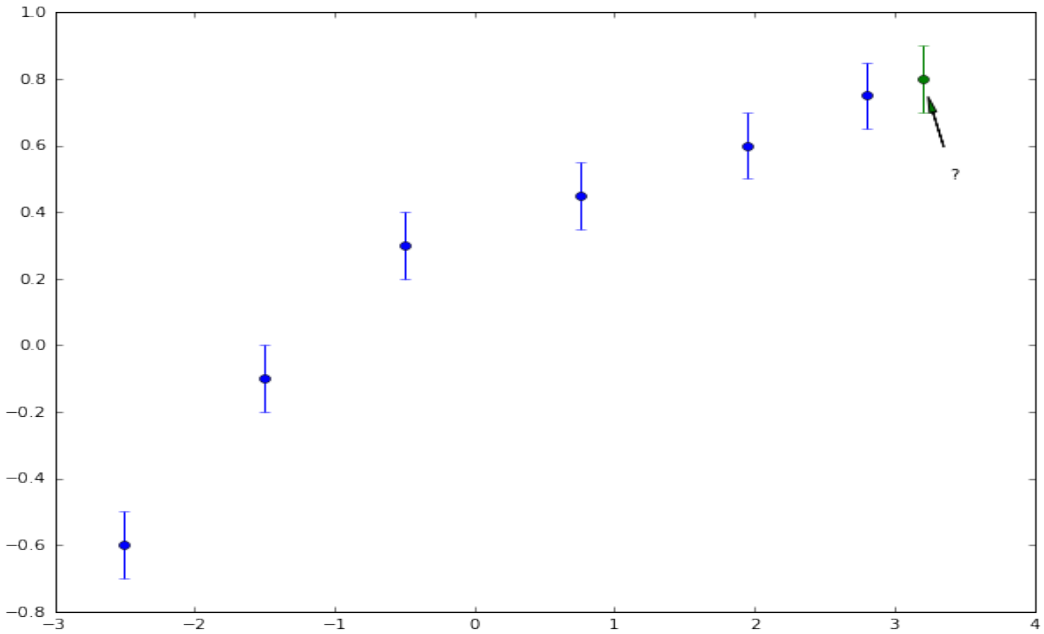


Figure 1.2: Blue points represent the noisy data known and the green one the seventh point that we are interested in estimating.

As mentioned in previous sections what relates one observation to another in such cases is the covariance function. Purely for simplicity of exposition our choice would be the squared exponential (SE) eq.(1.5) and since the data points are noisy we fold the noise into $k(x, x')$, by writing

$$k(x, x') = \sigma_f^2 exp\left(-\frac{(x - x')^2}{2\ell^2} \right) + \sigma_n^2 \delta(x, x') \tag{1.21}$$

where σ_f^2 should be high for functions which cover a broad range on the y axis, $\delta(x, x')$ is the Kronecker delta function and ℓ is the length-scale parameter whose role would be explained in detail in the next chapter.
To prepare for GPR, we calculate the covariance function, among all possible combinations of these points, summarizing our findings in three matrices:

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \ldots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \ldots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \ldots & k(x_n, x_n) \end{bmatrix}$$

$$K_* = [k(x_*, x_1), k(x_*, x_1), \cdots, k(x_*, x_n)] \qquad K_{**} = [k(x_*, x_*)]$$

In our particular case there are 6 observations with $x_i = [-2.5, -1.50, -0.5, 0.75, 1.95, 2.8]$ for $i = \{1, \ldots, 6\}$ and with judicious choices of $\sigma_f^2 = 1.5$, $\ell = 2.0$ and $\sigma_n^2 = 0.1$ from the error bars. Thus, we have enough to calculate the covariance matrices using eq. (1.21). Thus:

$$K = \begin{bmatrix} 1.6 & 1.323745 & 0.909795 & 0.400577 & 0.126205 & 0.044789 \\ 1.323745 & 1.6 & 1.323745 & 0.796643 & 0.33879 & 0.148705 \\ 0.909795 & 1.323745 & 1.6 & 1.233866 & 0.708328 & 0.384510 \\ 0.400577 & 0.796643 & 1.233866 & 1.6 & 1.252905 & 0.88705 \\ 0.126205 & 0.33879 & 0.708328 & 1.252905 & 1.6 & 1.370468 \\ 0.044789 & 0.148705 & 0.384510 & 0.88705 & 1.370468 & 1.6 \end{bmatrix}$$

$K_* = [0.025841 \quad 0.094819 \quad 0.270959 \quad 0.708328 \quad 1.233866 \quad 1.470298]$ and $K_{**} = [1.6]$

Since the test set consists of a single point x_* and taking into consideration the expressions given by (1.13) and (1.14) the mean of the distribution and the uncertainty of the estimatiton captured by the variance are

$$\bar{f}_* = 0.68098409 \qquad \mathbf{V}[(f_*)] = 0.23471905$$

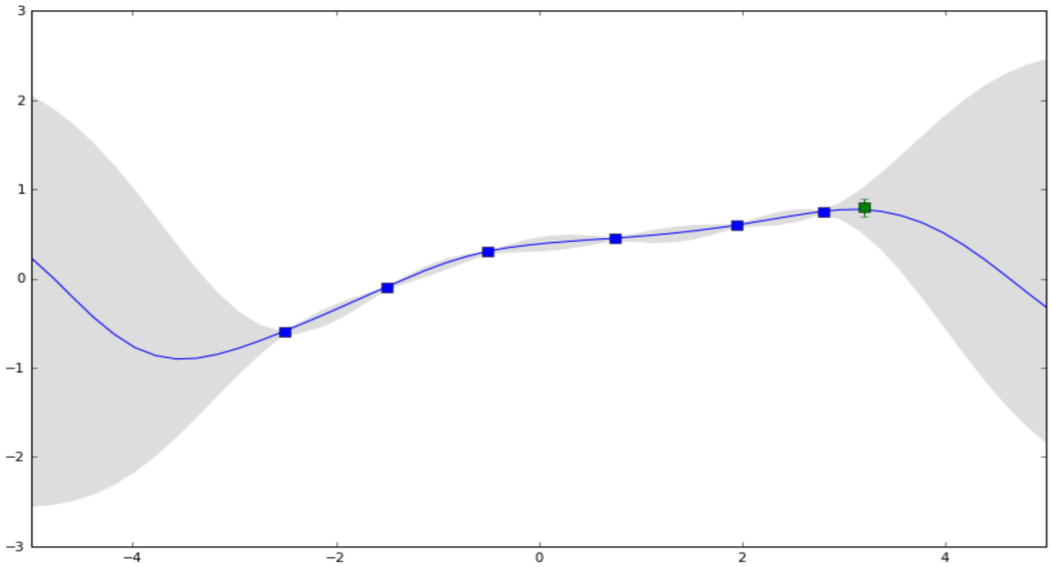


Figure 1.3: The GP posterior conditioned on six noisy observations and the prediction for the test point $x_* = 3.2$. The shaded area represents the 95% confidence region.

Chapter 2

Covariance Functions

The covariance function (also called kernel, kernel function, or covariance kernel) is the driving factor in a Gaussian process predictor as it specifies which functions are likely under the GP prior, which in turn determines the generalization properties of the model. In other words, choosing a useful kernel is equivalent to learning a useful representation of the input as it encodes our assumptions about the function which we wish to learn.

Colloquially, kernels are often said to specify the similarity between two objects, in our case, data points. In this way, as we have already mentioned in the first chapter, inputs $\mathbf{x}$ which are close are likely to have similar target values y, and thus training points that are near to a test point should be informative about the prediction at that point.

The purpose of this chapter is to give some theoretical properties of covariance function as well as some of the commonly-used examples. We also show how to use kernels to build models of functions with many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, changepoints and some of the structures which can be obtained by combining them.

2.1 Definition

Definition 2.1. *Let $\mathcal{X}$ be a nonempty set, sometimes referred to as the index set. A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a positive definite (p.d.) kernel on $\mathcal{X}$ if*

$$\sum_{i,j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \tag{2.1}$$

holds for any $n \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathcal{X}$, $c_1, \ldots, c_n \in \mathbb{R}$

Kernels are usually complex valued functions, but in this work we assume real-valued functions, which is the common practice in machine learning and other applications of p.d. kernels.

Some general properties are:

For a familiy of p.d. kernels $(K_i)_{i\in\mathbb{N}}$, $K_i : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

- The sum $\sum_i^n \lambda_i K_i$ is p.d. given $\lambda_1, \ldots, \lambda_n \geq 0$
- The product $K_1^{a_1}, \ldots, K_n^{a_n}$ is p.d. given $a_1, \ldots, a_n \in \mathbb{N}$
- The limit $K = \lim_{n\to\infty} K_n$ is p.d. if the limit exists

If $(\mathcal{X}_i)_{i=1}^n$ is a sequence of sets, and $(K_i)_{i=1}^n$, $K_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}$ a sequence of p.d. kernels, then both

- $K((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \prod_{i=1}^n K_i(x_i, y_i)$
- $K((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \sum_{i=1}^n K_i(x_i, y_i)$

are p.d. kernels on $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$
Moreover, let $\mathcal{X}_0 \subset \mathcal{X}$. Then the restriction K_0 of K to $\mathcal{X}_0 \times \mathcal{X}_0$ is also a p.d. kernel.

In this work we consider covariance functions where the inputs domain $\mathcal{X}$ is a subset of the vector space $\mathbb{R}^D$. A footnote, that provides an alternative and curious definition of kernel[1] in a machine learning context by R.Schaback and H.Wendland, as well as a more detailed explanation of positive defined matrix propierty is added below. [2]

Each kernel has a number of parameters which specify the precise shape of the covariance function. These are sometimes referred to as hyperparameters, since they can be viewed as specifying a distribution over function parameters, instead of being parameters which specify a function directly. The length-scale ℓ, the signal variance σ_f^2 and the noise variance σ_n^2 are the most representative. In the next section, for every mentioned kernel we will provide some graphics showing the effects of varying the hyperparameters on GP prediction as well as more formal definitions.

Before giving an overview of some commonly used kernels we will provide a popular classification of them.

If a covariance function is a function of $\boldsymbol{x} - \boldsymbol{x'}$, thus is invariant to translations in the input space, is called a **statitonary** covariance function. If further the covariance function is invariant to all rigid motions, meaning, is a function only of $|\boldsymbol{x} - \boldsymbol{x'}|$ it is called **isotropic**. For example the squared exponential covariance function given in equation (1.5) is isotropic and consequently stationary[3].

On the other hand, if a covariance function depends only on $\boldsymbol{x}$ and $\boldsymbol{x'}$ through $\boldsymbol{x} \cdot \boldsymbol{x'}$ we call it a dot product covariance function. An important example is the inhomogeneous

[1] A kernel is a function $K : \Omega \times \Omega \to \mathbb{R}$ where Ω can be an arbitrary nonempty set. Some readers may consider this as being far too general. However, in the context of learning algorithms, the set Ω defines the possible learning inputs. Thus Ω should be general enough to allow Shakespeare texts or X-ray images, i.e. Ω should better have no predefined structure at all. Thus the kernels occurring in machine learning are extremely general, but still they take a special form which can be tailored to meet the demands of applications.

[2] A real $n \times n$ covariance matrix K which satisfies $Q(v) = v^T K v \geq 0$ for all vectors $v \in R^n$ is called positive semidefinite (PSD). If $Q(v) = 0$ only when $v = 0$ the matrix is postive definite. $Q(v)$ is called a quadratic form . A symmettric matrix is PSD iff all of its eigenvalues are non-negative.

[3] As the *kernel* is now only a function of $r = |\boldsymbol{x} - \boldsymbol{x'}|$ this are also known as *radial basis functions* (RBFs)

polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x'}) = (\sigma_0^2 + \boldsymbol{x} \cdot \boldsymbol{x'})^p$ where p is a positive integer. Note that dot product covariance functions are invariant to a rotation of the coordinates about the origin, but not translations.

Finally, there are also other interesting kernels which are not included in the groups of stationary or dot product kernels. This covariance functions belongs to a particular type of neural network and this construction is due to Radford M. Neal and his exhaustive research in Bayesian Learning for Neural Networks.

2.2 Examples of Kernels

2.2.1 Stationary Kernels

As mentioned above, a stationary kernel is a function of $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x'}$. Sometimes in this case we will write k as a function of a single argument, i.e, $k(\boldsymbol{\tau})$. Although, not being the main aim of this section it must be said that the covariance function of a *stationary* process can be represented as the Fourier transform of a positive finite measure as Bochner's theorem states. [4]

Theorem 2.2. *(Bochner's theorem) A complex-valued function k on* $\mathbf{R}^d$ *is the covariance function for a weakly stationary*[5] *mean square continuous complex-valued random process on* $\mathbf{R}^d$ *if and only if it can be represented as*

$$k(\tau) = \int_{\mathbf{R}^d} e^{2\pi i s \cdot \tau} d\mu(s)$$

where μ *is a positive finite measure.*

If μ has a density $S(s)$, then is called the *spectral density* or *power spectrum* of k, and k and S are Fourier duals:

$$k(\tau) = \int S(s) e^{2\pi i s^T \tau} ds, \tag{2.2}$$

$$S(s) = \int k(\tau) e^{-2\pi i s^T \tau} d\tau. \tag{2.3}$$

In other words, a spectral density entirely determines the properties of a stationary kernel. And often spectral densities are more interpretable than kernels. If we Fourier transform a stationary kernel, the resulting spectral density shows the distribution of support for various frequencies. A heavy tailed spectral density has relatively large support for high frequencies. A uniform density over the spectrum corresponds to white noise. Therefore draws from a process with a heavy tailed spectral density tend to appear more erratic (containing higher frequencies, and behaving more like white noise) than draws from a

[4] A proof and further reading can be found in Stein, M. L. (1999). Interpolation of Spatial Data. Springer-Verlag, New York.

[5] A Gaussian time series $\{X_t\}$ is said to be stationary if $m(t) = \mathcal{E}[X_t] = \mu$ is independent of t and $Cov(X_{t+h}, X_t)$ is independent of t for all h.

process with spectral density that concentrates its support on low frequency functions. Indeed, one can gain insights into kernels by considering their spectral densities.
We now give some examples of stationary covariance functions.

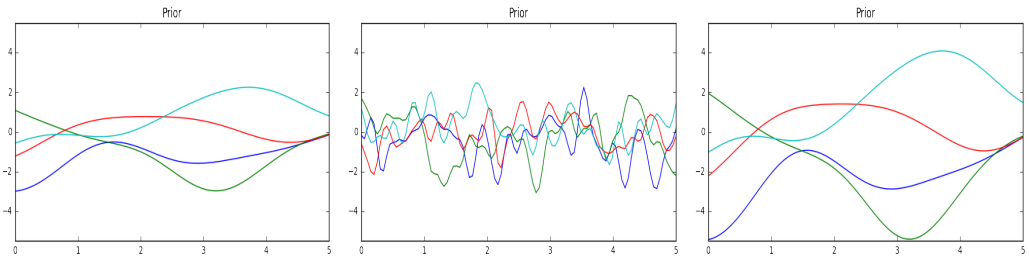
Squared Exponential

The SE kernel has become the de-facto default kernel for GPs and SVMs (Support Vector Machines). Also known as the Radial Basis Function or the Gaussian Kernel. It has already been introduced in the first chapter, equation (1.5), and has the form

$$k_{SE}(x, x') = \sigma_f^2 exp\left(-\frac{(x - x')^2}{2\ell^2} \right) \tag{2.4}$$

This covariance function has some nice properties, such as infinitely differentiability, which means that the GP has mean square derivatives of all orders, and thus is very smooth. It has two parameters:

-The lengthscale ℓ determines the length of the 'wiggles' in a function. In general, we won't be able to extrapolate more than ℓ units away from your data. Informally can be thought of as roughly the distance you have to move in input space before the function value can change significantly.[6]
- The output variance σ_f^2 determines the average distance of your function away from its mean. Every kernel has this parameter out in front; it's just a scale factor.



(a) $(\ell, \sigma_f^2) = (1, 1)$ (b) $(\ell, \sigma_f^2) = (0.1, 1)$ (c) $(\ell, \sigma_f^2) = (1, 3.3)$

Figure 2.1: GP priors generated with the SE kernel with different hyperparameters values. (a) Shows priors generated with $(l, \sigma_f^2) = (1, 1)$. The function is very smooth and the average distance of the functions away from its mean is quite controlled, thus the variance term is small. Panel (b) The length-scale has been shortened and the priors are more wiggled and oscillate more quickly. Panel (c) As the variance is larger the range of the priors increases.

[6]For 1-d GP one way to understand the characteristic length-scale of the process is in terms of the number of upcrossing of a level u. The number of upcrossings $\mathbf{E}[\mathbf{N_u}]$ of the level u on the unit interval by a zero-mean, stationary, almost surely continuous Gaussian process is given by

$$\mathbf{E}[N_u] = \frac{1}{2\pi}\sqrt{\frac{-k''(0)}{k(0)}}\exp(-\frac{u^2}{2k(0)})$$

The Matérn Class of Covariance Functions

The Matérn class of covariance functions is given by

$$k_{Matern}(\tau) = \frac{2^{1-v}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu\tau}}{\ell}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu\tau}}{\ell}\right)^{\nu} \tag{2.5}$$

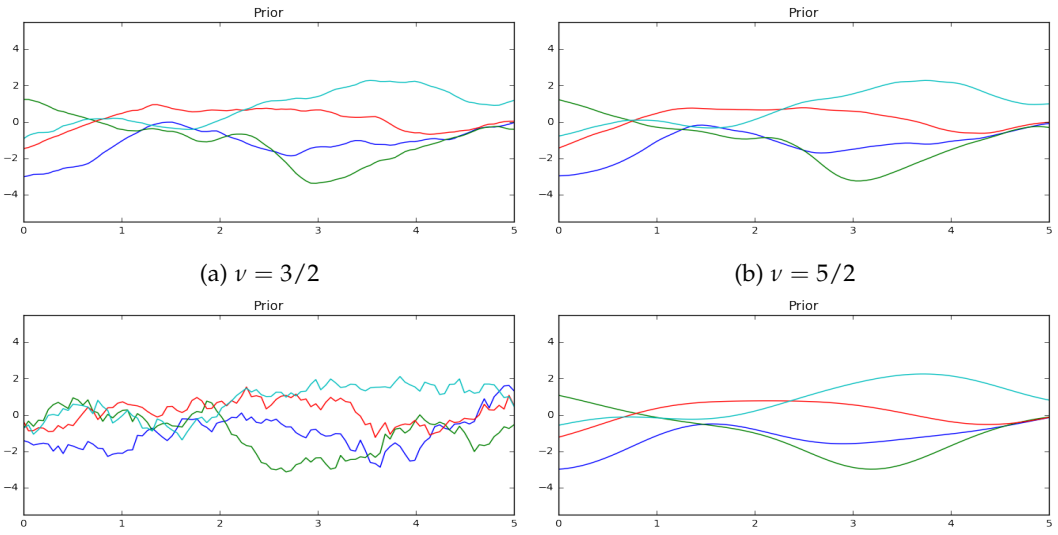
with positive parameters ν and ℓ, where K_{ν} is a modified Bessel function. [7]

It might not be immediate, but note that the scaling is chosen so that for $\nu \to \infty$ we obtain the SE covariance function. The most interesting cases for machine learning are $\nu = 3/2$ and $\nu = 5/2$, for which

$$k_{\nu=3/2}(\tau) = \left(1 + \frac{\sqrt{3}\tau}{\ell}\right)\exp\left(-\frac{\sqrt{3}\tau}{\ell}\right), \tag{2.6}$$

$$k_{\nu=5/2}(\tau) = \left(1 + \frac{\sqrt{5}\tau}{\ell} + \frac{5\tau^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}\tau}{\ell}\right) \tag{2.7}$$

Another special case is $\nu = 1/2$ known as the Laplacian covariance function which sample stationary Browninan motion.



(a) $\nu = 3/2$ (b) $\nu = 5/2$

(c) $\nu = 1/2$ (d) $\nu = 100$

Figure 2.2: GP priors generated with the Matérn kernel with different hyperparameters ν values. (c) For $\nu = 1/2$ the process becomes very rough. (d) For values of $\nu \geq 7/2$ is hard to distinguish between finite values of ν and $\nu \to \infty$, the smooth squared exponential case.

[7] The modified Bessel functions (also named the hyperbolic Bessel functions) of the first and second kind are defined by

$$I_{\alpha} = i^{-\alpha} J_{\alpha}(ix) = \sum_{m=0}^{\infty} \frac{1}{m!\Gamma(m+\alpha+1)}\left(\frac{x}{2}\right)^{2m+\alpha}$$

$$K_{\alpha}(x) = \frac{\pi}{2}\frac{I_{\alpha}(x) - I_{\alpha}(x)}{\sin(\alpha\pi)}$$

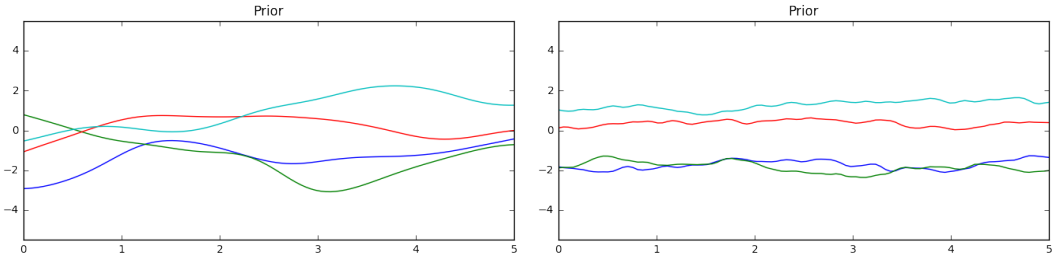
when α is not an integer; when α is an integer, then the limit is used.

The Rational Quadratic Covariance Function

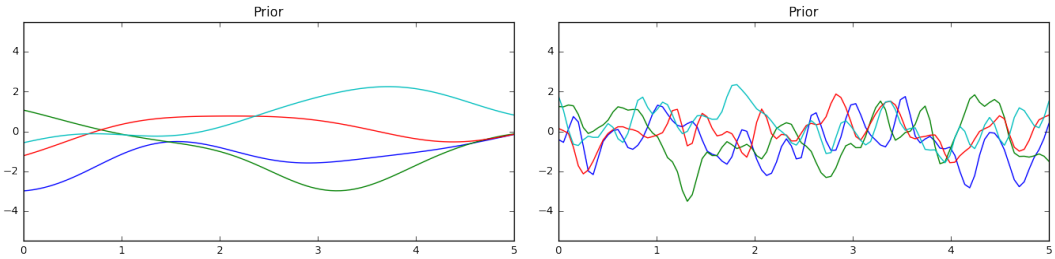
The Rational quadratic (RQ) covariance function is

$$k_{RQ}(\tau) = \sigma_f^2 \left(1 + \frac{\tau^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{2.8}$$

with α, $\ell > 0$ can be seen as an infinite sum of squared exponential (SE) covariance functions with different characteristic length-scales. So, GP priors with this kernel expect to see functions which vary smoothly across many lengthscales. The parameter α determines the relative weighting of large-scale and small-scale variations. When $\alpha \to \infty$, the RQ is identical to the SE. Note that the process is infinitely mean-squared differentiable for every α in contrast to the Matérn kernel.



(a) $(\sigma_f^2, \ell, \alpha) = (1.0, 1.0, 1.0)$ (b) $(\sigma_f^2, \ell, \alpha) = (1.0, 1.0, 0.01)$



(c) $(\sigma_f^2, \ell, \alpha) = (1.0, 1.0, 100)$ (d) $(\sigma_f^2, \ell, \alpha) = (1.0, 0.1, 40)$

Figure 2.3: Note that panel (c), since α value is high, is almost identical to figure 2.1a. In panel (d), as the value of the length-scale decreases the function becomes more wiggly as expected.

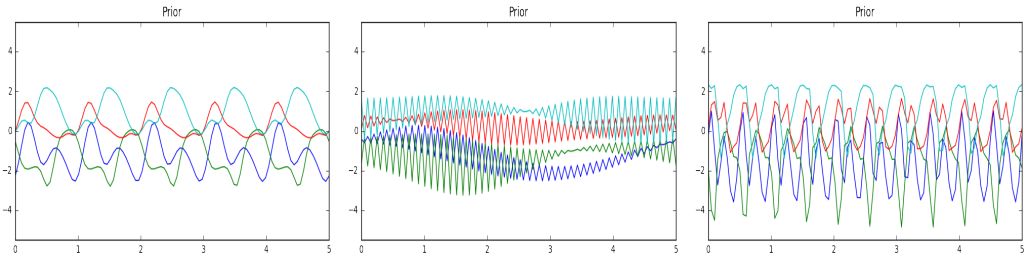
The Periodic Covariance Function

This kernel is useful when data has periodicity, therefore it allows one to model functions which repeat themselves exactly. The expression is given by

$$k_{Per}(x, x') = \sigma_f^2 \exp(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell^2}) \tag{2.9}$$

Its parameters are easily interpretable:

- The period p simply determines the distance between repititions of the function.
- The lengthscale acts in the same way as in the SE kernel.



(a) $(\sigma_f^2, \ell, p) = (1, 1, 1)$ (b) $(\sigma_f^2, \ell, p) = (1, 1, 0.1)$ (c) $(\sigma_f^2, \ell, p) = (2, 0.8, 0.5)$

Figure 2.4

Constant covariance function

Can be used as part of a product-kernel where it scales the magnitude of the other factor (kernel) or as part of a sum-kernel, where it modifies the mean of the Gaussian process.

$$k(x, x') = \sigma_0^2 \tag{2.10}$$

2.2.2 Dot Product Covariance Functions

Above we have seen examples of stationary kernels. However, there are also other interesting kernels which are not of this form. Being non-stationary means that the parameters of the kernel are about specifying the origin. In this section we describe and provide a graphical understanding of the linear kernel.

Linear Kernel

The linear kernel is just a particular case of the polynomial kernel, where the positive integer p given in the equation that follows is $p = 1$.

$$k_{Pol}(x, x') = (\sigma_b^2 + \sigma_v^2(x - c)(x' - c))^p \tag{2.11}$$

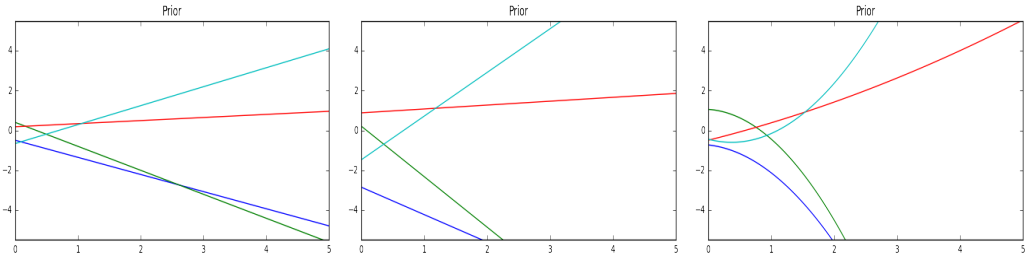
Where the hyperparameters means:

- The offset c determines the x-coordinate of the point that all the lines in the posterior go though. At this point, the function will have zero variance (unless you add noise).

-The constant variance σ_b^2 determines how far from 0 the height of the function will be at zero. It's a little confusing, because it's not specifying that value directly, but rather putting a prior on it. It's equivalent to adding an uncertain offset to our model.

-The hyperparameter p specifies the degree of the polynomial expression resulting from the product of p linear kernels. Therefore, if $p = 2$ the function sampled from GP prior will be quadratic, likewise if $p = 3$ cubic.

If you use just a linear kernel in a GP, you're simply doing Bayesian linear regression, and significantly improving the computation time, since instead of taking $\mathcal{O}(n^3)$ time, inference can be made in $\mathcal{O}(n)$. Indeed, combining it with other kernels gives rise to some nice properties, as it will be shown further on.



(a) $(\sigma_b^2, \sigma_v^2, c, p) = (1.0, 0.3, 0, 1)$ (b) $(\sigma_b^2, \sigma_v^2, c, p) = (2.0, 1.2, 0, 1)$ (c) $(\sigma_b^2, \sigma_v^2, c, p) = (1.0, 0.4, 0, 2)$

Figure 2.5: In panel (c) since $p = 2$, meaning that the kernel is the result of the product of two linear kernels, the prior is a quadratic function.

2.2.3 Other Non-stationary Covariance Functions

In this section we will describe the covariance function belonging to a particular type of neural network. The neural network kernel is perhaps most notable for research on Gaussian processes within the machine learning community. Its construction is due to Radford M. Neal who pursued the limits of large models, and showed that a Bayesian neural network becomes a Gaussian process with a neural network kernel as the number of units approaches infinity. Moreover, a brief explanation of White noise Kernel is provided.

Neural Network covariance function

Consider a neural network with one hidden layer

$$f(x) = b + \sum_{i=1}^{J} v_i h(x; \boldsymbol{u_i}). \tag{2.12}$$

v_i are the hidden to output weights, h is any bounded hidden unit transfer function, $\boldsymbol{u_i}$ are the input to hidden weights, and J is the number of hidden units. Let the bias b and the v's have independent zero-mean distribution of variance σ_b^2 and σ_v^2/J respectively, and the weights for each hidden unit $\boldsymbol{u_i}$ have independent and identical distributions.
The first two moments of $f(x)$ in equation (2.12), collecting all weights together into the vector $\mathbf{w}$, are

$$\mathbf{E}[f(x)] = 0 \tag{2.13}$$

$$cov[f(x), f(x')] = \mathbf{E}_w[f(x)f(x')] = \sigma_b^2 + \frac{1}{J}\sum_{i=1}^{J} \sigma_v^2 \mathbf{E}_u[h_i(x; \boldsymbol{u_i}) h_i(x'; \boldsymbol{u_i})] = \sigma_b^2 + \sigma_v^2 \mathbf{E}_u[h(x; \boldsymbol{u}) h(x'; \boldsymbol{u})] \tag{2.14}$$

where the last equality follows from the fact that the $\boldsymbol{u_i}$ are identically distributed. The sum in equation (2.14) is over J i.i.d random variables, and all moments are bounded. If b has a Gaussian distribution, the central theorem can be applied to show that as $J \to \infty$ any collection of function values $f(x_1), \ldots, f(x_N)$ has a Joint Gaussian distribution, and thus the neural network in equation (2.14) becomes a Gaussian process with covariance function given by the last term in (2.14).

If we choose the transfer function as $h(x; \boldsymbol{u}) = erf(u_0 + \sum_{j=1}^{P} u_j x_j)$, where $erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$, and we choose $\boldsymbol{u} \sim \mathcal{N}(0, \sum)$, then we obtain [8] from (2.14)

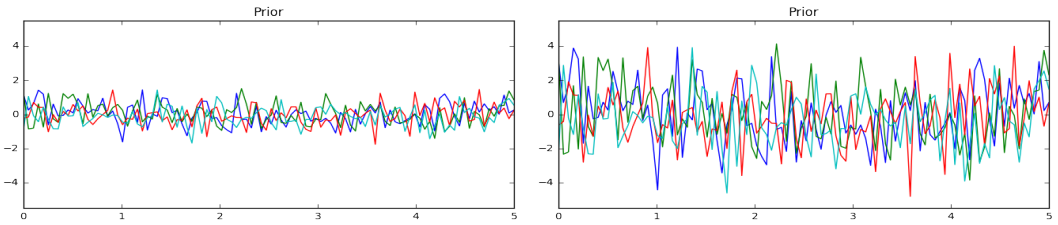
$$k_{NN}(x, x') = \frac{2}{\pi} sin\left(\frac{2\tilde{x}^T \sum \tilde{x}'}{\sqrt{(1 + 2\tilde{x}^T \sum \tilde{x})(1 + 2\tilde{x}'^T \sum \tilde{x}')}} \right) \tag{2.15}$$

where $x \in \mathbb{R}^P$ and $\tilde{x} = (1, x^T)^T$. This is a true neural network covariance function. On the other hand the sigmoid kernel $k(x, x') = \tanh(a + bx \cdot x')$, although being proposed is never positive definite and thus is not a valid covariance function.

White noise covariance function

The main use-case of this kernel is as part of a sum-kernel where it explains the noise-component of the signal. Tuning its parameter corresponds to estimating the noise-level.

$$k(x, x') = \sigma_f^2 \delta(x - x') \tag{2.16}$$



(a) ($\sigma_f^2 = 0.4$) (b) ($\sigma_f^2 = 3.0$)

Figure 2.6: GP priors drawn from a White Noise Kernel

[8] A detailed steps by step explanation is provided by Williams, C. K. I. (1998) in Computation with Infinite Neural Networks. Neural Computation.

Table 2.1: Summary of covariance functions

Covariance function	Expression	Stationary
Constant	$k(x,x') = \sigma_0^2$	$\surd$
Linear	$k_{Pol}(x,x') = (\sigma_b^2 + \sigma_v^2(x-c)(x'-c))$	
Polynomial	$k_{Pol}(x,x') = (\sigma_b^2 + \sigma_v^2(x-c)(x'-c))^p$	
Squared Exponential	$k_{SE}(x,x') = \sigma_f^2 exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	$\surd$
Matérn	$k_{Matern}(\tau) = \frac{2^{1-v}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu\tau}}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu\tau}}{\ell}\right)^{\nu}$	$\surd$
Rational Quadratic	$k_{RQ}(\tau) = \sigma_f^2\left(1+\frac{\tau^2}{2\alpha\ell^2}\right)^{-\alpha}$	$\surd$
Periodic	$k_{Per}(x,x') = \sigma^2 \exp(-\frac{2\sin^2(\pi\|x-x'\|/p)}{\ell^2})$	$\surd$
Neural Network	$k_{NN}(x,x') = \frac{2}{\pi} sin\left(\frac{2\tilde{x}^T \sum \tilde{x}'}{\sqrt{(1+2\tilde{x}^T \sum \tilde{x})(1+2\tilde{x}'^T \sum \tilde{x}')}}\right)$	
White Noise	$k(x,x') = \sigma_f^2 \delta(x-x')$	

2.3 Combining Kernels

In the previous section we have developed many covariance functions all of them summarized in Table 2.1. The kernels above are useful if data is all the same type, however this is not a common situation. Therefore, we must provide solutions if the kind of structure of our data is not expressed by any known kernel.

The next few sections of this chapter will explore ways in which kernels can be combined to create new ones with different properties. This will allow us to include as much high-level structure as necessary into our models.

Although being many ways of combining kernel, such as convolution [9] or tensor product, the more extended and the ones analyzed are addition and multiplication. [10]

[9] Consider an arbitrary fixed kernel $h(x,z)$ and the map $g(x) = \int h(x,z)f(z)dz$. Then $cov(g(x), g(x')) = \int h(x,z)k(z,z')h(x',z')dzdz'$

[10] For further explanation proving that the sum and the product of two kernels is a kernel see http://ttic.uchicago.edu/ dmcallester/ttic101-07/lectures/kernels/kernels

Sum and multiplication of kernels are usually written as:

$$k_1 + k_2 = k_1(x, x') + k_2(x, x') \tag{2.17}$$
$$k_1 \times k_2 = k_1(x, x') \times k_2(x, x') \tag{2.18}$$

2.3.1 Combining Kernels through multiplication

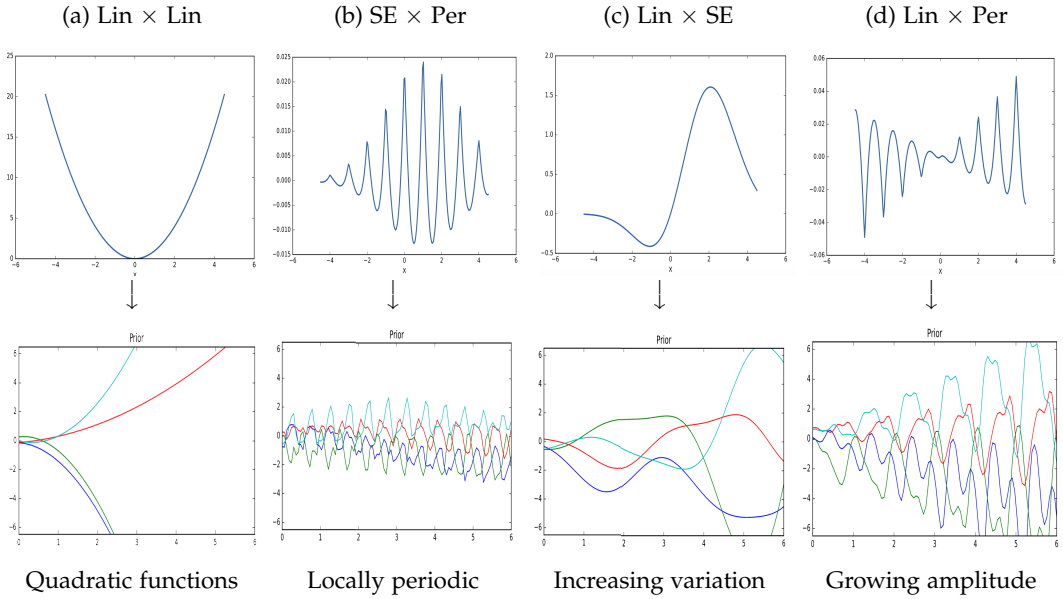
Each kernel in a product modifies the resulting GP model in a consistent way. Thus, multiplying kernels is an efficient way to produce kernels combining several high-level properties. But what properties do these new kernels have? Here, we discuss a few examples.

- **Multiplication by SE**: Converts any global correlation structure into local correlation since $SE(x, x')$ decreases monotonically to 0 as $|x - x'|$ increases. For example, Periodic kernel corresponds to exactly periodic structure, whereas $Per \times SE$ corresponds to locally periodic structure, as shown in the figure (2.7).

- **Multiplicating Linear Kernels**: By multiplying together n linear kernels, we obtain a prior on polynomials of degree n. As we have shown in the previous section in figure (2.5c).

- **Multiplication by a linear kernel**: It is equivalent to multiplying the function being modeled by a linear function. If $f(x) \sim GP(0, k)$ then $x \times f(x) \sim GP(0, Lin \times k)$, thus the marginal standard deviation of the function being modeled grows linearly away from the location given by kernel parameter c.

Figure 2.7: Examples of one-dimensional structures expressible by multiplying kernels. First row shows $k(x, x')$. The second one functions $f(x)$ sampled from GP prior.



2.3.2 Multi-dimensional models

Let's discuss briefly a way to model functions having more than one input. It consists on multiplying together kernels defined on each individual input. For example, a product of SE kernels over different dimensions each having a lengthscale parameter. Usually named as Automatic Relevance Determination (ARD) the particular case mentioned before would have the following expression

$$\text{SE-ARD}(x, x') = \prod_{d=1}^{D} \sigma_d^2 \exp\left(-\frac{1}{2} \frac{(x_d - x'_d)^2}{\ell_d^2} \right) = \sigma_f^2 \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{\ell_d^2} \right) \tag{2.19}$$

Note that this proceedure was named ARD since estimating the lengthscale parameters $\ell_1, \ell_2, \cdots, \ell_D$ implicitly determines the relevance of each dimension. Take into account that input dimensions with relatively large lengthscales imply relatively little variation along those dimensions in the function being modeled.

2.3.3 Modelling Sums of Functions

Let f_a, f_b be functions drawn independently from GP priors $f_a \sim GP(\mu_a, k_a)$ and $f_b \sim GP(\mu_b, k_b)$. Then the distribution of the sum of those functions is simply another GP:

$$f_a + f_b \sim GP(\mu_a + \mu_b, k_a + k_b) \tag{2.20}$$

As it is easy to encode additivity into GP models, note that kernels k_a and k_b can be of different types, allowing us to model the data as a sum of independent functions, each possibly representing a wide range type of structures.

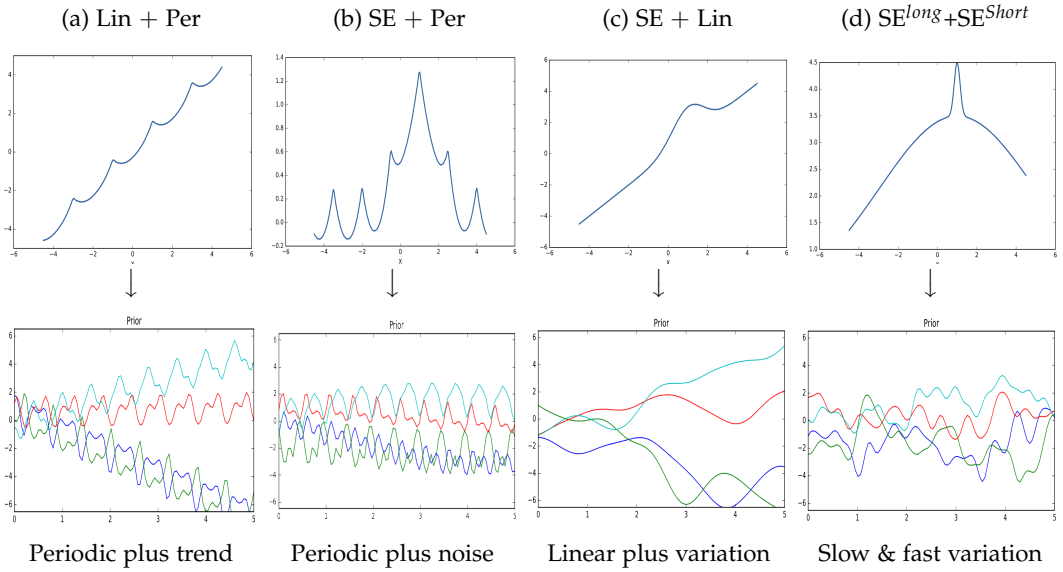


Figure 2.8: Examples of one-dimensional structures expressible by adding kernels. SE^{long} denotes a SE kernel whose lengthscale is long relative to that of SE^{Short}.

2.3.4 Changepoints

An example of how combining kernels can give rise to more structured priors is given by changepoint kernels, which can express a change between different types of structure. Changepoints kernels can be defined through addition and multiplication with sigmoidal functions such as $\sigma(x) = \frac{1}{1+\exp(-x)}$:

$$CP(k_1, k_2)(x, x') = \sigma(x)k_1(x, x')\sigma(x') + (1 - \sigma(x))k_2(x, x')(1 - \sigma(x')) \tag{2.21}$$

which can be written in shorthand as

$$CP(k_1, k_2)(x, x') = k_1 \times \sigma + k_2 \times \bar{\sigma} \tag{2.22}$$

where $\sigma = \sigma(x)\sigma(x')$ and $\bar{\sigma} = (1 - \sigma(x))(1 - \sigma(x'))$. This compound kernel expresses a change from one kernel to another. The parameters of the sigmoid determine where, and how rapidly, this change occurs.

Chapter 3

Model Selection and Adaptation of Hyperparameters

In chapter 1 it has been seen how to do regression using a Gaussian process with a given fixed covariance functions as well as in chapter 2 several examples of those covariance functions were presented. While some properties of them such as stationarity may be easy to determine from the context, it may not be trivial to specify with confidence the value of free hyperparameters, length-scales and variances. Therefore, the natural question that follows, and that will give rise to turn Gaussian processes into powerful practical tools, is how to develop methods that adress the model selection problem, refering to the discrete choice of the functional form for the covariance function as well as values for any hyperparameters.

In section 3.1 we outline the model selection for regression problems, focusing on Bayesian approach in section 3.1.1 and cross-validation, in particular the leave-one-out estimator, in section 3.1.2. In the remaining section Bayesian principles are applied to a specific case in order to provide a practical view.

3.1 Model Selection for GP Regression

Based on a set of training data, our task is to make inference about the form and parameters of the covariance function, in other words, about the relationships in the data. It should be clear that model selection is essentially open ended, meaning that even for a particular kernel there is a huge variety of possible distance measures. Thus we need to be able to compare two (or more) methods differing in values of particular parameteres, or the shape of the covaraince functions, or compare a GP model to any other kind of model.

In the next section we describe the Bayesian view on model selection, which involves the computation of the probability of the model given the data, based on the marginal likelihood.

3.1.1 Marginal Likelihood

To estimate the kernel parameters, we could use exhaustive search over a discrete grid of values, with validation loss as an objective, but this can be quite slow. Here we consider an empirical Bayes approach, which will allow us to use continuous optimization methods, which are much faster. In particular, we will maximize the marginal likelihood [1]

$$p(y|X) = \int p(y|f, X)p(f|X)df \tag{3.1}$$

Since $p(f|X) = \mathcal{N}(f|0, K)$ and $p(y|f) = \prod_i \mathcal{N}(y_i|f_i, \sigma_y^2)$ the marginal likelihood is given by

$$\log((p(y|X)) = \log \mathcal{N}(y|0, K_y) = -\frac{1}{2} y K_y^{-1} y - \frac{1}{2}\log|K_y| - \frac{N}{2}\log(2\pi) \tag{3.2}$$

where $K_y = K_f + \sigma_n^2 I$ is the covariance matrix for the noisy targets $\mathbf{y}$ and K_f is the covariance matrix for the noise-free latent f.

The first term of the marginal likelihood in equation 3.2 is a data fit term since it is the only one involving the observed targets. The second term, $\log|K_y|/2$, is the model complexity depending only on the covariance function and the inputs. Lastly, $\log(2\pi)/2$ is just a normalization constant.

To understand the tradeoff between the first two terms, consider a 1-dimensional SE kernel, as we vary the length scale ℓ and hold σ_y^2 fixed. For short length scales, the fit will be good, so $\boldsymbol{y}K_y^{-1}\boldsymbol{y}$ will be small. However, the model complexity will be high: $\boldsymbol{K_y}$ will be almost diagonal, since most points will not be considered near any others, so the $\log|\boldsymbol{K_y}|$ will be large. For long length scales, the fit will be poor but the model complexity will be low: $\boldsymbol{K_y}$ will be almost all 1's so $\log|\boldsymbol{K_y}|$ will be small.

Now, it is time to maximize the marginal likelihood in order to set the hyperparameters. We seek the partial derivatives of the marginal likelihood w.r.t the hyperparameters denoted by θ. Using 3.2 and the rules of the matrix derivatives [2] one can show that

$$\frac{\partial}{\partial\theta_j}\log p(y|X,\theta) = \frac{1}{2}y^T K_y^{-1}\frac{\partial K}{\partial\theta_j}K_y^{-1}y - \frac{1}{2}tr(K_y^{-1}\frac{\partial K_y}{\partial\theta}) = \frac{1}{2}tr\left((\boldsymbol{\alpha\alpha^T} - K_y^{-1})\frac{\partial K_y}{\partial\theta_j}\right) \tag{3.3}$$

where $\alpha = K^{-1}\boldsymbol{y}$. It takes $\mathcal{O}(n^3)$ time to compute K_y^{-1}, and $\mathcal{O}(n^2)$ time per hyperparameter to compute the gradient. Often there exists some constraints on the hyperparameters, such as $\sigma_y^2 \geq 0$. In this case, we can define $\theta = \log(\sigma_y^2)$, and then use the chain rule.

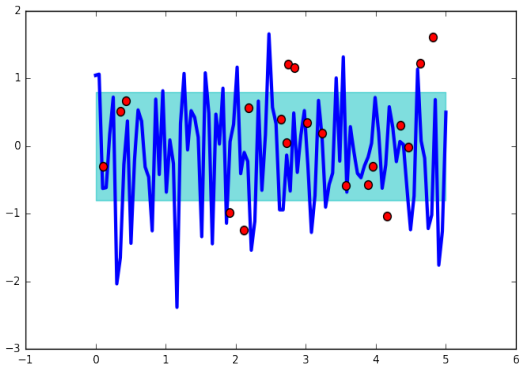
Although not being a devastating problem note that there is no guarantee that the marginal likelihood does not suffer from multiple local optima since the objective is not convex and every local maximum corresponds to a particular interpretation of the data.

[1] The reason it is called the marginal likelihood, rather than just likelihood, is because we have marginalized out the latent Gaussian vector **f**.

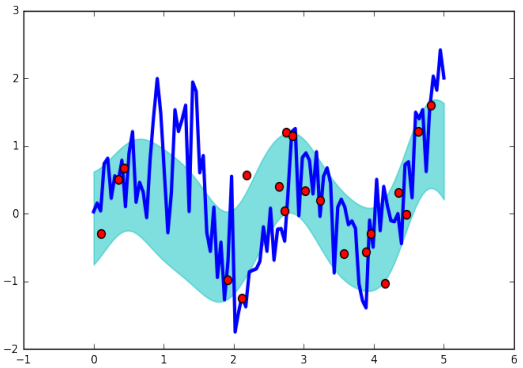
[2] $\frac{\partial}{\partial\theta}K^{-1} = -K^{-1}\frac{\partial K}{\partial\theta}K^{-1}$ where $\frac{\partial K}{\partial\theta}$ is a matrix of elementwise derivatives.
For the log determinant of a p.d. symmetric matrix we have $\frac{\partial}{\partial\theta}\log|K| = tr(K^{-1}\frac{\partial K}{\partial\theta})$.

In Figure 3.1 an example with two local optima is provided. For a data set of 20 observations randomly generated the inferred underlying functions have been specified considering to different sum-kernels. Despite the fact both of them result from the addition of a Squared Exponential plus a White noise kernel they differ in the values of the hyperparameters such as the lengthscale and noise level.

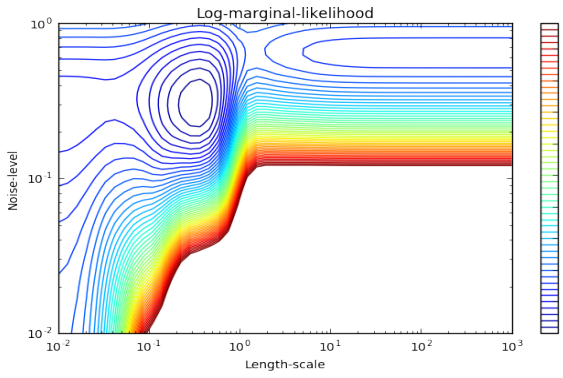
By doing so, one may expect, that even there exists differences in the kernels, the log-marginal-likelihood will converge to a global maximum after the optimitzation process. However the illustration of the log-marginal-likelihood (LML) landscape shows that there exist two local maxima of LML. The first corresponds to a model with a high noise level and a large length scale, which explains all variations in the data by noise. The second one has a smaller noise level and shorter length scale, which explains most of the variation by the noise-free functional relationship.



(a) Initial: $(\ell, \sigma_f^2, \sigma_n^2) = (100, 1, 1)$
Optimum: $(\ell, \sigma_f^2, \sigma_n^2) = (109, 0.00316, 0.637)$
Log-Marginal-Likelihood: -23.872337362



(b) Initial: $(\ell, \sigma_f^2, \sigma_n^2) = (1, 1, 1e-5)$
Optimum: $(\ell, \sigma_f^2, \sigma_n^2) = (0.365, 0.64, 0.294)$
Log-Marginal-Likelihood: -21.8050908902



(c)

Figure 3.1: Panels (a) and (b) show the underlying functions (and 95% confidence intervals). Panel (c) shows marginal likelihood as a function of the hyperparameters ℓ and σ_n^2.

3.1.2 Cross Validation

The basic idea of cross-validation is to split the training set into two disjoints sets, one used for training and the other used to monitor performance usually called the *validation set*. Having defined a dataset to test the model in the training phase shrinks problems like overfitting and variability and gives an insight on how the model will generalize to an independent dataset.

Although different types of cross-validation implementations can be distinguished, in the Gaussian processes context, the most extended are those that involves multiple rounds of cross-validations using different partitions and validations results are averaged over the rounds. Methods such k-fold cross-validation and leave-one-out cross validation(LOO-CV) belong to that particular framework.

In the k-fold CV approach the training set is splitted into k smaller sets. Then the model is trained using k-1 of the folds as training data and the resulting model is validated on the remaining part of the data. Finally, as mentioned before, the performance measure reported by k-fold cross-validation is the average of the values computed in the loop.

On the other side, LOO-CV is an extreme case of k-fold cross validation since the number of training cases $k = n$. Despite the fact that computational costs are prohibitive, there are computational shortcuts that allows LOO-CV to become an efficient way to tune the hyperparameters and model selection.

To begin with, since cross-validation can be used with any loss function, using the negative predictive log probabiity when leaving out training case i is

$$\log p(y_i|X, y_{-i}, \theta) = -\frac{1}{2}\log \sigma_i^2 - \frac{(y_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2}\log 2\pi \tag{3.4}$$

where the notation y_{-i} means all the targets except number i, the training sets are taken to be (X_i, y_i) and μ_i and σ_i^2 are computed according to (1.11) and (1.12) respectively. Thus, the LOO log predictive probability is

$$L_{LOO}(X, y, \theta) = \sum_{i=1}^{n} \log p(y_i|X, y_{-i}, \theta) \tag{3.5}$$

Note that in each of the n LOO-CV loops the inverse of the covariance matrix has to be computed in order to determine the mean and variance in eq. (1.11) and (1.12). However, in each rotation the expressions are almost the same since only a single column and row of the covariances matrix are removed. Therefore, applying matrix inversion using partitioning on the complete covariance matrix increases efficiency, since the expressions for the LOO-CV predictive mean and variance become

$$\mu_i = y_i - \left[K^{-1}y\right]_i / \left[K^{-1}\right]_{ii} \qquad \text{and} \qquad \sigma_i^2 = 1/\left[K^{-1}\right]_{ii}, \tag{3.6}$$

where the computational expense of these quantities is $\mathcal{O}(n^3)$ once for computing the inverse of K plus $\mathcal{O}(n^2)$ for the entire LOO-CV procedure. At this point, replacing the expressions of (3.6) into eq. (3.4) and (3.5) gives rise to a performance estimator that is possible to optimize w.r.t hyperparameters to do model selection.

But before giving an expression to the partial derivatives of L_{LOO} w.r.t the hyperparameters we need the partial derivatives of the LOO-CV predictive mean and variances from eq. (3.6) w.r.t the hyperparameters, and those are

$$\frac{\partial \mu_i}{\partial \theta_j} = \frac{\left[Z_j \boldsymbol{\alpha}\right]_i}{[K^{-1}]_{ii}} - \frac{\boldsymbol{\alpha_i} \left[Z_j K^{-1}\right]_{ii}}{[K^{-1}]_{ii}^2}, \qquad \frac{\partial \sigma_i^2}{\partial \theta_j} = \frac{\left[Z_j K^{-1}\right]_{ii}}{[K^{-1}]_{ii}^2} \tag{3.7}$$

where $\boldsymbol{\alpha} = K^{-1}\boldsymbol{y}$ and $Z_j = K^{-1}\frac{\partial K}{\partial \theta_j}$. Thus the partial derivatives of (3.5) obtained using the chain rule and eq. (3.7) are

$$\frac{\partial L_{LOO}}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial \log p(y_i|X, y_{-i}, \theta)}{\partial \mu_i} \frac{\partial \mu_i}{\partial \theta_j} + \frac{\partial \log p(y|X, y_{-i}, \theta)}{\partial \sigma_i^2} \frac{\partial \sigma_i^2}{\partial \theta_j} = \tag{3.8}$$

$$\sum_{i=1}^{n} \left(\alpha_i \left[Z_j \boldsymbol{\alpha}\right]_i - \frac{1}{2}\left(1 + \frac{\alpha_i^2}{[K^{-1}]_{ii}}\right) \left[Z_j K^{-1}\right]_{ii} \right) / \left[K^{-1}\right]_{ii} \tag{3.9}$$

where the computational complexity is $\mathcal{O}(n^3)$ for the inverse of K and $\mathcal{O}(n^3)$ for hyperparameter for the derivative equation above.

Having reached this stage, one natural question to ask is under which circumstances which of the methods discussed, marginal likelihood or LOO-CV, might be preferable since their computational complexity is very similar. In the following sections as in the bank account balance forecasting problem model selection and adaptation of hyperparameteres has been done using marginal likelihood for programming convenience and widespread use in machine learning community.

3.2 Mauna Loa Atmospheric Carbon Dioxide Example and Discussion

This example is based on Section 5.4.3 of 'Gaussian Processes for Machine Learning' [Rasmussen & C.K.I Williams] and it represents a clear example of complex kernel and hyperparameter optimization using gradient ascent on the log-marginal-likelihood.

The data consists of the monthly average atmospheric CO_2 concentrations (in parts per million by volume (ppmv)) collected at the Mauna Loa Observatory in Hawaii, between 1959 and 2003. The data is shown in figure 3.2. The objective is to model the CO2 concentration as a function of the time x. Thus, we are working in a time series environment.

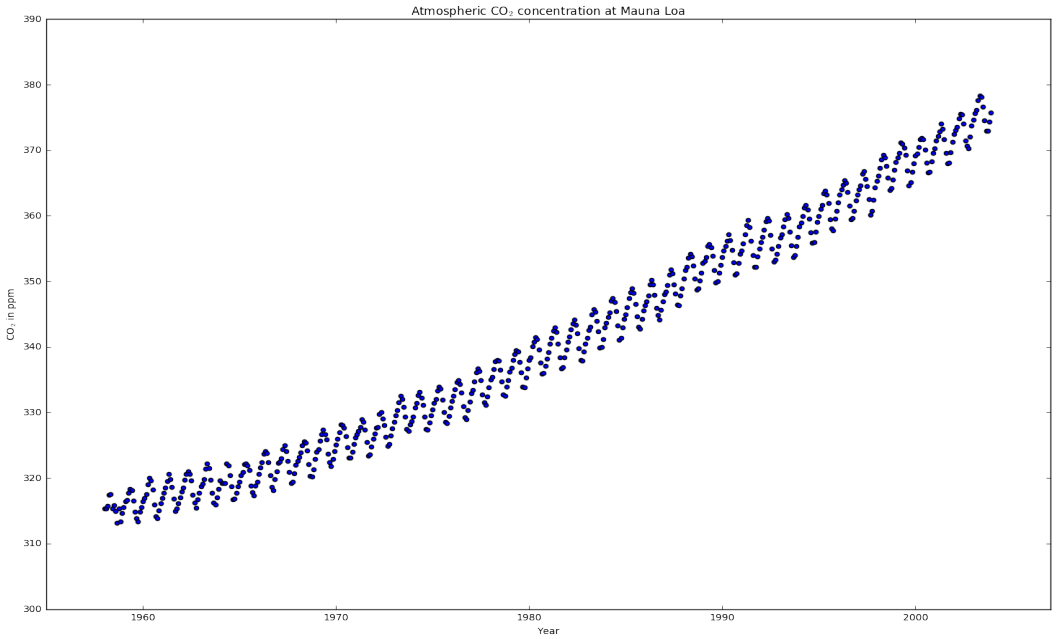


Figure 3.2: The 540 observations of monthly CO_2 concentration made between 1959 and 2003. Three missing values were replaced by the mean concentracion of the year. Note the rising trend and seasonal variations.

A reasonable way to start the modelling process is identifying the different components of our data. Classic deterministic analysis of time series suggests that every function should be decomposed in all or some of this features: trend, seasonal variations, term irregularities and noise. Once this first step is accomplished, the kernel and, therefore, hyperparameter selection and combination should take care of these individual properties in order to provide an excellent fit to the data.

In our particular case Mauna Loa dataset presents a pronounced long term rising trend that we try to model using a squared exponential kernel with two hyperparameters controlling the variance θ_1 and characterisitc length-scale θ_2

$$k_1(x, x') = \theta_1 \exp\left(-\frac{(x - x')^2}{2\theta_2^2} \right) \tag{3.10}$$

The SE kernel with a large length-scale enforces this component to be smooth, note that it is not enforced that the trend is rising which leaves this choice to the GP.

On the other side, the seasonal component [3] seems also evident. We can use the periodic kernel with a fixed periodicity of 1 year. The length-scale θ_5 of this periodic component, controlling its smoothness, is a free parameter and θ_3 gives the magnitude. In order to allow decaying away from exact periodicity, the product with an SE kernel is taken.

[3] Seasonal component is caused by different concentrations of plants.

The length-scale θ_4 of this SE component controls the decay time and is a further free parameter.

$$k_2(x, x') = \theta_3^2 \exp\left(-\frac{(x - x')^2}{2\theta_4^2} - \frac{2\sin^2(\pi(x - x'))}{\theta_5^2} \right) \tag{3.11}$$

In order to capture the medium term irregularities a rational quadratic term is used. These irregularities can better be explained by a Rational Quadratic than a SE kernel component, since it gives lower marginal likelihood as it can accommodate several length-scales.

$$k_3(x, x') = \theta_6^2 \left(1 + \frac{(x - x')^2}{2\theta_7^2 \theta_8} \right)^{-\theta_8} \tag{3.12}$$

As before θ_6 is the variance, the length-scale θ_7 and θ_8 (also known as α parameter, which determines the diffuseness of the length-scales) are to be determined.

The last feature to model, the noise term, has been specified using an SE kernel, which explains the correlated noise components such as local weather phenomena, and a White Noise kernel

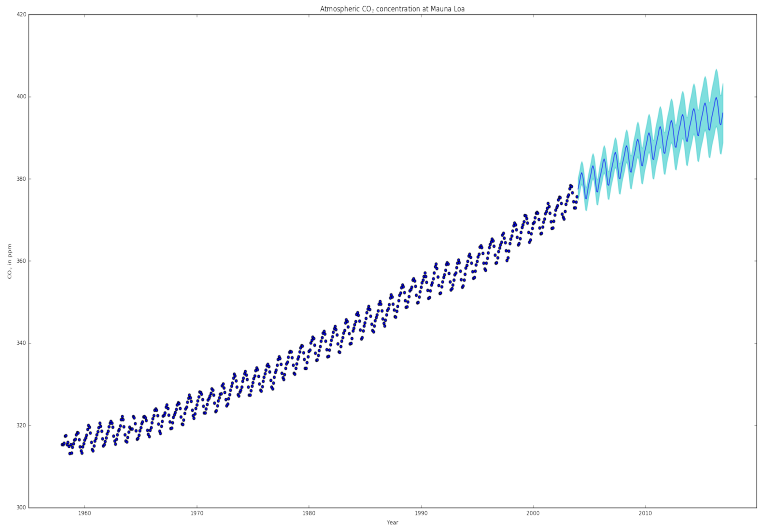
$$k_4(x, x') = \theta_9^2 \exp\left(-\frac{(x - x')^2}{2\theta_2^{10}} \right) + \theta_{11}^2 \delta_{pq} \tag{3.13}$$

where θ_9 is the magnitude of the correlated noise component, θ_{10} is its length-scale and θ_{11} is the variance of the independent noise component.
To sum up, the final covariance function is

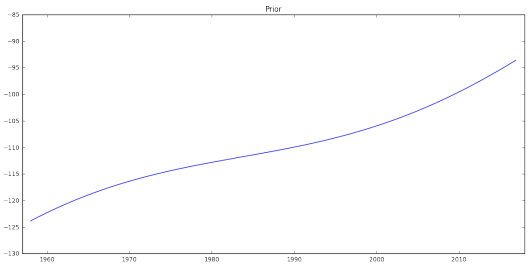
$$k(x, x') = k_1(x, x') + k_2(x, x') + k_3(x, x') + k_4(x, x') \tag{3.14}$$

with hyperparameters $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_{11})^T$. Before fitting the hyperparameters by optimizing the marginal likelihood using conjugate gradient optimitzer as BFGS[4] algorithm we substract the empirical mean of the data. To avoid bad local minima and to improve the global performance a few random starts are tried as well as k-cross validation on training data with $k = 3$. The best marginal likelihood result was $\log p(y|X, \theta) = -77.146$ that gives rise to an $R^2 = 0.9455$ score on the predictions.
The plot containing the final model and the predictions together with its decomposition into additive components is shown in figure 3.3.

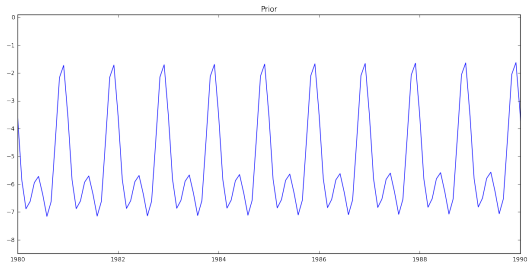
[4]The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is an iterative method for solving unconstrained nonlinear optimization problems.



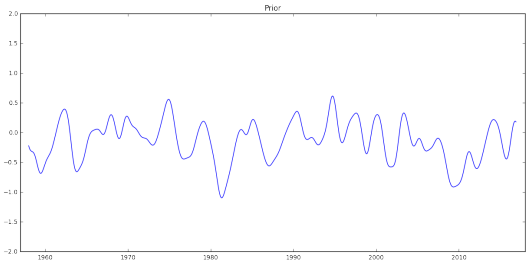
(a) The full posterior on the Mauna Loa dataset



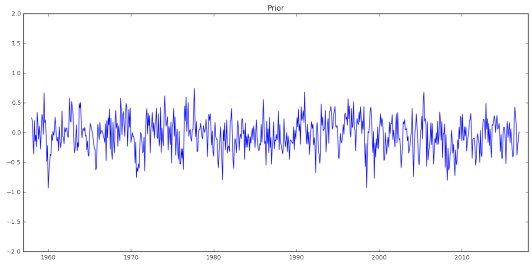
(b) Long term smooth rising trend



(c) Seasonal variation over the years



(d) Medium term irregularities



(e) White Noise

Figure 3.3: Panel (a) shows the underlying function together with 95% predictive confidence region for predictions until 2017. The confidence interval gets wider as the time increases. In Panels (b),(c),(d),(e) some components capture slowly-changing structure while others capture quickly-varying structure.

To sum up, Mauna Loa has been a good way to test the powerful possibilities that inference in composite covariance functions provides, as well as the utility to allow the data to determined the hyperparameters. The experience acquired facing this problem enables to confront the main project of this work, the bank account forecasting problem, with a more solid background.

Chapter 4

Bank account forecasting Problem

In this final chapter, since the main theoretical contributions and an introductory example as Mauna Loa has been covered, we are going to face a more complex problem that consists of forecasting bank accounts balance using Gaussian Processes.

Nowadays, with the advance of affordable, fast computation the machine learning community has adressed increasingly large and difficult problems and one appealing area that can be applied is banking. Even though, it is a extremely regulated environment it offers many possibilities and it becomes extremely exciting when working with massive amounts of financial data and how to deal with them in order to produce new tools.

In that context, anticipating the behaviour of personal accounts or predicting people's expenses is a challenge that holds a special place in any modern platform of intelligent banking services. In that particular exercise, we have 3.35 milion accounts[1] and we are trying to predict for each client the 13th state of the balance account with only having 12 historical values. More precisely we work on just a year-worth of data with monthly aggregations and the 13th is our forecasting goal.[2]

As mentioned in the introduction a prestigious European bank tackled that problem in order to provide their customers with a warning system in case the balance of their bank accounts has high probabilities to decrease significantly or even be negative.

Therefore, we are facing a time-series[3] regression problem. However, it is important to note that since our time series are very short statistical methods such as GARCH, ARIMA or Holt-Winters do not perform accurately. This led us to the next question: Can Gaussian Processes predict accurately the 13th state of a bank account given the 12th previous ones?

To answer that question we are going to approach the problem in two different ways. First,

[1] For computational limitations only 100.000 accounts will be analyzed as they are a representative sample. Examine the hole sample can be considered as a very interesting Big Data problem.

[2] The anonymized bank account data comes from real customers and is provided by a Bank whose name would not be disclosed.

[3] A time series is a doubly infinite sequence $\cdots, X_{-2}, X_{-1}, X_0, X_1, X_2, \ldots$ of random variables or random vectors. We refer to the index t of X_t as time and think of X_t as the state or output of a stochastic system at time t. The variable X_t is assumed to be real valued and for our purposes $t = 1, \ldots, 13$.

all the bank accounts will be analyzed as a global entity, meaning, that one main kernel would be considered to perform the Gaussian Process regression. On the other hand, in the second approach, in order to be more flexible and adaptive the bank accounts will be clustered regarding its similarity. Thus, for every cluster a different and more appropiate kernel can be considered. The details about the clustering process such as the similarity measure, the choice of centroids and kernels will be explained in the Clustering section.

To sum up, the questions to answer are whether or not GPR works as expected and if clusteritzation processes improve the performance of GPR. Thus, a comparative evaluation of the both approacches will be shown.

4.1 Global approach

Data description

Before starting the forecasting process we will focus a bit more on the structure of the data. We have at our disposal 3.35 million of bank accounts. Each account x_i is a vector of 1 row and 13 columns, where the first 12 columns represent the last 12 monthly states of the account and the 13th state is our goal prediction. For example, the first of our accounts is:

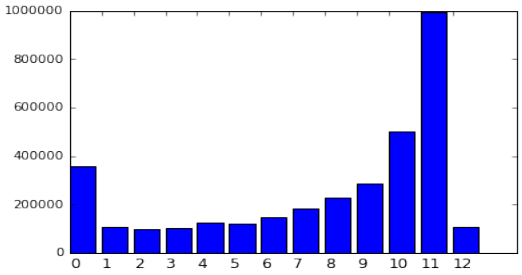
$$x_1 = [-3000, -5000, -3000, -5500, 0, -4000, -2000, -2000, 0, -5000, 0, -2000]$$

$$x_{1*} = [-4200]$$

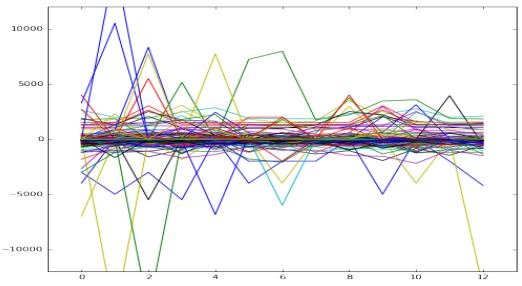
We actually know the value of the 13th state in order to evaluate the error of our forecasting. Thus, more formally our dataset can be expressed as $\mathcal{D} = \{(x_i, y_j)$ for $i = 1, \ldots, 3.351.168$ and $j = 1, \ldots, 13\}$, or alternatively can be thought as a matrix of 3.351.168 rows and 13 columns.

Given the size of the dataset we performed some data cleaning to reduce the number of rows. Firstly, we drop all the accounts whose 12th first states were zeros. From 3.35 million accounts 107.245 were dropped. Since 3.243.923 accounts are still remaining and the computational expenses[4] are unaffordable, we consider two random simple samples of 25.000 and 100.000 accounts. The reason why we take that samples size is because in the clustering the 25.000 accounts will be distributed in 20 groups and the 100.000 in 40, matching with all the different patterns observed and checking if more clusters mean more accuracy.

[4]Moreover, the forecasting process for 100.000 lasts more than 36 hours.



(a) Histogram showing the distribution of zeros over the 12 firsts states of the accounts.



(b) 400 bank accounts

Figure 4.1: Note that the accounts tend to have many zeros over the different months.

Forecasting

As mentioned above two different random simple samples have been considered ($N_1 = 25.000$ and $N_2 = 100.000$ accounts). Both have been analyzed in terms of the same kernel function, which has been designed to adapt and fit well with the general properties of the accounts. Note that since we have to be able to model a wide range of different behaviour patterns, the properties expressed by kernels can not be very specific or particular. Thus, the kernel function is derived by combining several different kinds of simple covariance functions. The final covariance function is

$$k(x, x') = k_1(x, x') + k_2(x, x') + k_3(x, x') \tag{4.1}$$

where

$$k_1(x, x') = \theta_1^2 \exp\left(-\frac{(x - x')^2}{2\theta_2^2} \right) = 450^2 \exp\left(-\frac{(x - x')^2}{2 * 75.0^2} \right) \tag{4.2}$$

To model the long term smooth trend we use a SE covariance function with a high amplitude paramater θ_1 since the variance is high and a large characteristic length-scale θ_2.
To model the medium term irregularities a Rational Quadratic term is used:

$$k_3(x, x') = \theta_3^2 \left(1 + \frac{(x - x')^2}{2\theta_4^2 \theta_5} \right)^{-\theta_5} = 5.5^2 \left(1 + \frac{(x - x')^2}{2 * 2.0^2 * 1.5} \right)^{-1.5} \tag{4.3}$$

One could have also used a SE form for this component, but it turns out that rational quadratic works better. Finally we specify a noise model as the sum of a SE and WN kernel. Noise in the series models rare behaviour changes caused by unexpected expenditures.

$$k_4(x, x') = \theta_6^2 \exp\left(-\frac{(x - x')^2}{2\theta_7^2} \right) + \theta_8^2 \delta_{pq} = 4.5^2 \exp\left(-\frac{(x - x')^2}{2 * 0.3^2} \right) + 20.2^2 \delta_{pq} \tag{4.4}$$

Note that we have not considered a periodic covariance function since it was not clear that all the accounts have a seasonal trend and least of all exactly periodic[5]. Actually, some trials have been done considering a seasonal kernel, but it turns out not to be relevant since a decay-time parameter became very large.

The hyperparameters were fitted by optimizing the marginal likelihood using a conjugate gradient optimizer as BFGS, likewise we have proceeded in the Mauna Loa problem. Moreover, for each account we have considered a k-cross validation for $k = 3$ since it gives rise to more accurate predicitons.

Before showing the predictions and the results achieved we must define the prediction metrics or at least, taking into account the final goal of the forecasting, the measure accuracy that allows to grade the 13th state of the bank account forecasted as good or poor. Some of the Measures of Forecast Accuracy used are the regulars in the statistics field while others were specially designed for that particular problem. Therefore we considered

- **Mean Absolute Error**(MAE): MAE = $\frac{1}{N}\sum_{i=1}^{N}|\hat{y}^{(i)} - y^{(i)}|$, where $\hat{y}^{(i)}$ is the predicted value and $y^{(i)}$ is the actual one.

- **Root Mean Squared Error** (RMSE): RMSE = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})^2}$

- **Mean Absolute Scaled Error** (MASE) :

$$MASE = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}^{(i)} - y^{(i)}|}{\frac{1}{T-1}\sum_{t=2}^{T}|y_t^{(i)} - y_{t-1}^{(i)}| + \lambda}$$

  One interesting and favorable property of MASE when compared to other methods for calculating forecast errors is the interpretability, since values greater than one indicate that in-sample one-step forecasts from the naïve method perform better than the forecast values under consideration. Thus, values near zero means a good accuracy compared to naïve method. For our particular case $T = 12$ and $\lambda \neq 0$ is a constant parameter to avoid zero division.

- **Percentage of sign accounts forecasted correctly** (SAFC):

$$SAFC = \frac{\text{Number of sign accounts predicted correctly}}{N}$$

  Given the initial aim of the project driven by the bank this index can be seen as a intuitive measure of accuracy. If the sign of the bank account balance forecasted matches with the 13th actual state of the account a counter is increased. The following line of code expresses that idea.

```
if(accountsreg[i][12] < 0 and ypred < 0) or (accountsreg[i][12] > 0 and ypred >
0) or (np.abs((accountsreg[i][12] - ypred)) <= 1e - 1)
```

[5]However, indeed some of the accounts behave seasonaly and in the clustering the kernel will capture that.

- **Percentage of sign accounts forecasted correctly including 0** (SAFC0):

$$SAFC0 = \frac{\text{Num. of sign accounts predicted correctly } \cup \{0\}}{N}$$

Similar to the measure mentioned above but with one important modification. Throughout the project we realize that in a significant amount of bank accounts the 13th balance state was 0. However, our forecast in a not negligible number of accounts that fulfill the situation mentioned in the sentence above was very close to 0 but not that specific value. Thus we reformulate the indicator SAFC in order to adjust that situation changing the code:

$$if(accountsreg[i][12] <= 0 \ and \ ypred < 0) \ or \ (accountsreg[i][12] >= 0 \ and \ ypred > 0) \ or \ (np.abs((accountsreg[i][12] - ypred)) <= 1e-1)$$

- **Percentage of sign accounts forecasted correctly including an uncertainty zone around 0** (SAFCcor):

$$SAFCcor = \frac{\text{Num. of sign accounts predicted correctly} + (\ y^{(i)} = 0 \ \wedge \ \hat{y}^{(i)} \in [-5,5])}{N}$$

The definition of SAFC0 and specially the inclusion of 0 makes sense for the type of bank accounts we are facing. However we can provide a more accurate measure since SAFC0 takes as a good forecast, for example, a prediction of 350 when the actual value is 0. Thus, we define SAFCcor that evaluates the number of accounts whose sign predicition matches with the actual 13th state and instead of including 0 (which seems quite gullible) we also consider as correct those predictions whose final value was 0 but the forecasted value was not, but it fits in a small uncertainty zone around zero (we define a bounded zone around zero from -5 to 5). For example, a prediction of 2.5 when the actual value is 0 would be considered as a good forecast.

- **95% confidence intervals** :

$$\frac{\text{Number of } \ y^{(i)} \in [\hat{y}^{(i)} - 1.96 \cdot \sqrt{var(\hat{y}^{(i)})}, \ \hat{y}^{(i)} + 1.96 \cdot \sqrt{var(\hat{y}^{(i)})}]}{N}$$

Since for every predicition we have access to its 95% confidence interval we can compute the percentage of predicitons that fit in that interval.

Results

	MAE	RMSE	MASE	SAFC	SAFC0	SAFCcor	95% conf. inter.
N = 25.000 accounts	150.6495	1897.5950	0.92243	65.644%	91.724 %	81.464%	88.288%
N = 100.000 accounts	159.9937	5389.1124	0.71326	64.826 %	91.409 %	81.058%	88.233%

After many trials, specially in order to decide the structure of the kernel and the initial values of the hyperparameters, the final performance is the one presented above. On the whole, the results are cautiously positive, however it is obvious that there is still room for improvement and hopefully it will be achieved in the clustering approach.

MASE for both cases is less than one, meaning that GP perform better than naïve method. Moreover SAFC measure and mainly SAFC0 show very good accuracy achieving more than 91% of good sign predicitions (as it includes zeros). At the same time SAFCcor is around 81%, meaning that the gap between SAFC0 and SAFCcor is approximately 10%. Consequently a small percentage of the 0 includes as a good predictions in SAFC0 corresponds to predictions that actually are distant from 0.

On the negative side, the percentage of accounts whose predicted value is in the 95% confidence interval is much less than expected since it barely overcomes 88%. This is not excessively worrying and concerning because that value is not quite far from 95% but as the definition of confidence interval states and given the fact that the uncertainty region is wide we expect that in the clustering approach the results will improve. From a strategic and business perspective, this index is extremely critical seeing as the goal of bank was to provide a warning system able to predict when the balance of the bank account will be negative or a drastic decrease of the balance can occur and thus send a message to the customer. Obviously, the customer might feel confused, upset or ashamed if receiving the alarm message the balance of the bank account remains without great changes. Thus, the bank expects to be highly sure and in this way be a powerful aid instead of disturbing customers. In the next section we decided to cluster the accounts with the aim of improving the results.

4.2 Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in another group. For our particular problem, classification methods such as SVMs and Naive Bayes would not be a good choice since they assume that the input features are independent. Simultaneously, the k-NN algorithm could still work, however it relies on the notion of a similarity measure between input examples. Thus, how do we measure the similarity between two time series?

At first, one might think that simply calculating the Euclidean distance between two time series would give us a good idea of the similarity between them. Given two time series of length n, $\{X_t : t = 1, \ldots, n\}$ and $\{Y_t : t = 1, \ldots, n\}$ we define the Euclidean distance between them as:

$$dist(X_t, Y_t) = \sqrt{\sum_{t=1}^{n} (X_t - Y_t)^2} \tag{4.5}$$

After all, the Euclidean distance between identical time series is zero and between very different time series is large. However, before we settle on Euclidean distance as a simi-

larity measure we should clearly state our desired criteria for determining the similarity between two time series.

With a good similarity measure, small changes in two time series should result in small changes in their similarity. With respect to Euclidean distance this is true for changes in the y-axis, but it is not true for changes in the time axis. This is the problem with using the Euclidean distance measure. It often produces pessimistic similarity measures when it encounters distortion in the time axis. The way to deal with this is to use dynamic time warping.

4.2.1 Dynamic Time Warping

Dynamic time warping (DTW) finds the optimal non-linear alignment between two time series. Thus, the Euclidean distances between alignments are then much less susceptable to pessimistic similarity measurements due to distortion in the time axis.

Considering to time series, $\{X_t : t = 1, \cdots, n\}$ and $\{Y_t : t = 1, \ldots, n\}$ of the same length n Dynamic time warping works in the following way. The first thing is to construct an $n \times n$ matrix where the element (i, j) is the Euclidean distance between X_i and Y_j. The goal is to find a path through this matrix that minimizes the cumulative distance. This path then determines the optimal alignment between the two time series. It should be noted that it is possible for one point in a time series to be mapped to multiple points in the other time series.

Let M be the path. $M = m_1, m_2, \ldots, m_k$ where each element of M represents the distance between a point X_i and Y_j. The optimal path, the one with the minimum Euclidean distance, given by the expression $M^* = \text{argmin}_m \left(\sqrt{\sum_{i=1}^{k} m_i} \right)$ is found via dynamic programming, specifically applying the following recursive function

$$\gamma(i, j) = dist(X_i, Y_j) + min(\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)) \tag{4.6}$$

The only important drawback is that dynamic time warping is quadratic in the length of the time series used. Thus, having two time series where n is the length of the first one and m is the length of the second gives rise to a complexity of $\mathcal{O}(nm)$. Therefore since we are performing DTW multiple times and this can be prohibitively expensive we speed things up using a lower bound of DTW known as LB Keogh which is defined as:

$$LBKeogh(X_t, Y_t) = \sum_{t=1}^{n} (Y_t - U_t)^2 I(Y_t > U_t) + (Y_t - L_t)^2 I(Y_t < L_i) \tag{4.7}$$

where U_t and L_t are upper and lower bounds for time series X_t which are defined as $U_t = \max(X_{t-r}, \ldots, X_{t+r})$ and $L_t = \min(X_{t-r}, \ldots, X_{t+r})$ for a reach r and I is the indicator function.[6]

[6]The indicator function of a subset A of a set X is a function $\mathbf{I}_A : X \to \{0, 1\}$ defined as $\mathbf{I}_A(x) := \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$

Another way to speed things up when using DTW is to enforce a locality constraint. This works for long time series under the assumption that it is unlikely for two elements to be matched if they are too far apart. The threshold is determined by a window size usully noted as w . This way, only mappings within this window are considered which speeds up the inner loop. Since the time series involved in our problem are quite short this second speed up was not required.

In the next subsection the clusteritzation conducted in the bank account forecasting problem is discussed in detail.

4.2.2 Clustering of Bank Accounts

Once a reliable method to determine the similarity between two time series is already specified everything is set to begin the process of clustering. The purpose, as mentioned above, is to lump together the bank accounts that behave the same way in order to specify a kernel function that gathers better the properties of the accounts in the cluster and be able to demonstrate if clustering improves the performance of GPR.

After a deep data exploration with the aim of identifying the most common behaviour patterns and since the number of clusters has to be chosen manually, two clusteritzation processes have been performed. The first one, grouping 25.000 accounts in 20 different clusters and the second one 100.000 in 40. [7]

For both approaches the process of clustering begins taking a simple and representative random sample of the bank accounts. More precisely, 10.000 for the 20 clusters and 15.000 for the 40 clusters. Then, again randomly taken, 20 and 40 accounts are specified as a centroids. Therefore, through all the 10.000 and 15.000 accounts initially taken a search is performed in order to find the centroid that minimizes the DTW distance with the account, or in other words, to find the centroid that is more similar to the account analyzed.

Given that DTW is quadratic, this can be computationally expensive. Thus, since LB Keogh lower bound is linear we can speed up classification. Note that given two time series X_t, Y_t it is always true that $LBKeogh(X_t, Y_t) \leq DTW(X_t, Y_t)$. Hence we can eliminate time series that cannot possibly be more similar than the current most similar time series. In this way we are eliminating many unnecessary DTW computations.

After all the accounts have been assigned to a cluster, we recalculate the centroid of each cluster finding the account that minimizes the global distance between all the accounts in the cluster. The main goal is to provide a representative centroid for each cluster. Consequently given the need to determine whether a new account belongs to a cluster or another we only need to compute the DTW distance between the account and the centroids. Figure 4.2 shows a graphical representation of the clustering process performed in the first approach (20 clusters).

[7] Given the computational limitations we cloud not performe a clustering involving 100 groups of more. Thus, it can be understand as a future exciting exercise.

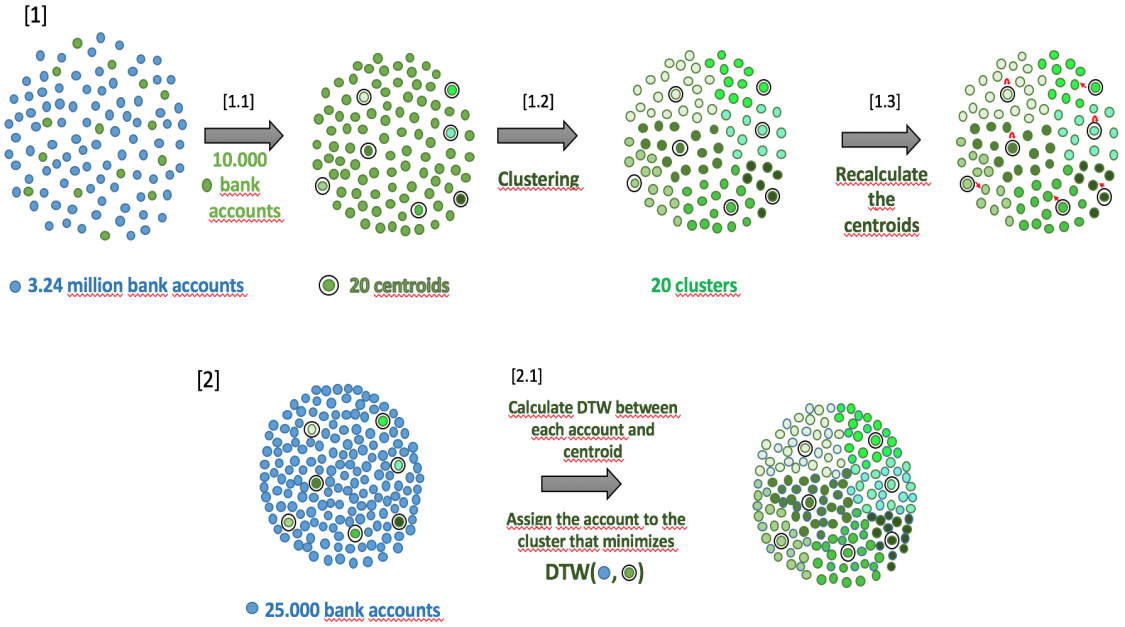
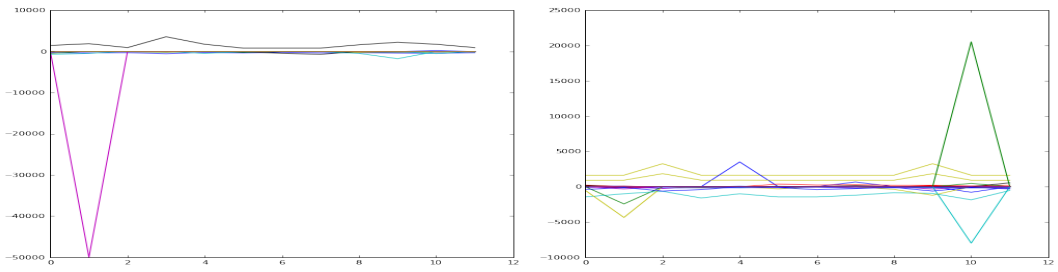


Figure 4.2: Clustering of 25.000 bank accounts in 20 clusters. **[1] Selection process of representative centroids.** Take a simple random sample of 10.000 bank accounts. [1.1] Again randomly, specify 20 centroids. [1.2] Compute DTW distance between the 10.000 accounts and the centroids. Cluster the accounts. [1.3] Recalculate de centroids finding the account that minimizes the global distance between all the accounts of the cluster. **[2] Clusteritzacion of the accounts.** Take a simple random sample of 25.000 bank accounts. [2.1] DTW distance between the accounts and the 20 centroids found in [1]. Then the accounts are assigned to the cluster that minimizes distance between centroid and account.

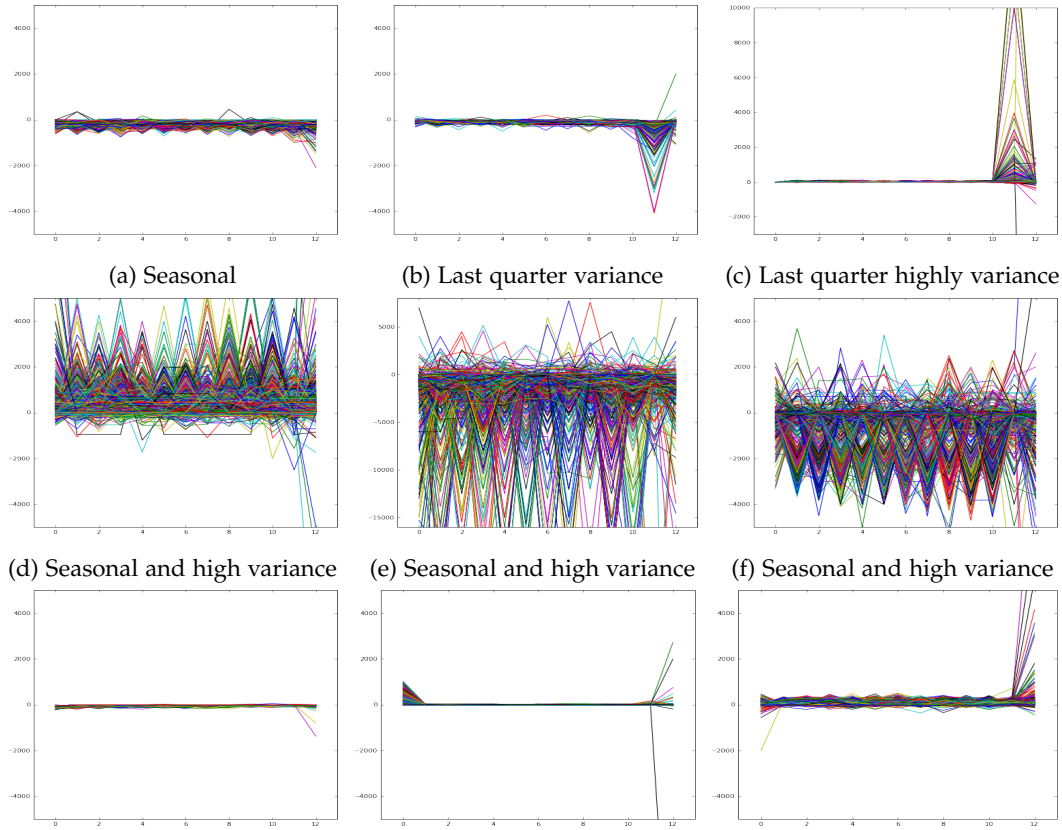
Note that the clustering of 100.000 bank accounts in 40 clusters is done using the same mechanism as shown above. However, instead of taking 10.000 accounts in order to determine the centroids, the sample size was 15.000. The figures below show the centroids found in both approaches.



(a) 20 centroids (b) 40 centroids

Figure 4.3: Plot showing the 12 historical values of the different centroids. Figure(a) shows the 20 centroids found for the first approach. In figure(b) 40 centroids are plotted.

Finally, some images of accounts belonging to the same cluster are provided. Many of that clusters include accounts that are extremely similar, meaning that even DTW provides a non-linear alignment the similarity between the accounts in the cluster is remarkably high.



(a) Seasonal (b) Last quarter variance (c) Last quarter highly variance

(d) Seasonal and high variance (e) Seasonal and high variance (f) Seasonal and high variance

(g) Stationary (h) Earlier and final variations (i) Seasonal and final variations

Figure 4.4: Examples of bank account clusters. Panel(a) shows a cluster which contains account with seasonal behavior and medium variance. Panels (b) and (c) present large ineastability in the last quarter. Panels (d), (e) and (f) outcome extremely high variance with seasonal patterns. Panels (g) and (h) are quite stable with no remarkable oscilations, only (h) shows variation in the early and final months. Panel (i) differs with (a) since in the last quarter the accounts shows more activity.

Forecast

Once the clustering process is complete for both samples of bank accounts we are able to begin with the forecasting. The first stage involves a deep and exhaustive exploration of the bank accounts of each cluster trying figure out the particularities and features that has to be expressed by each kernel function.

Since the election of a specific kernel function is done manually[8], it is sometimes a trial

[8]Cambridge Machine Learning Group on this topic is currently developing and automatic statistician, https://www.automaticstatistician.com/index/

and error procedure. Thus, our particular election might not be the optimal, however to mitigate this problem several restarts and a wide variety of kernel functions have been considered for each cluster.

As in the global approach, the hyperparameters were fitted by optimizing the marginal likelihood using a conjugate gradient optimizer as BFGS. On the other side, since our goal is to compare the results achieved in the different approaches the measures of accuracy considered in the clustering are the same as in the global view, but for exploratory reasons we incorporate the 68,75% confidence interval. Hence, we specify the forecast results for each cluster as well as for all the clusters as an entity. The results of the samples cointaing 25.000 and 100.000 bank accounts clustered in 20 and 40 different groups respectively would be analyzed separately.

Results of the 25.000 accounts sampled in 20 clusters

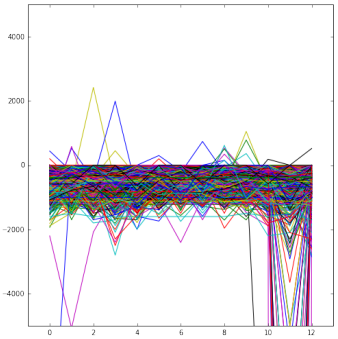
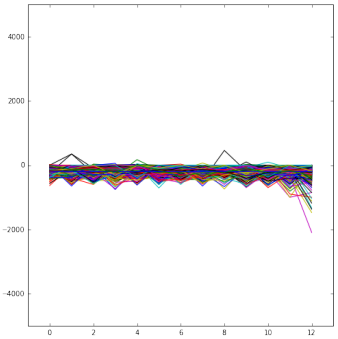
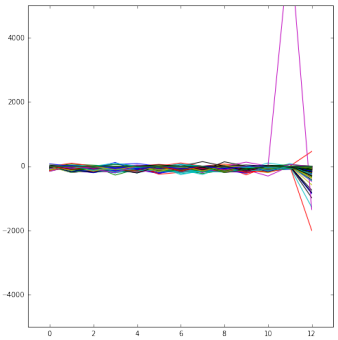
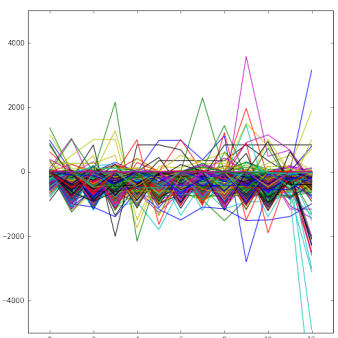
The analysis will proceed from the general to the particular, since we will compare first the results achieved with the clustering to the ones in the global approach. Then, we will provide a table showing for each cluster a plot of the accounts, the kernel function used and the different measures of accuracy.

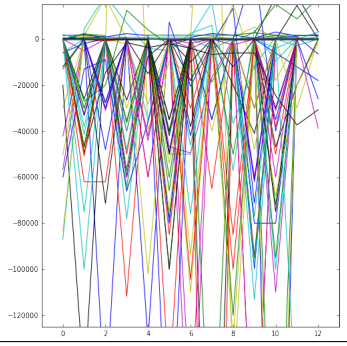
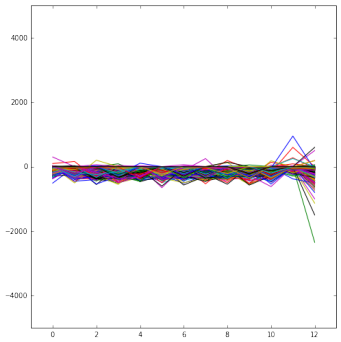
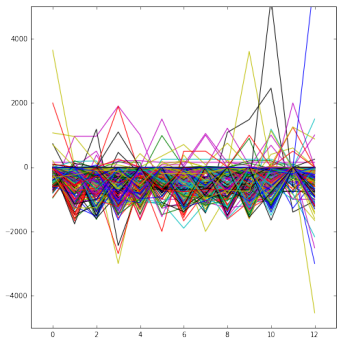
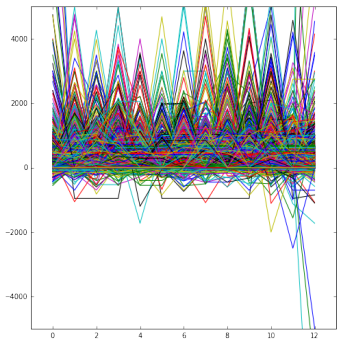
The general results of the approaches used to forecast the 13th state of 25.000 bank accounts are summarized in the table below. We can state that clustering has overperformed the global study in all of the measures of accuracy. Most significantly, the increase in the percentage of predictions that actually fit in the 95% of confidence interval as well as the reduction in all the error magnitudes. There is a substantial improvement, the fact that we reach a percentage close to 95% in the last measure of accuracy it is promising. However, it has been noted that those accounts whose confidence interval has not been accurately specified has a larger DTW distance from the centroid, meaning that there is not a representative cluster for that particular accounts and consequently the kernel function used was not the appropiate. Those miss specifications are caused by accounts with and erratic behaviour and high oscillations in the 13th state that do not get captured by GPs.

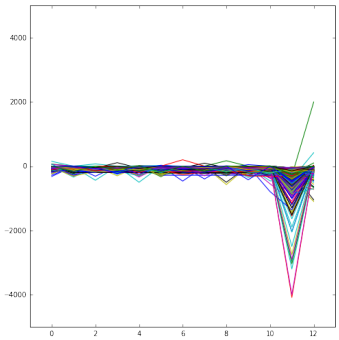
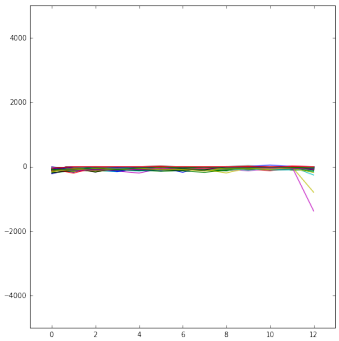
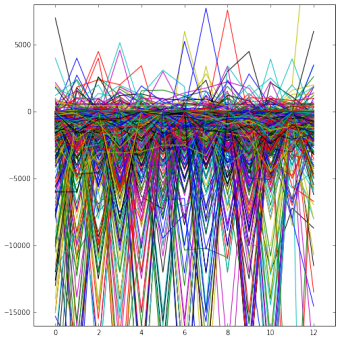
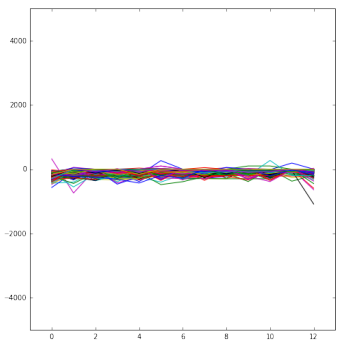
From a bussiness point of view it would be extremely profitable the study of each cluster independently since it provides a more comprehensive and useful way to understand in which type of accounts GP performs better. Therefore, we provide a table showing the performance of each cluster in the following pages.

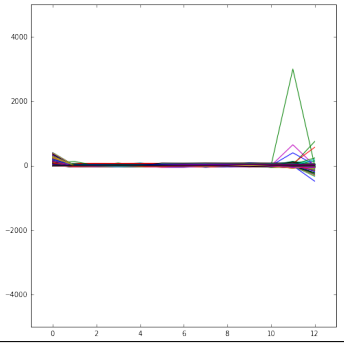
Table 4.1: Measures of accuracy achieved examining the 25.000

	MAE	**RMSE**	**MASE**	**SAFC**	**SAFC0**	**SAFCcor**	**95% conf. inter.**
N = 25.000 accounts cluster	148,7359	1786,455	0,901701	65,68%	96,712%	83.142%	93,568%
N = 25.000 accounts	150.6495	1897.5950	0.92243	65.644%	91.724 %	81.464%	88.288%

<table>
<tr><th>Clusters</th><th>Kernel</th><th>Measures of accuracy</th></tr>
<tr><td>[1] 1023 accounts
  </td><td>• $k_1(x,x') = 1050^2 \exp(-\frac{(x-x')^2}{2*70.0^2})$
• $k_2(x,x') = 20^2 \exp(-\frac{(x-x')^2}{2*10^2} - \frac{2\sin^2(\pi(x-x')/4)}{20^2})$
• $k_3(x,x') = 0.5^2(1+\frac{(x-x')^2}{2*0.5})^{-0.5}$
• $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>209,915</td></tr><tr><td>RMSE</td><td>325,979</td></tr><tr><td>MASE:</td><td>0,7799</td></tr><tr><td>SAFC:</td><td>0,931574</td></tr><tr><td>SAFC0:</td><td>0,998045</td></tr><tr><td>SAFCc:</td><td>0,931574</td></tr><tr><td>95%:</td><td>0,937439</td></tr><tr><td>68,75%:</td><td>0,680352</td></tr></table></td></tr>
<tr><td>[2] 1108 accounts
  </td><td>• $k_1(x,x') = 170^2 \exp(-\frac{(x-x')^2}{2*23.0^2})$
• $k_2(x,x') = 10^2 \exp(-\frac{(x-x')^2}{2*0.6^2})$
• $k_3(x,x') = 3.5^2(1+\frac{(x-x')^2}{2*0.7})^{-0.7}$
• $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>91,7583</td></tr><tr><td>RMSE</td><td>164,059</td></tr><tr><td>MASE:</td><td>0,73564</td></tr><tr><td>SAFC:</td><td>0,8510</td></tr><tr><td>SAFC0:</td><td>0,9981</td></tr><tr><td>SAFCc:</td><td>0,8574</td></tr><tr><td>95%:</td><td>0,9205</td></tr><tr><td>68,75%:</td><td>0,7229</td></tr></table></td></tr>
<tr><td>[3] 1647 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*15.0^2})$
• $k_2(x,x') = (7.5 + 5.0^2(x-c)(x'-c))$
• $k_3(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.8})^{-0.8}$
• $k_4(x,x') = 1.5^2 \exp(-\frac{(x-x')^2}{2*0.8^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>20,667</td></tr><tr><td>RMSE</td><td>94,138</td></tr><tr><td>MASE:</td><td>0,5581</td></tr><tr><td>SAFC:</td><td>0,63995</td></tr><tr><td>SAFC0:</td><td>0,9477</td></tr><tr><td>SAFCc:</td><td>0,9453</td></tr><tr><td>95%:</td><td>0,9489</td></tr><tr><td>68,75%:</td><td>0,9058</td></tr></table></td></tr>
<tr><td>[4] 1139 accounts
  </td><td>• $k_1(x,x') = 725^2 \exp(-\frac{(x-x')^2}{2*45.0^2})$
• $k_2(x,x') = 20^2 \exp(-\frac{(x-x')^2}{2*8^2} - \frac{2\sin^2(\pi(x-x')/4)}{12^2})$
• $k_3(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.6})^{-0.6}$
• $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>90,822</td></tr><tr><td>RMSE</td><td>388,137</td></tr><tr><td>MASE:</td><td>0,5103</td></tr><tr><td>SAFC:</td><td>0,5891</td></tr><tr><td>SAFC0:</td><td>0,9666</td></tr><tr><td>SAFCc:</td><td>0,7524</td></tr><tr><td>95%:</td><td>0,9631</td></tr><tr><td>68,75%:</td><td>0,9376</td></tr></table></td></tr>
</table>

Clusters	Kernel	Measures of accuracy
[5] 128 accounts 	• $k_1(x,x') = 1570^2 \exp(-\frac{(x-x')^2}{2*43.0^2})$ • $k_2(x,x') = 18^2 \exp(-\frac{(x-x')^2}{2*0.6^2})$ • $k_3(x,x') = 3.5^2(1+\frac{(x-x')^2}{2*0.7})^{-0.7}$ • $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$ • $kernel = k_1 + k_2 + k_3 + k_4$	**MAE:** 3413,176 **RMSE** 15365,86 **MASE:** 0,1883 **SAFC:** 0,7421 **SAFC0:** 0,9843 **SAFCc:** 0,7656 **95%:** 0,7890 **68,75%:** 0,7656
[6] 1561 accounts 	• $k_1(x,x') = 38^2 \exp(-\frac{(x-x')^2}{2*15.0^2})$ • $k_2(x,x') = (7.5 + 4.5.0^2(x)(x'))$ • $k_3(x,x') = 1.2^2(1+\frac{(x-x')^2}{2*0.8})^{-0.8}$ • $k_4(x,x') = 1.5^2 \exp(-\frac{(x-x')^2}{2*0.8^2}) + 0.4^2\delta_{pq}$ • $kernel = k_1 + k_2 + k_3 + k_4$	**MAE:** 37,225 **RMSE** 119,897 **MASE:** 0,4687 **SAFC:** 0,6111 **SAFC0:** 0,9160 **SAFCc:** 0,8821 **95%:** 0,9525 **68,75%:** 0,9160
[7] 407 accounts 	• $k_1(x,x') = 1150^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$ • $k_2(x,x') = 3^2 \exp(-\frac{(x-x')^2}{2*1.1^2}) + 40 \exp(-\frac{2\sin^2(\pi(x-x')/3)}{2^2})$ • $k_3(x,x') = 10.5^2(1+\frac{(x-x')^2}{2*1.5})^{-1.5}$ • $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 5.2^2\delta_{pq}$ • $kernel = k_1 + k_2 + k_3 + k_4$	**MAE:** 208,0299 **RMSE** 520,6157 **MASE:** 0,6107 **SAFC:** 0,5601 **SAFC0:** 0,9680 **SAFCc:** 0,6240 **95%:** 0,9557 **68,75%:** 0,8525
[8] 2666 accounts 	• $k_1(x,x') = 1840^2 \exp(-\frac{(x-x')^2}{2*431.0^2})$ • $k_2(x,x') = 68^2 \exp(-\frac{(x-x')^2}{2*10^2} - \frac{2\sin^2(\pi(x-x')/4)}{2^2})$ • $k_3(x,x') = 10^2 \exp(-\frac{(x-x')^2}{2*0.4^2})$ • $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 8.2^2\delta_{pq}$ • $kernel = k_1 + k_2 + k_3 + k_4$	**MAE:** 180,942 **RMSE** 1359,294 **MASE:** 0,8055 **SAFC:** 0,6657 **SAFC0:** 0,9662 **SAFCc:** 0,7674 **95%:** 0,9429 **68,75%:** 0,8732

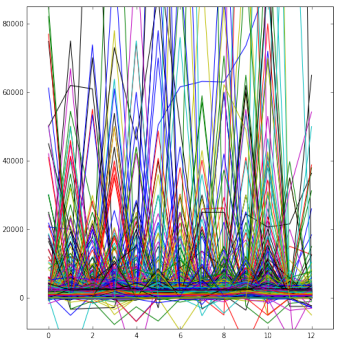
<table>
<tr><th>Clusters</th><th>Kernel</th><th>Measures of accuracy</th></tr>
<tr><td>[9] 1066 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 3.0^2 \exp(-\frac{2\sin^2(\pi(x-x')/4)}{2*2^2})$
• $k_3(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*2*0.8})^{-0.8}$
• $k_4(x,x') = 0.9^2 \exp(-\frac{(x-x')^2}{2*0.5^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>39,290</td></tr><tr><td>RMSE</td><td>108,776</td></tr><tr><td>MASE:</td><td>0,7780</td></tr><tr><td>SAFC:</td><td>0,7298</td></tr><tr><td>SAFC0:</td><td>0,9765</td></tr><tr><td>SAFCc:</td><td>0,8958</td></tr><tr><td>95%:</td><td>0,93808</td></tr><tr><td>68,75%:</td><td>0,8151</td></tr></table></td></tr>
<tr><td>[10] 297 accounts
  </td><td>• $k_1(x,x') = 30^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.5})^{-0.5}$
• $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3$</td><td><table><tr><td>MAE:</td><td>20,3396</td></tr><tr><td>RMSE</td><td>97,1138</td></tr><tr><td>MASE:</td><td>0,8351</td></tr><tr><td>SAFC:</td><td>0,6498</td></tr><tr><td>SAFC0:</td><td>0,97643</td></tr><tr><td>SAFCc:</td><td>0,94276</td></tr><tr><td>95%:</td><td>0,92929</td></tr><tr><td>68,75%:</td><td>0,88552</td></tr></table></td></tr>
<tr><td>[11] 1214 accounts
  </td><td>• $k_1(x,x') = 2198^2 \exp(-\frac{(x-x')^2}{2*760.0^2})$
• $k_2(x,x') = 550^2 \exp(-\frac{(x-x')^2}{2*56^2} - \frac{2\sin^2(\pi(x-x')/4)}{2*21^2})$
• $k_2(x,x') = 35.5^2(1+\frac{(x-x')^2}{2*2*0.8})^{-0.8}$
• $k_4(x,x') = 0.9^2 \exp(-\frac{(x-x')^2}{2*0.8^2}) + 10.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>695,318</td></tr><tr><td>RMSE</td><td>4083,819</td></tr><tr><td>MASE:</td><td>0,5403</td></tr><tr><td>SAFC:</td><td>0,7067</td></tr><tr><td>SAFC0:</td><td>0,9654</td></tr><tr><td>SAFCc:</td><td>0,7578</td></tr><tr><td>95%:</td><td>0,9093</td></tr><tr><td>68,75%:</td><td>0,7578</td></tr></table></td></tr>
<tr><td>[12] 573 accounts
  </td><td>• $k_1(x,x') = 105^2 \exp(-\frac{(x-x')^2}{2*22.5^2})$
• $k_2(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.5})^{-0.5}$
• $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.5^2}) + 12.5^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3$</td><td><table><tr><td>MAE:</td><td>39,096</td></tr><tr><td>RMSE</td><td>82,827</td></tr><tr><td>MASE:</td><td>0,6128</td></tr><tr><td>SAFC:</td><td>0,4781</td></tr><tr><td>SAFC0:</td><td>0,9895</td></tr><tr><td>SAFCc:</td><td>0,8376</td></tr><tr><td>95%:</td><td>0,9546</td></tr><tr><td>68,75%:</td><td>0,8708</td></tr></table></td></tr>
</table>

<table>
<tr><th>Clusters</th><th>Kernel</th><th>Measures of accuracy</th></tr>
<tr>
<td>[13] 2163 accounts
  </td>
<td>

- $k_1(x,x') = 65^2 \exp(-\frac{(x-x')^2}{2*25.0^2})$
- $k_2(x,x') = 3.5^2(1+\frac{(x-x')^2}{2*2*0.5})^{-0.5}$
- $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 0.2^2\delta_{pq}$
- $kernel = k_1 + k_2 + k_3$

</td>
<td>

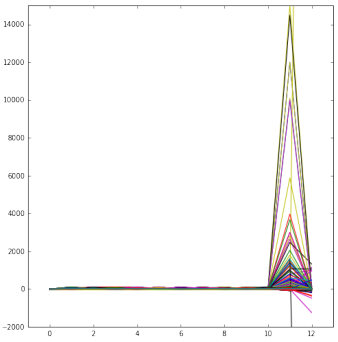
MAE:	5,7145
RMSE	31,596
MASE:	0,389677
SAFC:	0,742025
SAFC0:	0,941285
SAFCc:	0,973185
95%:	0,930652
68,75%:	0,900139

</td>
</tr>
<tr>
<td>[14] 1092 accounts
  </td>
<td>

- $k_1(x,x') = 9050^2 \exp(-\frac{(x-x')^2}{2*955.0^2})$
- $k_2(x,x') = 85^2 \exp(-\frac{2\sin^2(\pi(x-x')/4)}{2*35^2})$
- $k_3(x,x') = 135.5^2(1+\frac{(x-x')^2}{2*67*1.5})^{-1.5}$
- $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 45.2^2\delta_{pq}$
- $kernel = k_1 + k_2 + k_3 + k_4$

</td>
<td>

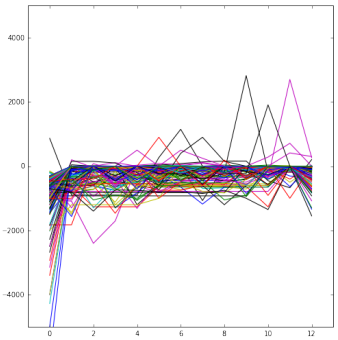
MAE:	930,989
RMSE	4652,450
MASE:	0,5371
SAFC:	0,8397
SAFC0:	0,9761
SAFCc:	0,8443
95%:	0,8690
68,75%:	0,7710

</td>
</tr>
<tr>
<td>[15] 2657 accounts
  </td>
<td>

- $k_1(x,x') = 140^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
- $k_2(x,x') = 25.0^2 \exp(-\frac{2\sin^2(\pi(x-x')/3)}{2^2}) * (10.5 + 5.0^2(x-c)(x'-c))$
- $k_3(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.5})^{-0.5}$
- $k_4(x,x') = 1.5^2 \exp(-\frac{(x-x')^2}{2*0.8^2}) + 0.2^2\delta_{pq}$
- $kernel = k_1 + k_2 + k_3 + k_4$

</td>
<td>

MAE:	58,0384
RMSE	1973,60
MASE:	3,3544
SAFC:	0,6620
SAFC0:	0,9649
SAFCc:	0,9793
95%:	0,9401
68,75%:	0,9036

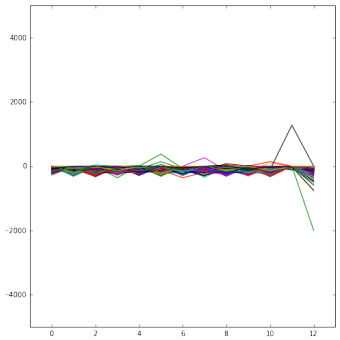
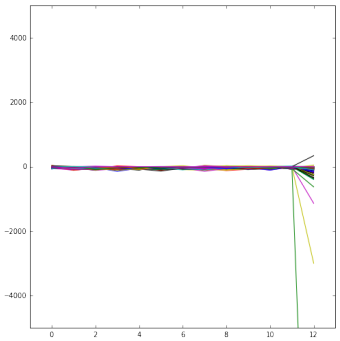
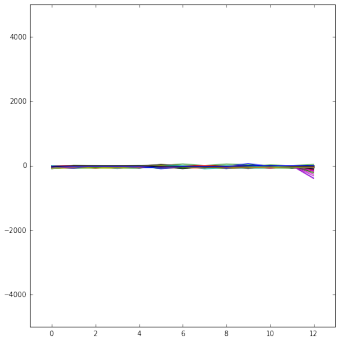
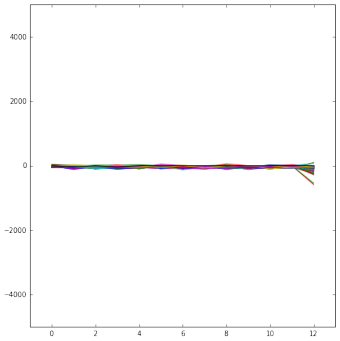
</td>
</tr>
<tr>
<td>[16] 261 accounts
  </td>
<td>

- $k_1(x,x') = 120^2 \exp(-\frac{(x-x')^2}{2*60.0^2}) * (5.0^2(x-c)(x'-c))$
- $k_2(x,x') = 3.5^2(1+\frac{(x-x')^2}{2*12*0.5})^{-0.5}$
- $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 12.0^2\delta_{pq}$
- $kernel = k_1 + k_2 + k_3$

</td>
<td>

MAE:	80,1597
RMSE	176,4381
MASE:	0,4522
SAFC:	0,6551
SAFC0:	0,9731
SAFCc:	0,7816
95%:	0,9808
68,75%:	0,9080

</td>
</tr>
</table>

<table>
<tr><th>Clusters</th><th>Kernel</th><th>Measures of accuracy</th></tr>
<tr><td>[17] 1316 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 12 * \exp(-\frac{2\sin^2(\pi(x-x')/4)}{2*2^2})$
• $k_3(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*2.0*0.5})^{-0.5}$
• $k_4(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 8.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3 + k_4$</td><td><table><tr><td>MAE:</td><td>44,1591</td></tr><tr><td>RMSE</td><td>88,0341</td></tr><tr><td>MASE:</td><td>0,6279</td></tr><tr><td>SAFC:</td><td>0,4962</td></tr><tr><td>SAFC0:</td><td>0,9992</td></tr><tr><td>SAFCc:</td><td>0,7963</td></tr><tr><td>95%:</td><td>0,9354</td></tr><tr><td>68,75%:</td><td>0,7796</td></tr></table></td></tr>
<tr><td>[18] 2000 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*0.65})^{-0.65}$
• $k_3(x,x') = 2.8^2 \exp(-\frac{(x-x')^2}{2*0.8^2}) + 0.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3$</td><td><table><tr><td>MAE:</td><td>19,7471</td></tr><tr><td>RMSE</td><td>399,311</td></tr><tr><td>MASE:</td><td>0,9873</td></tr><tr><td>SAFC:</td><td>0,6725</td></tr><tr><td>SAFC0:</td><td>0,9475</td></tr><tr><td>SAFCc:</td><td>0,95</td></tr><tr><td>95%:</td><td>0,9435</td></tr><tr><td>68,75%:</td><td>0,9035</td></tr></table></td></tr>
<tr><td>[19] 1030 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 1.5^2(1+\frac{(x-x')^2}{2*5*0.5})^{-0.5}$
• $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 1.8^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3$
1</td><td><table><tr><td>MAE:</td><td>11,8496</td></tr><tr><td>RMSE</td><td>29,2630</td></tr><tr><td>MASE:</td><td>0,5213</td></tr><tr><td>SAFC:</td><td>0,4728</td></tr><tr><td>SAFC0:</td><td>0,9932</td></tr><tr><td>SAFCc:</td><td>0,9631</td></tr><tr><td>95%:</td><td>0,9388</td></tr><tr><td>68,75%:</td><td>0,8048</td></tr></table></td></tr>
<tr><td>[20] 1652 accounts
  </td><td>• $k_1(x,x') = 40^2 \exp(-\frac{(x-x')^2}{2*20.0^2})$
• $k_2(x,x') = 12^2(1+\frac{(x-x')^2}{2*8*0.5})^{-0.65}$
• $k_3(x,x') = 0.5^2 \exp(-\frac{(x-x')^2}{2*0.3^2}) + 10.2^2\delta_{pq}$
• $kernel = k_1 + k_2 + k_3$</td><td><table><tr><td>MAE:</td><td>16,5497</td></tr><tr><td>RMSE</td><td>36,331</td></tr><tr><td>MASE:</td><td>0,5600</td></tr><tr><td>SAFC:</td><td>0,4279</td></tr><tr><td>SAFC0:</td><td>0,9927</td></tr><tr><td>SAFCc:</td><td>0,9854</td></tr><tr><td>95%:</td><td>0,9285</td></tr><tr><td>68,75%:</td><td>0,8008</td></tr></table></td></tr>
</table>

In the table above for each cluster we provide a plot of the accounts, in the second column we specify the kernel function used to perform the forecasting process. Note that each kernel is the result of adding different covariance functions that can actually be combinations of products or sums of other covariance functions in order to capture the particularities and properties of the accounts. In the third column the measures of accuracy are written.

Results of the 100.000 accounts sampled in 40 clusters

As seen in table below, the clustering process has improve all the previous results given by the global approach. MAE has decreased by 17%, RMSE by slightly more than 5% and MASE has consolidated this positive trend since it has also been reduced. Note that MAE, RMSE and MASE are cumulative. Thus, even thought the error is divided by the size of the sample, it seems natural that the error measures for the 100.000 bank accounts clustering would be greater than in the 25.000 sample as occurs with MAE and RMSE. However, MASE decreases as more data we consider meaning that naive method performs worse.

The accuracy in the sign predicitions including 0, expressed by SAFC0, gets an impresive 96,96% and SAFCco shows a performance slightly greather than 83% also improving the non-clustering approach.

Again the 95% confidence interval measure of accuracy is promising. We are not far from the desired result and the reason why we did not achieve it is because in a few clusters, the ones with high variance and steep oscillations, the performance is less satisfactory and affects negatively the global accuracy.

However, from the perspective of the Bank that is not an important drawback, since once a particular account is assigned to that type of clusters they have to take into consideration that characteristic behaviour and be more adaptive and flexible. Similarly as done before in the 20 clusters approach, a table showing a plot of the accounts, the kernel function used and the different measures of accuracy for each cluster is shown, but given the space it takes up is attached in the Github.

Table 4.22: Measures of accuracy achieved examining the 100.000

	MAE	**RMSE**	**MASE**	**SAFC**	**SAFC0**	**SAFCcor**	**95% conf. inter.**
N = 100.000 accounts cluster	134,2835	5146,1119	0,691138	64,8630%	96,965%	83.312%	93,4240%
N = 100.000 accounts	159.9937	5389.1124	0.71326	64.826 %	91.409 %	81.058%	88.233%

To sum up, once the comparative analysis between global approach and clustering is widely discussed, we can state without reservation that the latter one works better and provides a more accurate forecasting. As expected being able to adapt the kernel functions to the behaviour of the accounts gives rise to a more flexible model. Obviously, there are

some aspects that have to be improved, specifically being able to work with the whole sample of bank accounts while speeding up the computations. Thus, we could establish with greater certitude the number of optimal clusters, redefine some covariance functions and, ultimately, upgrade the forecasting model.

Finally, in the last chapter we analyze all the results from a more global perspective, outlining many of the aspects developed all along the project, specially for the bank account problem, and suggesting possible future directions of work on Gaussian processes.

Chapter 5

Conclusions

In this chapter we briefly wrap up some threads we developed throughout the project, a bussines perspective of the bank account forecasting problem and, as said before, propose exciting and new problems that can be adressed using Gaussian Processes.

Over the course of the project we saw how Gaussian process regression is a natural extension of Bayesian linear regression to a more flexible class of models. Placing Gaussian process priors over functions is a clear Bayesian viewpoint. Thus, since the adoption of Bayesian methods in the machine learning community is quite widespread this allowed me to acquire solid foundations in order to move forward in that exciting field.

From my personal perspective one of the most challenging issues was working with covariance functions and the incorporation of prior knowledge through the choice and the parameterization of the kernels. I do not like to view Gaussian processes as a black box, (what exactly goes in the box is less important, as long it gives good predicitons) thus I have always tried to ask how and why the models work or fail. This meant testing different hypotheses, trying out different components and basically experimenting by trial and error in order to gain real insight into the data.

In that sense the first part of the work, with the chapters up to and including chapter 3, where core material and theoretical concepts are given an in depth treatment have represented a substantial part of the time invested. I attempted to put in practice all that knowledge with the prediciton example developed at the end of chapter 3. Hence to face the bank account forecasting problem with the highest expectations of success.

Once the problem of the bank accounts has been adressed and the results provided we can assert that Gaussian Process has not performed as well as LSTM. Although, our goal was not to overperform LSTM since it has been proved by the bank that this latter works better. Therefore the aim was to provide a forecasting model using Gaussian Processes for Regression and decide whether or not clustering the accounts gives rise to a better performance. Both questions have been answered positively.

With the clustering approach we were able to determine an uncertainty region for the prediction of each account and it turned out that 93,42% of the values fitted in that region.

Moreover, having defined different clusters, having acces to individual measures of accuracy and knowing the behaviour patterns of the accounts belonging to each cluster we can adapt our predicitons to a wide range of needs.

In that sense, given the duty to predict the 13th state of a new account the procedure will be clear. Firstly, we would compute the DTW between the account and all of the centroids and assign the account to the centroid that minimizes the DTW. Once that account is placed in a cluster we can begin the forecasting taking into consideration the features of the cluster and adapating all of the decisions regarding that particular propiertes. Thus, if the account was placed in a cluster that actually gives poor performance we have to be cautious and prudent with the applications of that forecast. By contrast, if the account belongs to a cluster with nice perfomances rates our we can be more confident.

To sum up, our model performs quite accurately but when a company as a bank is developing a risky project like this, every variable has to be perfectly studied and no mistakes are allowed. Hence, it would be unnacceptable to send a message to a client warning him to take care of his expenses and fail in the forecast. The model has to be trained and the more data we have the more precise the prediction will be since the kernel function can fit better the data or the cluster be more precise. In fact, as mentioned at the end of chapter 4, one area for future work might be learn to deal with the 3.35 million of bank accounts .

Finally, let me state some interesting developments done in the Gaussian Process context. In the recent years one of the most important advances in the Gaussian process framework has been the creation of an Automatic Statistician. This project developed by a group of the most prestigious experts in Gaussian processes, as James Lloyd, David Duvenaud and Zoubin Ghahramani from Cambridge University, automatically produces a 10-15 page report describing patterns discovered in data. Moreover, it returns a statistical model with state-of-the-art extrapolation performance evaluated over real time series data sets from various domains. The system is based on reasoning over an open-ended language of nonparametric models using Bayesian inference. Actually, The Automatic Statistician project has won the Google Focused Research Award. This Award consists of a no-strings-attached donation to support research on this topic.

Another interesting area in which there has been an explosion of work over the last years is the use of deep neural networks for modeling high-dimensional functions as a popular alternative to Gaussian process. In that sense, many researches propose to study deep neural networks as priors on functions. As a starting point, they relate neural networks to Gaussian processes and examine a class of infinitely wide deep neural networks called deep Gaussian processes which turn to be compositions of functions drawn from GP priors. One of the main reason why deep GPs are attractive from a model-analysis point of view is because they remove some of the details of finite neural networks. Thus, this area seems to be exciting and appealing for further study.

Bibliography

[1] Cunningham, J. P., Shenoy, K. V., and Sahani, M. (2008). Fast Gaussian process methods for point process intensity estimation. In International Conference on Machine Learning, pages 192-199.

[2] Duvenaud D., James Robert Lloyd, Roger B. Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In Proceedings of the 30th International Conference on Machine Learning, 2013.

[3] M. Ebden. Gaussian Processes for Regression: A Quick Introduction. University of Oxford, 2008.

[4] Gordon Wilson, Andrew. Covariance Kernels for Fast Automatic Pattern Discovery and Extrapolation with Gaussian Processes. Trinity College & University of Cambridge, 2014, p. 12-45.

[5] Kristjanson Duvenaud, David. Automatic Model Construction with Gaussian Processes. University of Cambridge, 2014, p. 8-30.

[6] David J. C. MacKay. Introduction to Gaussian processes. NATO ASI Series F Computer and Systems Sciences,1998.

[7] David J. C. MacKay. Information theory, inference, and learning algorithms. Cambridge University press, 2003.

[8] Radford M. Neal. Bayesian learning for neural networks. PhD thesis, University of Toronto, 1995.

[9] Murphy, K. P. Machine Learning, a Probabilistic Perspective. The MIT Press, 2006. p. 496-536.

[10] Carl E. Rasmussen and Christopher K.I. Williams. Gaussian Processes for Machine Learning, volume 38. The MIT Press, Cambridge, MA, USA, 2006.

[11] Sardá - Espinosa, Alexis. Comparing Time-Series Clustering Algorithms in R Using the dtwclust Package. Cologne University of Applied Science. p.4-22

[12] Robert Schaback,Holger Wendland Kernel Techniques: From Machine Learning to Mesh less Methods, Cambridge University Press (2006)