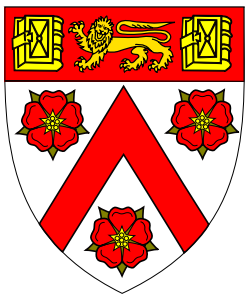
The Automatic Statistician for Classification



Qiurui He

Supervisor: Professor Zoubin Ghahramani

Department of Engineering
University of Cambridge

A Fourth-year Project Report
Submitted in Partial Fulfilment of the Requirements
For the Degree of
Master of Engineering
In
Information and Computer Engineering

Trinity College 24 May 2016

Acknowledgements

This project was completed under the supervision of Professor Zoubin Ghahramani, whom I would like to thank for his invaluable advice, guidance and encouragement. I am also grateful to Dr Christian Steinruecken for all the helpful discussions and suggestions. Their input was crucial to the success of this project.

My Directors of Studies in Trinity College have always been excellent mentors and sincere friends. I am eternally in their debt. They are Dr Joan Lasenby (2012-2014), Professor Nick Kingsbury (2014-2015) and Dr Per Ola Kristensson (2015-2016).

I have been supported by a full scholarship provided by the Jardine Foundation during my four years at Cambridge. Their generosity is very much appreciated.

Abstract

Thanks to developments in sensing, networking and digital processing, our society is producing data at a staggering rate. Within this flood of data lies a huge amount valuable information about our personal lives, business opportunities and social trends. To unearth the information, we must find ways to relate the data and communicate the hidden patterns in an accurate and meaningful manner.

The ultimate goal of the Automatic Statistician is to build an artificial intelligence for data science: to discover the model underlying an arbitrary dataset without human input. It has already been successful in producing accurate and interpretable data analysis reports automatically for time series analysis tasks (i.e. regression with one input variable).

This project focuses on binary classification, another major category of machine learning problems. We introduce AutoGPC, an Automatic Statistician for classification, which is, to the best of our knowledge, the first tool that can carry out automatic construction and natural-language description of binary classification models.

AutoGPC is based on Gaussian processes (GPs). Gaussian processes characterise models by covariance functions (kernels), which provide a flexible and elegant model representation.

In specific, we designed and implemented a Python package that is able to read in a training dataset, explore and evaluate possible models, and generate a data analysis report. The package consists of four modules: an inference engine, a kernel search procedure, a visualisation library and a report generator. The relationship between these modules is shown in Fig. 1.

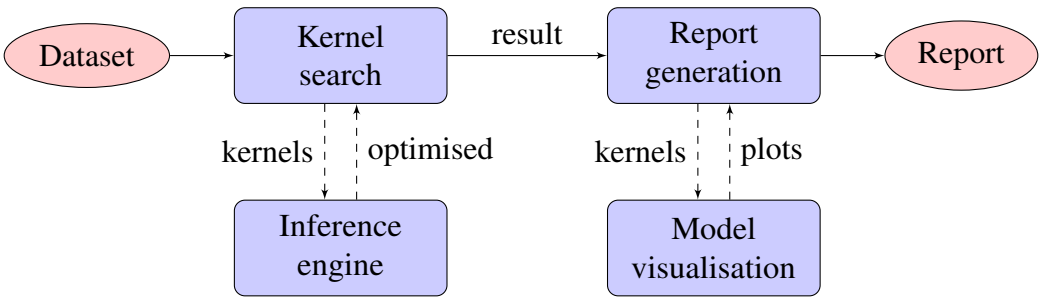


Fig. 1 Schematic diagram of the AutoGPC.

- **Inference engine** optimises hyperparameters given data and form of the kernel. The training objective is to maximise (log) marginal likelihood. GPy, an open-source Gaussian process framework in Python, is used.
- **Kernel search** identifies the best model in the light of the training data. An open-ended kernel space is defined with a finite set of one-dimensional base kernels and '$+$' and '$\times$' operations. Models of greater complexity are created progressively by augmenting the current kernel with an extra base kernel. The search of kernel space is guided by cross-validated classification error, and is implemented with a greedy beam search.
- **Model visualisation** presents GP classification models (up to three input dimensions) and datasets (of arbitrary dimensionality) with intuitive graphs. The visualisation library is also capable of plotting a lower-dimensional projection of a high-dimensional model.
- **Report generation** involves the automatic analysis and natural-language description of the full model found by the search procedure. The generated report describes the relevance and monotonicity of each input variable. It also tries to decompose the full model into additive components and analyse each component individually.

We conducted experiments on five popular real-world datasets. These tests confirmed that AutoGPC does indeed find good GP models of the data which perform no worse than models obtained by other methods. The tests also showed the report generation mechanism generalises well to different applications. We can therefore conclude that AutoGPC is a fully functional binary classifier and that it is able to discover and describe structures automatically.

Our code is publicly available on GitHub.[1]

[1] http://github.com/charles92/autogpc

Table of contents

Chapter 1

Introduction

We live in an age of data. The information that we digest every day is unprecedented in its scale, depth and diversity, which ultimately relies on data. At the same time, almost all aspects of our daily lives are recorded digitally by devices such as smartphones and wearables.

With the help of data, we are able to tackle many previously unsolved problems. Data science has helped researchers achieve breakthroughs in diagnostic medicine, particle physics, behavioural science and many other areas. It has also revolutionised marketing, finance, fraud prevention and urban planning.

However, unprocessed data has little value. It is the underlying pattern, rather than the raw data itself, that provides the predictive power which we desire. While there are software packages that automate simple model fitting (such as linear regression), finding the structure of a general model remains largely a black art practised by domain experts and data scientists. In nonparametric regression, for example, the form of the kernel has to be specified by a human expert in advance before the learning algorithm can find optimal hyperparameters of the model.

The Automatic Statistician is a novel concept invented in the Machine Learning Group at Cambridge. It aims to fully automate the process of data analysis. Given a set of observations, it should be able to automatically search for the model structure that best fits the data, interpret the underlying pattern, and eventually generate a data analysis report in natual language, all without human input.

The Automatic Statistician for time series regression has already been implemented by Lloyd et al. using Gaussian processes (GPs) [14]. It uses a greedy search in an infinite space of kernels defined by a compositional grammar. The resulting complex kernel is decomposed into interpretable terms, which are then used to generate a reader-friendly analysis report.

Attempts have been made to extend the approach to perform binary classification. In specific, Mrkšić confirmed it is feasible to apply a similar search procedure in kernel structure discovery for GP classification [18].

The Automatic Statistician for Classification

This project focuses on building an Automatic Statistician for binary classification problems with many continuous or discrete (cardinal) input variables. We employed the GP framework and a search procedure similar to [14, 18]. In addition, we attempted to synthesise a human-readable data analysis report with figures, tables and quantitative descriptions. We name the package *AutoGPC*, an abbreviation for the Automatic Gaussian Process Classifier.

It involves four main tasks, whose relationship is shown in Fig. 1.1:

- **Inference engine:** Given a GP kernel, perform estimation of hyperparameters and evaluation of classification performance. See Chapter 3 for details.
- **Kernel search:** Given data and an inference engine, search for the best model according to a predefined kernel grammar. See Chapter 3 for details.
- **Model visualisation:** Given an optimised kernel, which may involve multiple input dimensions, present the GP posterior in an intuitive plot. See Chapter 4 for details.
- **Report generation:** Given data, a kernel search procedure and a visualisation library, describe the input dataset in natural language in a meaningful and insightful manner. See Chapter 5 for details.

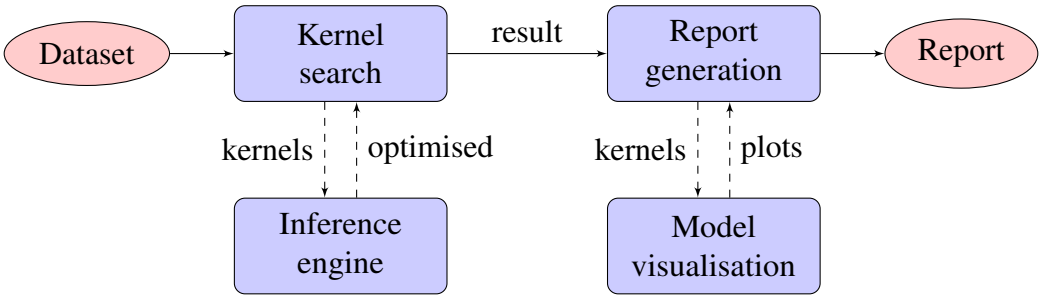


Fig. 1.1 Schematic diagram of the AutoGPC.

Although some components of the AutoGPC have been developed before as research projects or open-source software, our work is indeed the first attempt (to the best of our knowledge) to integrate these components in an effort to automatically analyse and describe the pattern underlying a binary classification problem. This is the main contribution of this project.

The project is implemented in Python and is released on GitHub at http://github.com/charles92/autogpc.

Chapter 2

Background

Supervised learning is an important topic in statistics for the analysis of (labelled) datasets. Simple parametric models are often used for this task for their simplicity and interpretability, but they may lack expressive power to deal with complex datasets. On the other hand, more complex parametric models such as neural networks are difficult to interpret [21].

Gaussian processes (GPs) can be used to form a Bayesian nonparametric framework for regression and classification. Such a framework offers great flexibility without sacrificing interpretability. It is therefore chosen as the machine learning model for previous work on the Automatic Statistician [4, 14, 18] and also for this project.

In this chapter, we give a brief introduction to Gaussian processes both in general and in the context of classification. We will present some key concepts such as kernels, hyperparameters and latent functions. We will also describe the general training criteria and procedures used to obtain posterior GPs.

Most materials in this chapter can be found in Rasmussen and Williams' *Gaussian Processes for Machine Learning* [23], which gives a more detailed and mathematically rigorous treatment of GPs.

2.1 Gaussian Processes

A *Gaussian process* (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution [21, 23].

A Gaussian process is uniquely specified by a *mean function* $m(\mathbf{x})$ and a *covariance function* or *kernel* $k(\mathbf{x}, \mathbf{x}')$, which are defined for process $f(\mathbf{x})$ as

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \end{aligned} \tag{2.1}$$

In this case, we say function $f : \mathbb{R}^D \mapsto \mathbb{R}$ is drawn from the Gaussian process $\mathcal{GP}$:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.2}$$

A Gaussian process can be thought of as the generalisation of a *multivariate Gaussian distribution* to infinite dimensions. Evaluating function $f(\mathbf{x})$ at arbitrary discrete points $\mathbf{x}_i$ produces a vector $\mathbf{f}$ where $f_i \triangleq f(\mathbf{x}_i)$. Thus random vector $\mathbf{f}$ has a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where

$$\begin{aligned} \mu_i &= m(\mathbf{x}), \\ \Sigma_{ij} &= k(\mathbf{x}_i, \mathbf{x}_j). \end{aligned} \tag{2.3}$$

As the number of samples goes to infinity, we obtain a Gaussian process. Vector $\mathbf{f}$ becomes function $f(\mathbf{x})$, mean vector $\boldsymbol{\mu}$ becomes mean function $m(\mathbf{x})$ and covariance matrix Σ becomes covariance function $k(\mathbf{x}, \mathbf{x}')$. Conversely, for any function $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$, its values $\mathbf{f}$ at a finite number of arbitrary points are random variables and are jointly Gaussian distributed.

A common practice is to let $m(\mathbf{x})$ be zero [23]. Consequently, a Gaussian process is solely specified by the kernel $k(\mathbf{x}, \mathbf{x}')$.

2.1.1 Kernels

Kernels (covariance functions) play a crucial role in Gaussian processes as they encode the prior assumptions about the general shape of the functions that we wish to learn. The parameters of a kernel are thus called *hyperparameters* in Bayesian language, since they are parameters of the prior, and *not* parameters of the model underlying the system.

Not all functions of input pair $(\mathbf{x}, \mathbf{x}')$ are qualified as kernels. In fact, a kernel must produce a valid covariance matrix Σ when it operates on a finite set of arbitrary inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. Therefore, kernels $k(\mathbf{x}, \mathbf{x}')$ are required to be

- Symmetric: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$, and
- Positive semidefinite: $\mathbf{v}^\top \Sigma \mathbf{v} \geq 0 \quad \forall \mathbf{v} \neq \mathbf{0}$ and $\forall N, \{\mathbf{x}_i\}_{i=1}^N$.

The *squared exponential* (SE) covariance function is a commonly used kernel that satisfies these requirements. It is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right). \tag{2.4}$$

The effect of the kernel is not hard to see: function values correlate strongly if they are obtained from points close to each other in the input space, while distant points have very little correlation.

Functions drawn from the SE kernel are thus smooth functions, as shown in Fig. 2.1. The hyperparameters are *signal variance* σ_0^2 and *characteristic length scale* ℓ, which control the strength of the correlation and the range of interaction respectively.

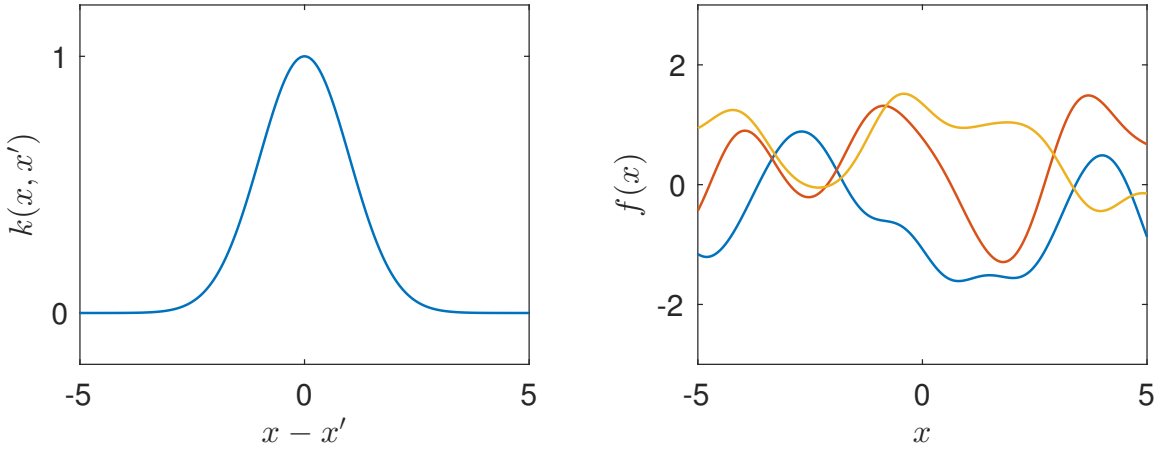


Fig. 2.1 One-dimensional squared exponential (SE) kernel and functions drawn from it.

The squared exponential kernel is a *stationary* kernel, as it only depends on the difference $\mathbf{x} - \mathbf{x}'$ and *not* the absolute locations of $\mathbf{x}$ and $\mathbf{x}'$. Moreover, since the SE kernel is only a function of distance $r = \|\mathbf{x} - \mathbf{x}'\|$, it is also known as a *radial basis function* (RBF) [23].

The *periodic* (PER) covariance function is a useful nonstationary kernel due to MacKay [16]. It is defined for scalar x as

$$k(x, x') = \sigma_0^2 \exp\left(-\frac{2}{\ell^2}\sin^2\left(\frac{\pi}{\lambda}(x - x')\right)\right). \tag{2.5}$$

This kernel models periodic functions with period λ. The other two hyperparameters σ_0^2 and ℓ have similar effects as in the SE kernel. Functions drawn from a GP with the periodic kernel are plotted in Fig. 2.2.

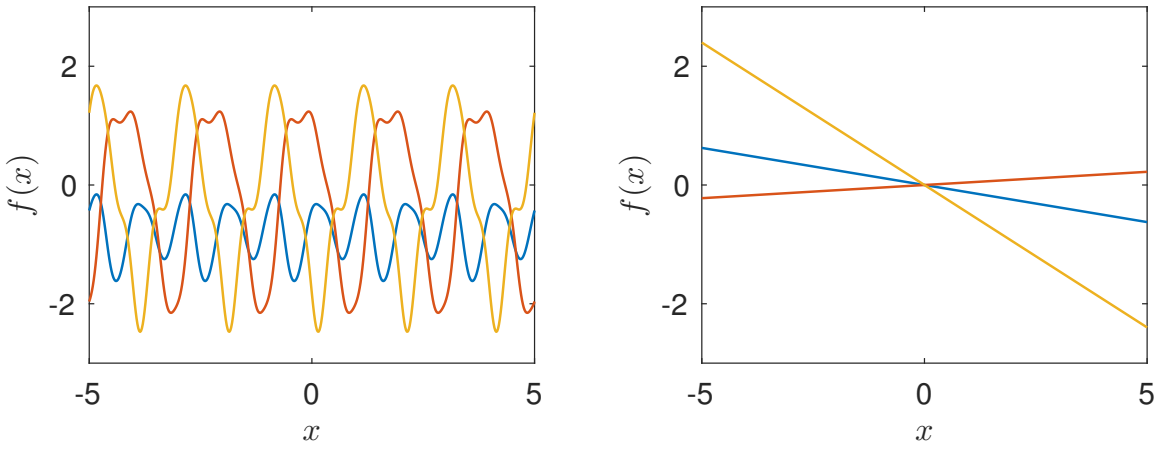


Fig. 2.2 Functions drawn from the periodic (left) and the linear (right) kernels.

There are many other classes of kernels, such as linear (LIN, also shown in Fig. 2.2), Matérn and rational quadratic kernels, which are summarised in [2, 23]. We will mainly focus on SE kernels as they are most relevant to classification tasks. Support for periodic kernels is also included in AutoGPC as an experimental feature.

2.1.2 Prediction

First consider prediction in Gaussian process regression. We will show shortly that classification is a natural extension of regression.

In a regression setting, we wish to model function $f(\mathbf{x})$ with noisy observations of the form $y_i = f(\mathbf{x}_i) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. The likelihood is therefore

$$\mathbf{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I). \tag{2.6}$$

In particular, we are interested in the *posterior* distribution of function values $\mathbf{f}_*$ given test points X_* and observations $(X, \mathbf{y})$, where X (and X_*) is many training (testing) input vectors stacked together.

Combining the Gaussian likelihood in eq. (2.6) with the Gaussian prior in eq. (2.3), we can derive an analytical expression for the joint distribution of the observed target values $\mathbf{y}$ and predicted function values $\mathbf{f}_*$, which is still Gaussian [23]:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \triangleq \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right). \tag{2.7}$$

Conditioning on observations,

$$\mathbf{f}_* \mid X_*, X, \mathbf{y} \sim \mathcal{N}\left(K_*^\top \left(K + \sigma_n^2 I\right)^{-1} \mathbf{y},\ K_{**} - K_*^\top \left(K + \sigma_n^2 I\right)^{-1} K_*\right), \tag{2.8}$$

which is the *predictive distribution* (or posterior distribution) that we want to find [21, 23]. In the noise-free case, eq. (2.8) simplifies to

$$\mathbf{f}_* \mid X_*, X, \mathbf{y} \sim \mathcal{N}\left(K_*^\top K^{-1} \mathbf{y},\ K_{**} - K_*^\top K^{-1} K_*\right). \tag{2.9}$$

Note that the covariance matrix of $\mathbf{f}$ is K_{**} *a priori*, and $K_{**} - K_*^\top \left(K + \sigma_n^2 I\right)^{-1} K_*$ *a posteriori*. The reduction in covariance reflects a lower level of uncertainty about the unknown function values in the light of extra information provided by observations $(X, \mathbf{y})$ [23]. Also note that if a single test point $\mathbf{x}_*$ is used in place of test data X_* in eqs. (2.8) and (2.9), we essentially arrive at what is called the *posterior Gaussian process* (posterior GP) with updated mean and covariance functions [21].

Computing the posterior GP has a complexity that scales with $\mathcal{O}(N^3)$ where N is number of training data points. It is dominated by inverting the matrix $K + \sigma_n^2 I$. In practice, Cholesky decomposition (complexity also $\mathcal{O}(N^3)$) is often used in place of matrix inversion, as the former is faster and numerically more stable [23]. Once the inverse (or Cholesky decomposition) is computed, predicting the distribution of the function value f_* for a single test point $\mathbf{x}_*$ takes $\mathcal{O}(N^2)$ (dominated by the calculation of variance).

2.1.3 Training

In the previous section, we showed how one could construct a posterior process from a prior GP and observed data. This requires accurate knowledge about the form of the kernel as well as its hyperparameters *a priori*, which is often infeasible in practice. It is therefore necessary to develop a method through which the correct model and its hyperparameters can be fitted to the training data. Such a process is called *training* of a Gaussian process [21].

The natural training objective for Gaussian processes is the *marginal likelihood* (also known as *evidence*), which quantifies the likelihood of the model. It is computed by marginalising out the function values $\mathbf{f}$ [23]:

$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X)\, p(\mathbf{f} \mid X)\, \mathrm{d}\mathbf{f}. \tag{2.10}$$

In our context, the prior is $\mathbf{f} \mid X \sim \mathcal{N}(\mathbf{0}, K)$ and the likelihood in eq. (2.6) is also Gaussian. This allows us to derive the expression for the *negative log marginal likelihood* (NLML):

$$\mathcal{L} = -\log p(\mathbf{y} \mid X) = \frac{1}{2}\mathbf{y}^\top \left(K + \sigma_n^2 I\right)^{-1} \mathbf{y} + \frac{1}{2}\log\left|K + \sigma_n^2 I\right| + \frac{n}{2}\log 2\pi, \tag{2.11}$$

which we minimise to find the optimal hyperparameters of a certain model. It can also be used to compare different models given data: models with a lower $\mathcal{L}$ are better than those with a higher $\mathcal{L}$.

The terms in eq. (2.11) have important implications. The first term measures the quality of fit between the model and the targets $\mathbf{y}$, whereas the second term penalises complexity of the model [18]. Minimising $\mathcal{L}$ therefore achieves a trade-off between the quality of fit and the predictive power on unseen data: an automatic Occam's razor. Otherwise, the model may overfit the data by setting noise level σ_n^2 close to zero and matching the predictive mean function to all training targets. Such a model is undesirable as it does not generalise well [21].

The minimisation of $\mathcal{L}$ can be implemented with standard gradient-based optimisation methods, such as the conjugate gradient method, as the gradients of $\mathcal{L}$ with respect to the hyperparameters have closed-form expressions [21]. However, as the objective is often noncon-

vex, one usually needs to restart the optimisation procedure with a number of different initial hyperparameters in order to avoid bad local minima.

2.2 Gaussian Process Classification

In a general binary classification task, we wish to assign a class label $y_* \in \{-1, 1\}$ to a D-dimensional test input vector $\mathbf{x}_* \in \mathbb{R}^D$. A training dataset of size N can be written as $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \{-1, 1\}$.

In particular, we are interested in *probabilistic* classification, where we model each output y as a Bernoulli random variable with parameter π, or $y \sim \mathcal{B}(\pi)$. In other words, π is the probability that y takes value 1, and $1 - \pi$ is the probability that it takes value -1. Prediction is essentially finding the distribution of π_* given data $\mathcal{D}$ and a trained model.

2.2.1 Classification as Regression

The GP regression techniques can be adapted for classification by mapping its output $f(\mathbf{x})$ to $\pi(\mathbf{x})$ with a (deterministic) *response function* $\sigma : \mathbb{R} \mapsto [0, 1]$ [23], namely

$$\pi(\mathbf{x}) \triangleq p\left(y = 1 \mid \mathbf{x}\right) = \sigma\left(f(\mathbf{x})\right). \tag{2.12}$$

Note that this formulation leads to a very compact representation of the likelihood of a single sample:

$$p\left(y_i \mid \mathbf{x}_i, f(\cdot)\right) = p\left(y_i \mid f_i\right) = \sigma(y_i f_i), \tag{2.13}$$

where for the last equality we used the fact that $p(y_i = -1 \mid \mathbf{x}_i) = 1 - \sigma(f_i) = \sigma(-f_i)$.

Common choices for the response function $\sigma(\cdot)$ are the logistic function (known as *logistic regression*) and the probit function (known as *probit regression*), among others [11, 23]. In AutoGPC, we use the probit function $\pi = \sigma(f) = \Phi(f)$ which is the cumulative distribution function of a standard normal distribution, defined as $\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(u \mid 0, 1)\, \mathrm{d}u$.

For multi-class problems, the softmax function can be used to yield a valid probabilistic distribution over class labels [23].

To summarise, the relationship between the variables in prediction can be illustrated as

$$\mathbf{x}_* \xrightarrow{f(\cdot) \sim \mathcal{GP}} f_* \xrightarrow{\sigma(\cdot)} \pi_* \xrightarrow{y \sim \mathcal{B}(\cdot)} y_*.$$

2.2.2 Challenge

To produce a probabilistic class label assignment, we need to compute the predictive distribution by marginalising out latent variable f_*

$$p\left(y_* \mid X, \mathbf{y}, \mathbf{x}_*\right) = \int \sigma(y_* f_*) p\left(f_* \mid X, \mathbf{y}, \mathbf{x}_*\right) \mathrm{d}f_*. \tag{2.14}$$

The second factor in the integrand of eq. (2.14) is the posterior distribution of latent variable f_*:

$$\begin{aligned} p\left(f_* \mid X, \mathbf{y}, \mathbf{x}_*\right) &= \int p\left(f_* \mid X, \mathbf{x}_*, \mathbf{f}\right) p\left(\mathbf{f} \mid X, \mathbf{y}\right) \mathrm{d}\mathbf{f} \\ &= \int p\left(f_* \mid X, \mathbf{x}_*, \mathbf{f}\right) \frac{p\left(\mathbf{f} \mid X\right) p\left(\mathbf{y} \mid \mathbf{f}\right)}{p\left(\mathbf{y} \mid X\right)} \mathrm{d}\mathbf{f} \\ &= \int p\left(f_* \mid X, \mathbf{x}_*, \mathbf{f}\right) \frac{p\left(\mathbf{f} \mid X\right)}{p\left(\mathbf{y} \mid X\right)} \prod_{i=1}^{N} \sigma(y_i f_i) \mathrm{d}\mathbf{f}, \end{aligned} \tag{2.15}$$

where the marginal likelihood can be written as

$$\begin{aligned} p\left(\mathbf{y} \mid X\right) &= \int p\left(\mathbf{f} \mid X\right) p\left(\mathbf{y} \mid \mathbf{f}\right) \mathrm{d}\mathbf{f} \\ &= \int p\left(\mathbf{f} \mid X\right) \prod_{i=1}^{N} \sigma(y_i f_i) \mathrm{d}\mathbf{f}. \end{aligned} \tag{2.16}$$

Assuming a GP prior on latent function $f(\mathbf{x})$, probabilities $p\left(f_* \mid X, \mathbf{x}_*, \mathbf{f}\right)$ and $p\left(\mathbf{f} \mid X\right)$ become Gaussian. However, both the posterior in eq. (2.15) and the marginal likelihood in eq. (2.16) involve an integration of the product of a Gaussian PDF and a probit function. Thus they do not have closed-form expressions. Similarly, the gradients of eq. (2.16) required during training and the predictive distribution in eq. (2.14) are analytically intractable as well [11, 23].

As a consequence, suitable approximation methods are necessary for Gaussian process classification. Historically, a popular solution is to approximate the non-Gaussian distribution by a Gaussian one [23]. Examples include the Laplace approximation [29] and expectation propagation [17]. Another approach is to approximate the integral by Monte Carlo sampling [20]. These methods typically scale as $\mathcal{O}(N^3)$. Kuss and Rasmussen wrote a good review on this topic [11].

When N is large, sparse GP classification may have an advantage as it only relies on a subset of training points (the *inducing points*) rather than the entire dataset. The informative vector machine [12], the generalised FITC method [19] and the sparse KL method [10] fall in this category. In our work, we include the sparse KL method as an alternative option.

Chapter 3

Automatic Structure Discovery

Many traditional machine learning techniques can only work with one (or a few) particular type(s) of models. The focus of such techniques is usually on the learning of parameters or hyperparameters.

One of the key advantages of the Automatic Statistician is its ability to search and optimise over a potentially infinite model space. The user is not expected to have a prior knowledge about the functional form of the pattern behind the dataset. Instead, the form of the model is *learned*.

In this chapter, we show how this can be achieved with Gaussian processes. In a GP framework, models are represented by kernels (Chapter 2). We will describe the construction of complex, open-ended kernels from a small set of basic kernels, and a practical search algorithm for the model (kernel) space.

Although this is an important component of AutoGPC, it is not completely new. We mostly follow Mrkšić's approach [18], with small modifications where necessary.

3.1 Related Work

The problem of automatic structure discovery in machine learning has caught great research interest. Historically, Todorovski and Dzeroski [28] and Schmidt and Lipson [25] tried to learn parametric equations that underlie a dataset. More recently, Grosse et al. [8] demonstrated that a greedy search of the compositional model space yields satisfactory results in an unsupervised setting.

Duvenaud et al. [4] focused on nonparametric regression problems with Gaussian processes. They defined a finite set of *base kernels* which can be added and multiplied repeatedly to form arbitrarily complex structures. They also devised a *kernel grammar* to guide the dynamic construction of new kernels during the search procedure. This approach is subsequently used by Lloyd et al. [14] to build the Automatic Statistician for regression with one-dimensional inputs.

In addition, some foundation work has been done for classification problems. Mrkšić [18] verified that it is possible to apply the techniques described in [4] to GP-based binary classification. The mathematical formulation of this setup is detailed in Section 2.2. The kernel search component of AutoGPC is largely based on the ideas presented in [18].

There exist a number of open-source packages that implement Gaussian processes in different programming languages.[1] The GPML Toolbox is written for Matlab by Rasmussen and Nickisch [22]. It provides a variety of likelihood, mean and covariance functions, as well as some approximate inference algorithms. Previous projects on the Automatic Statistician such as [4, 14, 18] use this toolbox as their inference engine.

For this project, we choose another implementation called GPy [7]. It is equally powerful as [22] but written in Python. As we will potentially deploy AutoGPC on a web server, Python provides more flexibility than Matlab. Another advantage is that GPy implements some sparse GP methods, such as [10], which may be used for efficient inference with large datasets.

3.2 Kernel Evaluation

The objective of structure discovery is to find a model that best explains the given data. It is therefore necessary to define some quantitative measure of performance or optimality. In AutoGPC, we use two performance measures: the marginal likelihood and the cross-validated classification error.

3.2.1 Marginal Likelihood

In the context of Gaussian processes, one straightforward measure is the marginal likelihood. As mentioned in Section 2.1, the marginal likelihood automatically incorporates a complexity penalty term, and thus prevents overfitting without the need for manual regularisation. This measure is used in previous GP regression work, such as [4, 14].

We use the negative log marginal likelihood (NLML) in AutoGPC as an optimisation objective when training the hyperparameters of a kernel. The NLML does not have a close-form expression for GP classification (see Section 2.2), so approximation has to be used.

Fortunately, we do not need to deal with the intricacies of the implementation, as it is already included in GPy [7] as part of the standard library. To compute the NLML, one simply needs to negate `Model.log_likelihood()`.

[1] http://www.gaussianprocess.org/#code

3.2.2 Cross-validated Classification Error

In classification problems, an important performance measure is the error rate. There are a number of ways in which the error rate can be computed, each having their own advantages and disadvantages.

One could simply compute the *training error*, which is the percentage of misclassified points in the training set. It is simple to implement, but models optimised for training error may not generalise well to unseen data as exactly the same dataset is used for both training and testing.

Another option is to reserve a fixed test set that is not used for training. A trained classifier is then evaluated using the test set, yielding the *test error*. Classifiers optimised for test error usually have improved generalisation. But this approach inevitably wastes some data since only a subset of the available observations is ever used for training. It can be problematic when the dataset is small.

When the goal is predictive accuracy, a popular alternative is to use the *cross-validated classification error* as an objective. During cross-validation, the training set is split into two complementary subsets, one for training and the other for computing error. The process is then repeated for different splits of the original dataset to reduce variability. The error rates obtained from different splits are then averaged to obtain the cross-validated error.

In our work, we use k-fold cross-validation. That is, the dataset is randomly partitioned into k subsets of (approximately) equal size. In each round of cross-validation, one subset is retained for validation (i.e. computing error rate), and the remaining $k-1$ subsets are used for training. A total of k rounds (the *folds*) of cross-validation are executed, with each subset used exactly once as validation data. Fig. 3.1 provides an illustration of the procedure.

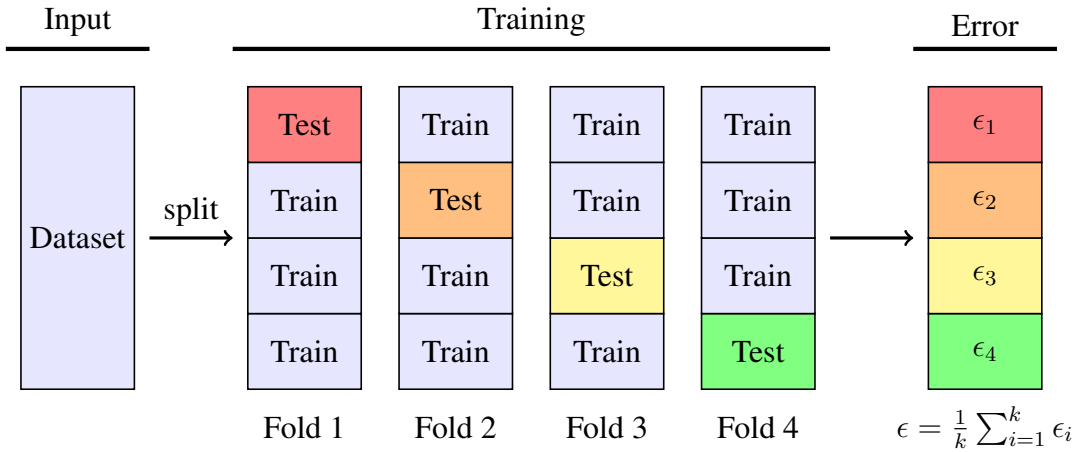


Fig. 3.1 Sketch of a k-fold cross-validation procedure ($k = 4$).

The cross-validated error is used as the performance measure in the search procedure to select the best kernels from a pool of candidates whose hyperparameters are already optimised.

This approach has been shown to have a superior performance to simply comparing the marginal likelihood [18].

3.3 Kernel Space

Finding the optimal kernel in the light of the given dataset is central to AutoGPC. To begin with, one has to define a (potentially infinite) *kernel space* that is simple to construct. At the same time, it has to be sufficiently flexible to model different patterns that may be present in the dataset. In this section, we describe how we tackle this problem by defining a set of base kernels and a simple yet powerful kernel grammar.

3.3.1 Base Kernels

We follow the practice of Mrkšić [18] and include all one-dimensional squared exponential (SE) kernels in the set of base kernels. For example, the base kernels for a two-dimensional input space are SE_1 and SE_2, as depicted in Fig. 3.2.

In addition, we also add support for one-dimensional periodic (PER) kernels which can capture periodic trends in the dataset. This may be useful when one or more of the input variables are time-related. However, the fact that there are only two class labels can potentially make it difficult to detect periodicity.

An interesting extension would be to also include one-dimensional linear (LIN) kernels. Linear kernels extract linear trend in the data, so they are potentially useful in identifying the presence of linearity in a single input dimension.

Since all base kernels are one-dimensional, we introduce a convenient notation where a kernel symbol is followed by a input dimension in subscript. For example, SE_1 means a squared exponential kernel operating on the first input dimension, and LIN_2 means a linear kernel in the second input dimension.

3.3.2 Kernel Grammar

Kernels can be combined using addition ('$+$') and multiplication ('$\times$'). The resulting kernels are called a *sum kernel* and a *product kernel* respectively. The '$+$' and '$\times$' operations can be applied repeatedly to construct complex kernels. It also enables us to capture structures that involve more than one input dimension; simply adding or multiplying the relevant one-dimensional kernels together would produce a multi-dimensional kernel.

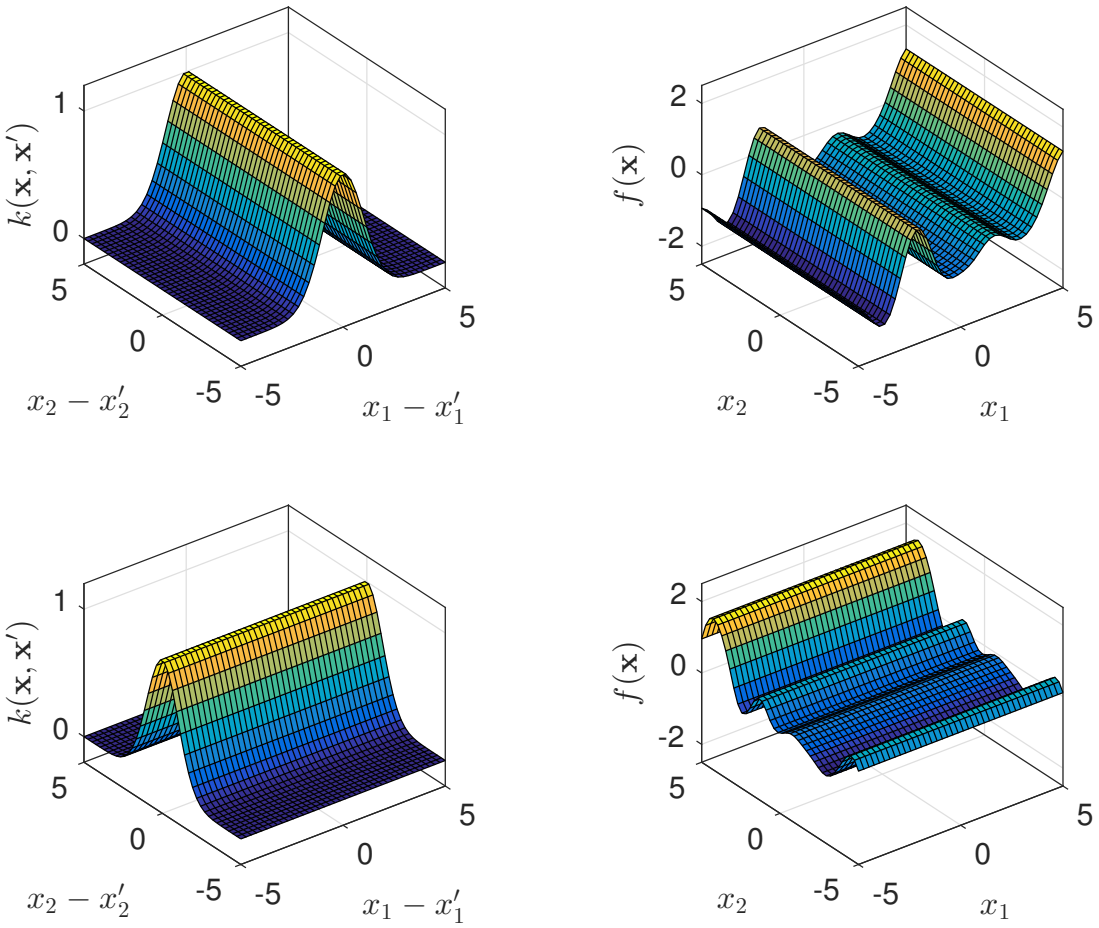


Fig. 3.2 One-dimensional kernels SE_1, SE_2 (left) and random functions drawn from these kernels (right).

Adding two kernels together (say $k_1 + k_2$) is, loosely speaking, a logical OR operation on k_1 and k_2; the resulting kernel will have a high value if either k_1 or k_2 is high. The sum of one-dimensional kernels puts a prior over functions which are a sum of one-dimensional functions, one for each input dimension [2]. For example, given a sum kernel

$$k(\mathbf{x}, \mathbf{x'}) = k_1(x_1, x_1') + k_2(x_2, x_2') \tag{3.1}$$

and a function $f(\mathbf{x})$ drawn from $k(\mathbf{x}, \mathbf{x'})$, the two input dimensions can be decoupled:

$$f(\mathbf{x}) = f_1(x_1) + f_2(x_2), \tag{3.2}$$

where $f_1(x_1)$ and $f_2(x_2)$ can be thought of as drawn from $k_1(x_1, x_1')$ and $k_2(x_2, x_2')$ respectively [3], as shown in Fig. 3.3.

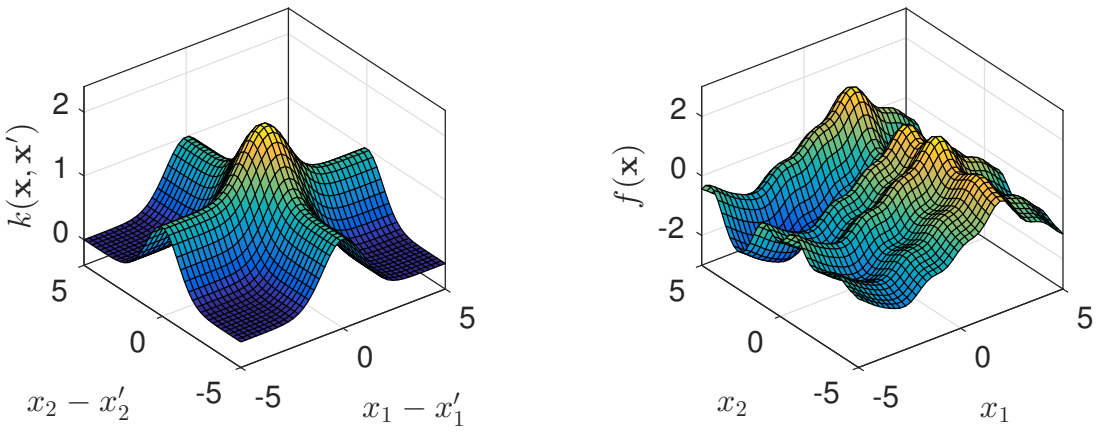


Fig. 3.3 Sum kernel $\mathrm{SE}_1 + \mathrm{SE}_2$ (left) and a random function drawn from the kernel (right).

Similarly, multiplying two kernels (say $k_1 \times k_2$) is roughly comparable to a logical AND operation of k_1 and k_2; the resulting kernel will only have a high value if both k_1 and k_2 are high [2]. The product of one-dimensional kernels represents a more complex interaction between dimensions. Suppose a product kernel is produced from the same one-dimensional kernels, i.e.

$$k(\mathbf{x}, \mathbf{x'}) = k_1(x_1, x_1')k_2(x_2, x_2'). \tag{3.3}$$

A function drawn from $k(\mathbf{x}, \mathbf{x'})$ takes the form

$$f(\mathbf{x}) = f(x_1, x_2), \tag{3.4}$$

which in general cannot be decoupled into one-dimensional components [3]. The product kernel is sketched in Fig. 3.4.

The kernel space specified by this grammar is named a *compositional* kernel space [4, 8].

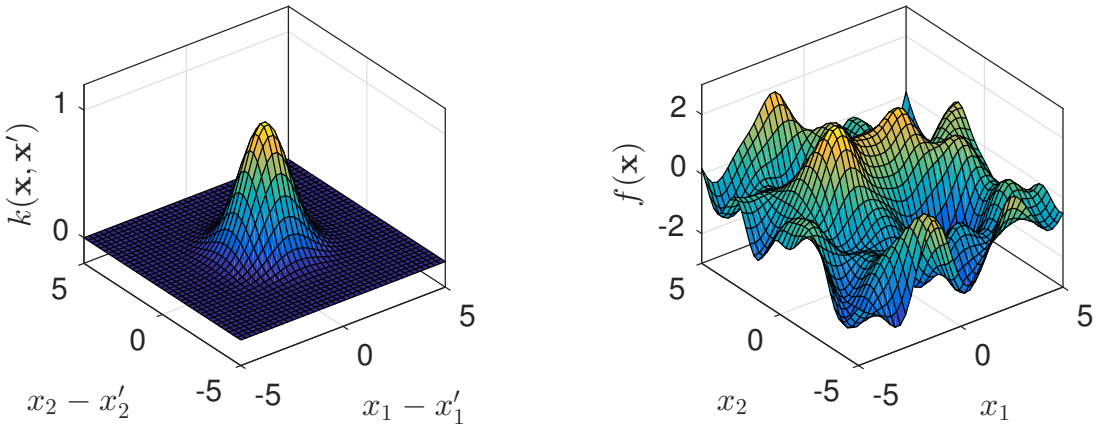


Fig. 3.4 Product kernel $\mathrm{SE}_1 \times \mathrm{SE}_2$ (left) and a random function drawn from the kernel (right).

3.4 Search Procedure

With the kernel evaluation scheme (Section 3.2) and the kernel space (Section 3.3) defined, we can adapt standard graph search algorithms to the kernel search, which we outline below.

3.4.1 Kernel Space as a Graph

Kernels in the compositional kernel space can be constructed step by step through the summation and multiplication of base kernels. This allows us to build a *graph* as follows. Let each compositional kernel k be a *node*, in addition to a *source node* representing the NULL model. A directional *edge* from node k_i to node k_j is added to the graph for each pair of nodes k_i, k_j such that kernel k_j can be constructed from k_i by a '$+$' or '$\times$' operation with a base kernel.

It is easy to show that the resulting graph is a *directed acyclic graph* (DAG). (However it is not a *tree*, since the same compositional kernel can very often be constructed via different routes.)

To *expand* a node is to find all nodes that can be reached from the current node in exactly one step. In our context, this means creating new kernels by summing and multiplying the current kernel with every base kernel.

3.4.2 Beam Search

Beam search is used to find the best-performing kernel in the above graph. It starts from the NULL node, and keeps a list of active nodes. In each iteration, it evaluates all active nodes (kernels) and select the top β nodes in terms of cross-validated error. β is the *beam width*. These selected nodes are then expanded, and the resulting nodes become the new active nodes (the old ones are discarded). The search terminates either when no further decrease in cross-validated

error can be achieved, or when the maximum search depth is reached. The complete algorithm is listed in Algorithm 3.1, and a graphical illustration is presented in Fig. 3.5.

Algorithm 3.1 Beam search

Require: $n \geq 1$ ▷ Maximum search depth
$H \leftarrow$ LIST() ▷ Search history
$i \leftarrow 1$, $A \leftarrow$ LIST(k_{NULL}) ▷ Add NULL kernel to the list of active models
while $i \leq n$ **do**
  TRAINALL(A) ▷ Train all active models
  SORT(A) by ERROR(A) ▷ Sort in descending order of error rate
  if ERROR($A[1]$) $\geq$ ERROR($H[i-1]$) **then**
    break ▷ Terminate if performance not improved
  else
    $H[i] \leftarrow A[1]$ ▷ Otherwise, store the current best kernel
  end if
  $j \leftarrow 1$, $A \leftarrow$ LIST()
  while $j \leq \beta$ **do**
    A.ADD(EXPAND($A[j]$)) ▷ Replace the active list with new models
    $j \leftarrow j + 1$
  end while
  REMOVEDUPLICATES(A) ▷ Remove duplicate kernel expressions
  $i \leftarrow i + 1$
end while
return H

Beam search is greedy. That is, it only makes locally optimal decisions and there is no guarantee for global optimality. This is a compromise between performance and accuracy, as traversing the entire graph would be prohibitively expensive with, say, breadth-first search (BFS). Neither can heuristic search algorithms such as A* be used as we are unable to define a consistent heuristic.

In practice, the search procedure is able to return high-quality models even with a beam width of $\beta = 2$. This implies that beam search is practically acceptable for the purpose of AutoGPC.

3.4.3 Constraints on Hyperparameters

We observed in our initial experiments that the search procedure occasionally returned overfitted models, as shown in Fig. 3.6. This can happen when the optimisation algorithm picks a very short length-scale ℓ that allows the model to treat nearby training points as independent observations. Consequently, the GP posterior can fit almost perfectly to the training points. However, such models are of little use, as they do not generalise. Discrete-valued input variables are more vulnerable to such problems.

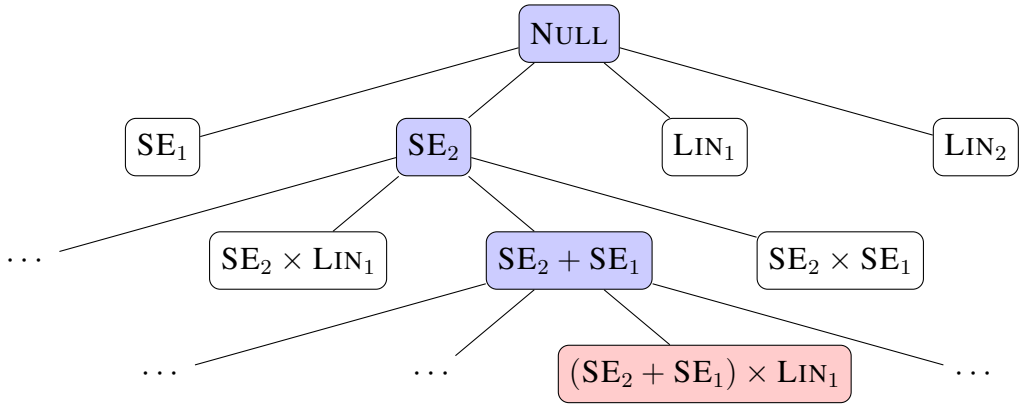


Fig. 3.5 Example of a search through the compositional kernel space. For simplicity, beam width is set to 1. The NULL node represents 'no structure' and is the start point of the search. Blue nodes represent the search path, and the red node represents the compositional kernel returned by the search procedure.

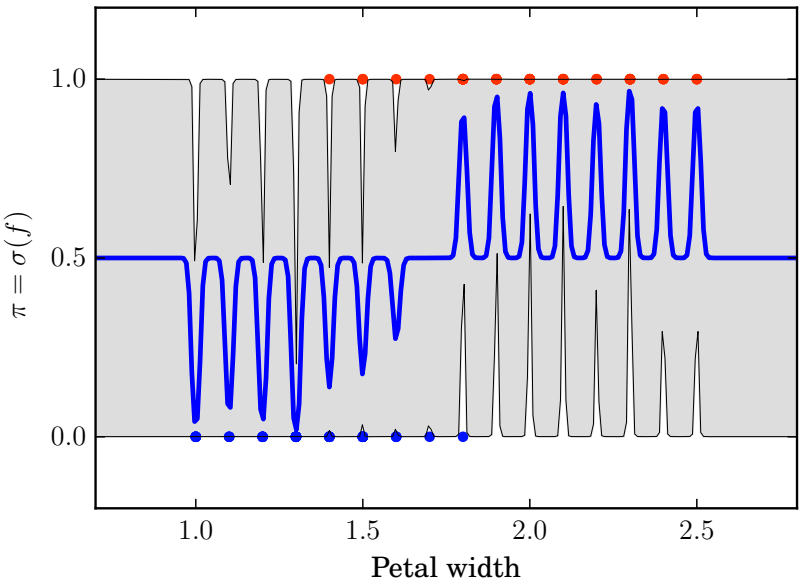


Fig. 3.6 An overfitted one-dimensional GP model when ℓ is too small.

A similar issue is underfitting. This happens when ℓ is greater than the range of the input variable by orders of magnitude. In such cases, the GP posterior behaves almost exactly like a constant kernel (Fig. 3.7), which is also undesirable.

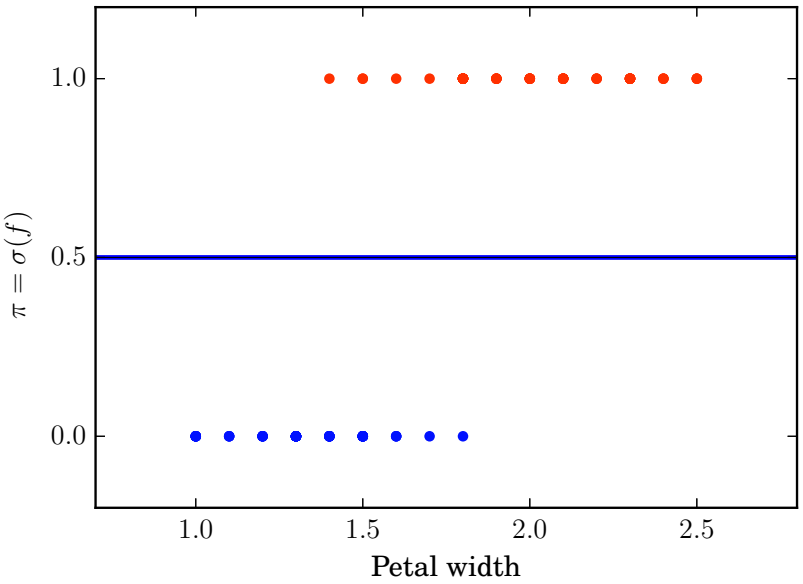


Fig. 3.7 An underfitted one-dimensional GP model when ℓ is too large.

Mrkšić also noticed the overfitting problem, and proposed to abandon models whose ℓ is smaller than the minimum distance between unique values of this variable [18].

We take a different, and probably more elegant, approach to the problem by bounding the hyperparameters. For example, length-scale ℓ of squared-exponential (SE) and periodic (PER) kernels are constrained between the minimum separation between unique values of the variable, and twice the maximum range of the variable (both computed for the training set). Mathematically,

$$\begin{cases} \ell \geq \min\{x_i - x_j \mid 1 \leq i, j \leq N,\ x_i > x_j\} \\ \ell \leq 2\max\{x_i - x_j \mid 1 \leq i, j \leq N\} \end{cases}. \tag{3.5}$$

Similar constraints are applied to the period hyperparameter λ of periodic kernels.

Constrained optimisation is supported by GPy [7]. It implements the bounds through a logistic transform, so the underlying optimisation is still unconstrained. Thus, we have successfully avoided overfitted or underfitted models in the search process.

Chapter 4

Model Visualisation

Good graphs are a concise and effective way of conveying information and often a cornerstone of an intuitive data analysis report. Hence it is desirable to be able to produce appropriate visualisations in order to facilitate the presentation of the models found during the search process.

Compared with previous work on the Automatic Statistician for time series regression [14], the nature of classification tasks poses major challenges to the visualisation of the dataset as well as the Gaussian process model. In specific,

- The outputs y are binary (for binary classification) or categorical (for multi-class problems), and do not necessarily have a natural ordering.
- The inputs $\mathbf{x}$ are often multi-dimensional and the model can potentially involve nontrivial interactions between various input dimensions.

In this chapter, we outline a series of visualisations designed specifically for GP classification. Our package is able to plot datasets with an arbitrary number of input dimensions and a multi-class output. It is also able to overlay on the dataset any binary GP classification model that involves ≤ 3 input dimensions. The graphs produced by our package have a consistent and appealing visual style.

4.1 Related Work

The problem of visualising high-dimensional data has been well studied. Most of these methods focus on *dimensionality reduction*. That is, to 'squash' the high-dimensional data to 2 or 3 dimensions.

One common solution is feature extraction, which is often achieved with *principal component analysis* (PCA). PCA is an orthogonal linear transformation. The transformed axes are linearly

uncorrelated, and are ranked according to their variance. The top-ranking features in the transformed space then selected and plotted.

Manifold learning has attracted much research attention recently. It can be regarded as a generalisation of linear techniques like PCA to the nonlinear domain. Isometric mapping (isomap) [27], locally linear embedding (LLE) [24] and t-distributed stochastic neighbour embedding (t-SNE) [15] are a few examples of manifold learning.

Although feature extraction and manifold learning techniques are successful in reducing the dimensionality of a dataset, the transformation from the high-dimensional input space to the low-dimensional feature space is difficult to interpret. In other words, it is unclear what patterns in the original input variables result in the clustering in the visualisation. Consequently, the natural structure of the data points in the original input space is inevitably lost.

In the Automatic Statistician, we try to uncover the patterns that are explained by the *original* input variables. Therefore dimensionality reduction is not a viable option. Instead, we seek elegant techniques that visualise data without manipulating the original input space. To this end, slicing (vary 2 or 3 variables while holding the rest constant) and projection are often used by data analysts. In addition, Heer et al. [9] wrote a summary of simple visualisations that can be applied to high-dimensional datasets, such as the *stem-and-leaf plot*, the *scatter plot matrix* (SPLOM), and *parallel coordinates*.

4.2 One-dimensional Data and Models

With only one input dimension, the graph is simple to draw. We present the data and the trained GP model in a Cartesian coordinate system where the horizontal axis represents the input variable x and the vertical axis represents the *probabilistic* classifier output $\pi \triangleq p(y = 1 \mid x) = \sigma(f)$ (Section 2.2).

Training data points have deterministic class labels, therefore $\pi = 1$ for positive data points (coloured red) and $\pi = 0$ for negative ones (coloured blue). They are plotted as (x, π) points on the graph.

The GP model produces a normally distributed latent function value f_* at each test point x_*. We first compute the mean and variance of f_* across the range of the x-axis. The mean and variance are then used to produce the upper and lower envelope of the 95% confidence interval (i.e. two standard deviations from the mean, or $\mathbb{E}[f_*] \pm 2(\mathbb{V}[f_*])^{1/2}$). Finally, the mean of f_* and the upper and lower envelopes are processed by the response function $\sigma(\cdot)$ to 'squash' them to range $[0, 1]$, yielding a soft class label assignment π (plotted with a thick blue line) and the corresponding predictive error bar (drawn as a solid grey region). Fig. 4.1 shows an example.

This visualisation is similar to what Mrkšić used in [18]. However he did not process the predictive latent function with the response function $\sigma(\cdot)$, resulting in a less intuitive illustration of the posterior GP model.

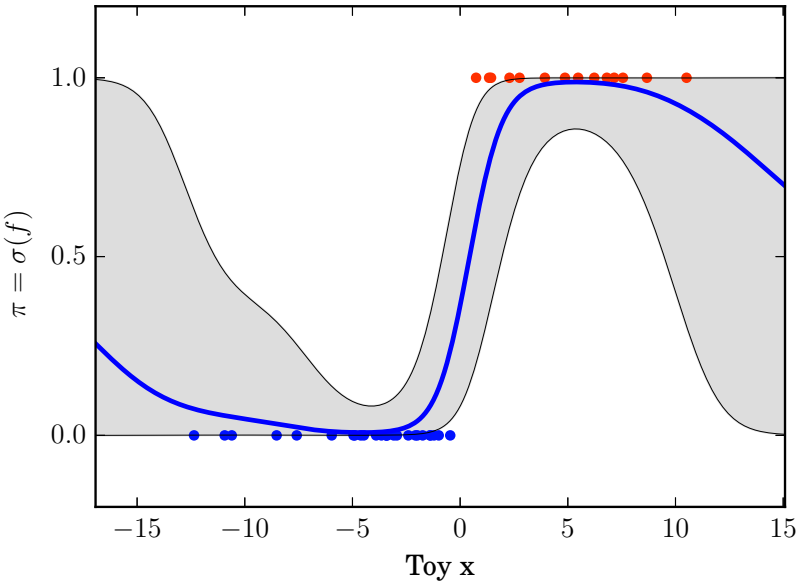


Fig. 4.1 Visualisation of 1-D data and GP model.

4.3 Two- or Three-dimensional Data and Models

Two-dimensional input space calls for 3-D points of the form (x_1, x_2, π) where $[x_1, x_2]^\top = \mathbf{x}$ is an input vector and π is the probabilistic class label assignment. Since π is a function of $\mathbf{x}$, a natural way of presenting the relationship is to use a 2-D contour plot with x_1, x_2 as independent variables.

Training data points are visualised using a 2-D scatter plot. Colour code is the same as in the one-dimensional plots: positive points are represented by red and negative ones by blue.

The GP model is processed in a similar way as in the one-dimensional case, the difference being the confidence interval (or error bar) is no longer shown by default (Fig. 4.2). Including the confidence interval in the same coordinate system would require a second set of contour plots (e.g. as background filled with colours). This tends to make the graph too complicated. As an alternative, we allow the user to create an auxiliary contour plot side-by-side with the original one, but showing confidence interval instead of the soft label assignment π.

In order to facilitate the analysis of a subset of input variables in a GP model, we make it possible to only plot the *active dimensions* of a model. This is especially useful in AutoGPC, as many kernels (including the base kernels) only operate on a subset of input dimensions. Fig. 4.3 is an illustrative example demonstrating this functionality.

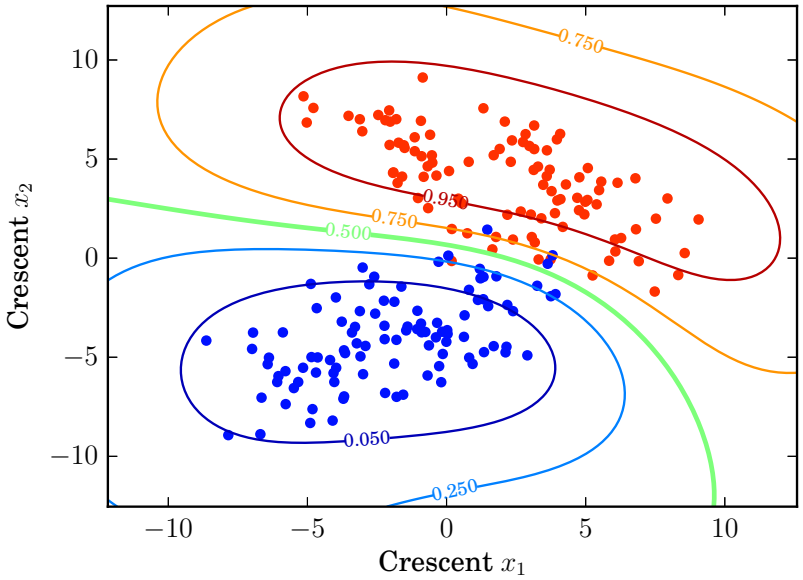


Fig. 4.2 Visualisation of 2-D data and GP model.

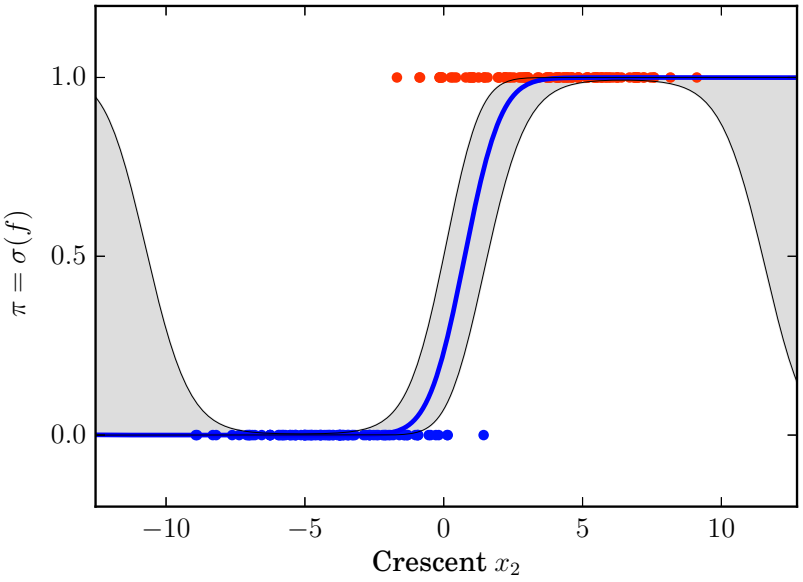


Fig. 4.3 Visualisation of 2-D data and GP model sliced into 1-D. The original full 2-D model is shown in Fig. 4.2

Three-dimensional input space is treated in a similar fashion. We simply replace 2-D scatter plot with a 3-D one, and use *contour surfaces* instead of contour lines. The resulting visualisation is presented in Fig. 4.4.

Although the confidence interval at each test point $\mathbf{x}_*$ can still be calculated in the old fashion, we decide not to support it 3-D plots. This is because in 3-D, the contour surfaces of confidence interval are often very complicated, and are very likely to distract and confuse the readers.

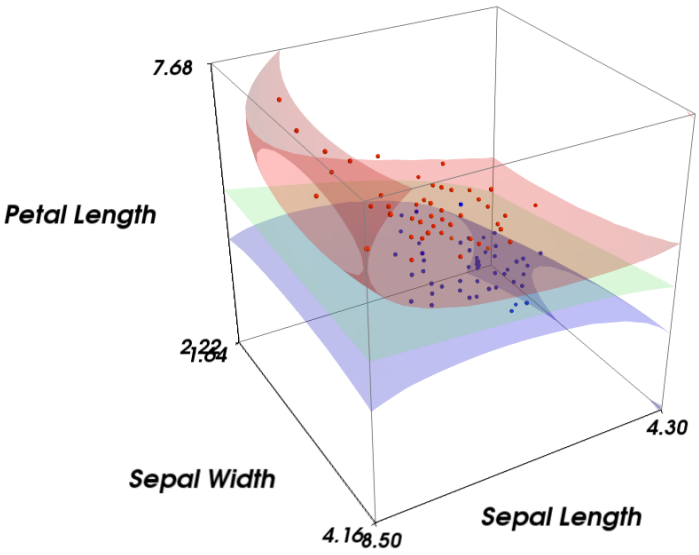


Fig. 4.4 Visualisation of 3-D data and GP model.

An interesting feature that we designed specifically for 3-D plots is the capability of generating a short animation (in `mp4` or `gif` format) in which the 3-D plot is automatically rotated to present views from all directions, as shown in Fig. 4.5. Although a report can only display a static view, future online projects may be able to take advantage of the animated visualisations.

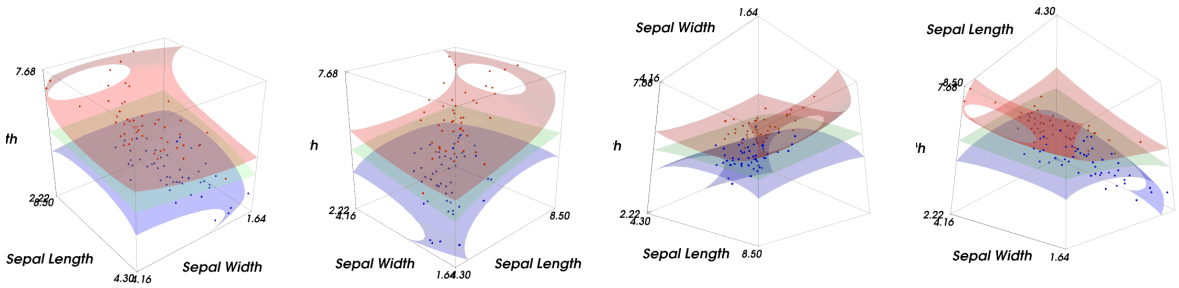


Fig. 4.5 Frames extracted from an animated visualisation of 3-D data and GP model.

4.4 Higher-dimensional Data

As discussed in Section 4.1, techniques based on dimensionality reduction are not suitable for our work. Instead, we seek simpler approaches which do not transform the input space.

The *parallel coordinates diagram* provides an elegant visualisation that meets our needs. In this graph, we create D parallel vertical axes, one for each input dimension. A D-dimensional data point is represented by a *ploy-line*: a series of line segments connecting corresponding values on each axis. The colour code for the two classes is the same as before. See Fig. 4.6 for an example.

The ordering of these axes is important, for patterns involving more than one dimension are the most discernible when the relevant axes are placed next to each other [9]. The study of the optimal ordering is beyond the scope of this project. Instead, we simply use the original ordering as given in the dataset.

It is not clear how one could overlay the predictive GP model on this plot. Therefore this visualisation is usually used to present datasets rather than trained models.

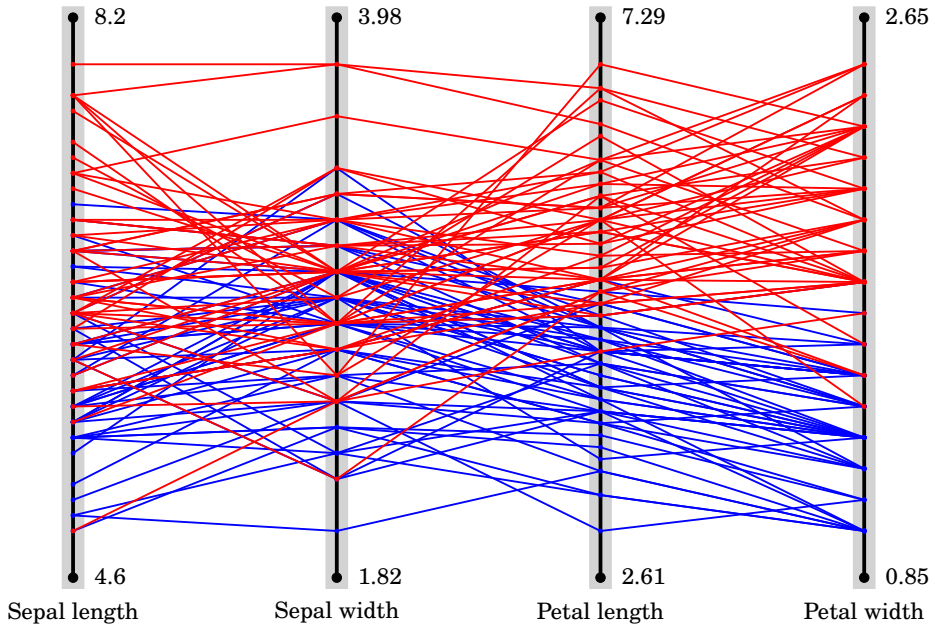


Fig. 4.6 Visualisation of a 4-D dataset.

Chapter 5

Automatic Pattern Analysis for Gaussian Process Classification

The final goal of the Automatic Statistician is to generate a data analysis report which describes the pattern underlying a given dataset in a quantitative and comprehensible way. In the context of binary classification, this requires a systematic way of associating the class labels with contributions from individual input variables and combinations of input variables.

With a structure discovery algorithm (Chapter 3) and a visualisation scheme (Chapter 4) in place, we are now ready to take the first steps towards the goal of automatic pattern analysis and report generation for Gaussian process classification. In this chapter, we first outline the overall structure of the generated report. Then, methods for creating the constituent sections of the report are proposed, with reference to the kernel grammar specified in Section 3.3.

A full-length data analysis report generated by our method can be found in Appendix A.

5.1 Related Work

There has been considerable research effort in the area of natural language generation with a concentration on large-scale systems. The scope of natural language required by AutoGPC is, however, quite limited, so we just naïvely hard-code the necessary phrases and sentences.

Regarding the analysis and description of Gaussian process models, Rasmussen and Williams [23] presented a manual analysis and description of a time series dataset by GP regression with handcrafted GP kernels. Duvenaud et al. [4] later automated the process of kernel selection for GP regression. See Section 3.1 for details.

Lloyd et al. [14] was then able to further develop the kernel search procedure in [4] and, for the first time, generate natural-language reports for GP time series regression in a fully

automated manner. They showed that, with carefully choosen base kernels and an appropriate search procedure, the resulting compositional kernel could be decomposed as a sum of products of base kernels (the *canonical form*). They designed a natural-language template for each of the base kernels, and a template for describing multiplicative action between base kernels. The canonical form could then be readily described one summand at a time, using the predefined templates. Their code is available on GitHub.[1]

Our approach is influenced by [14], but has significant differences. First and foremost, we work on classification problems, which means most of the language built for regression cannot be reused. Second, we must be able to analyse the contribution and interaction of multiple input variables (dimensions), whereas the time series analysis only needs to deal with one input variable: the time. Third, our base kernels and search operators are not the same as those in [14].

5.2 Overview

The AutoGPC report is a self-contained document written in the tone of the Automatic Statistician.

It begins with a brief introduction of the training dataset, which includes the dimensionality of the input space, the name of each input variable, and the interpretation of the positive and negative class labels, if supplied by the user. A visualisation of the entire dataset is also generated with the techniques in Chapter 4.

It consists of three sections: Executive Summary, Individual Variable Analysis and Additive Component Analysis. We will see how each section is compiled in the next few sections.

LaTeX is used for typesetting.

5.3 Executive Summary

The Executive Summary section has two main parts. The first part identifies the best model found in the search procedure. The next part ranks the input variables according to their relevance. As noted in Section 3.2, the models are evaluated and compared by their cross-validated classification error.

[1] https://github.com/jamesrobertlloyd/gpss-research

5.3.1 Verbal Description of a Sum-of-Products Kernel

The best model (kernel) is often not a simple sum kernel or product kernel. Rather, it often involves both '$+$' and '$\times$' in the same expression. In order to describe such a kernel verbally, we first expand it into the sum-of-products form using the implementation of [4, 14]. The sum-of-products expression is then translated to natural language using the following context-free grammar:[1]

$$
\begin{aligned}
\langle \textit{kernel} \rangle &::= \text{`}\texttt{only }\text{'}, \langle \textit{term} \rangle \\
&\quad\ \mid \text{`}\texttt{an additive combination of }\text{'}, \langle \textit{terms} \rangle \\
\langle \textit{terms} \rangle &::= \{\langle \textit{term} \rangle, \text{`}\texttt{, }\text{'}\}, \langle \textit{term} \rangle, \text{`}\texttt{ and }\text{'}, \langle \textit{term} \rangle \\
\langle \textit{term} \rangle &::= \text{`}\texttt{variable }\text{'}, \langle \textit{var} \rangle \\
&\quad\ \mid \text{`}\texttt{a }\text{'}, \langle \textit{n} \rangle, \text{`}\texttt{-way interaction between variables }\text{'}, \langle \textit{vars} \rangle \\
\langle \textit{vars} \rangle &::= \{\langle \textit{var} \rangle, \text{`}\texttt{, }\text{'}\}, \langle \textit{var} \rangle, \text{`}\texttt{ and }\text{'}, \langle \textit{var} \rangle
\end{aligned}
$$

Here, $\langle \textit{kernel} \rangle$ is the verbal description of the sum-of-products kernel that we want; $\langle \textit{term} \rangle$ is the verbal description of a single additive term in the sum-of-products kernel; $\langle n \rangle$ is the number of input variables involved in the additive term; and $\langle \textit{var} \rangle$ is the name of a input variable. A kernel expression is appended to the end of this verbal description for ease of comprehension in case the verbal description is long.

For example, suppose a dataset has three input dimensions which represent variables 'plasma glucose', 'body mass index' and 'age' respectively. To describe kernel $\mathrm{SE}_1 \times (\mathrm{SE}_2 + \mathrm{SE}_3)$, we first expand it as $\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_1 \times \mathrm{SE}_3$. The corresponding verbal description is then

> [The model involves] an additive combination of a 2-way interaction between variables 'plasma glucose' and 'body mass index' and a 2-way interaction between variables 'plasma glucose' and 'age' ($\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_1 \times \mathrm{SE}_3$).

This translation framework is also used in other parts of the report, where the verbal description of a kernel is needed.

5.3.2 Summary of Input Variables and Their Relevance

Another important purpose of the Executive Summary section is to give a concise summary of all input variables. Each input variable is tabulated with its name, simple statistics (minimum, maximum and mean) and the best classification performance achieved using this input variable alone. This gives the reader an overall feeling of how each input dimension contributes to the class label assignment.

[1] *Extended Backus–Naur Form* (Extended BNF or EBNF) notation is used.

The input variables are ranked according to their *relevance*. Although we do not have an algebraic definition of relevance, it can be inferred from the one-dimensional classifier performance. In specific, we assume without proof that a lower cross-validated error implies a higher level of relevance. In case two cross-validated errors are the same (or very close to each other), the model with a lower NLML is preferred. The performance of all one-dimensional classifiers is readily available from the first-layer nodes of the search tree (Section 3.4).

For example, if the base kernels are defined as $\{\mathrm{SE}_i \mid i = 1, \ldots, 4\}$, the summary of variables may look like Table 5.1.

Table 5.1 A table generated by AutoGPC that summarises all input variables

		Statistics			Classifier Performance		
Dimension	Variable	Min	Max	Mean	Kernel	NLML	Error
1	plasma glucose	44.00	199.00	121.88	smooth	381.22	**25.00%**
2	body mass index	18.20	67.10	32.47	smooth	347.18	33.71%
3	pedigree function	0.08	2.42	0.47	smooth	366.12	33.71%
–	*Baseline*	–	–	–	*constant*	*373.79*	*34.39%*
4	age	21.00	81.00	33.35	smooth	**344.69**	34.54%

Note that we have included a baseline performance in the table. It is discussed in detail in the next section.

5.4 Individual Variable Analysis

The Individual Variable Analysis section develops upon the previous section of the AutoGPC report and provides an in-depth review for each of the input variables. The review not only quotes the numbers listed in Table 5.1, but also attempts to draw conclusion about the predictive power of each input variable in terms of class label prediction. It also tries to find and report any significant pattern in any one input dimension, such as monotonicity.

All ingredients needed for such analysis are included in the one-dimensional GP classification models. These models are represented by the 1-D kernels in the first-layer nodes of the search.

In this section, we first introduce a baseline performance against which all trained classifiers are compared. Then, we move on to the detection of monotonicity given an input dimension. It is also possible to test for periodicity and linearity with periodic (PER) and linear (LIN) kernels respectively, although we do not currently include these analyses in AutoGPC.

5.4.1 Baseline Performance

The *baseline* performance is achieved with a *constant kernel* (C). It is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2, \tag{5.1}$$

and thus not a function of $\mathbf{x}$ or $\mathbf{x}'$. Training a GP classification model with the constant kernel will result in a constant mean function $m(\mathbf{x}) \equiv m$ whose transformed value $\sigma(m)$ is the probability of a randomly drawn training sample being positive (Fig. 5.1).

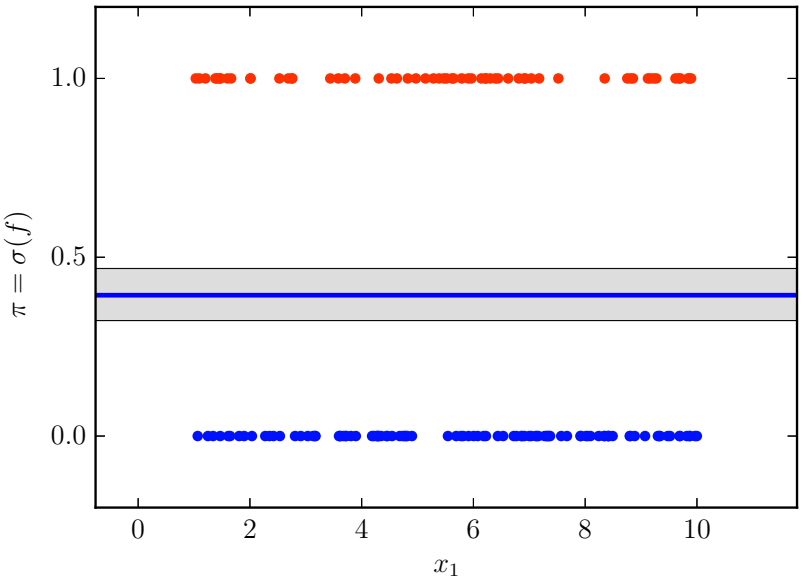


Fig. 5.1 GP posterior with a constant kernel sliced to 1-D.

More intuitively, the baseline classifier will indiscriminately classify all test points as the majority class of the training dataset. The resulting error rate will be the frequency of the minority class. Thus, it is a simple yet useful criterion regarding the relative significance of a classifier (or a variable) in class label determination.

We categorise one-dimensional GP classifiers into four types based on the ratio between their cross-validated error rate (ϵ) and the baseline error rate (ϵ_0). Each type is described by a standard sentence, as shown in Table 5.2. Note that this categorisation is entirely empirical.

5.4.2 Monotonicity

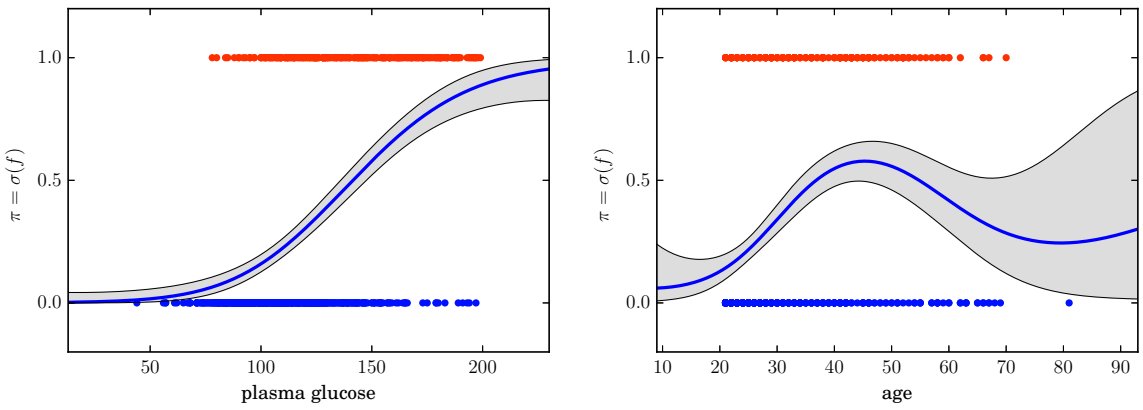
Monotonicity is defined for any input variable as a purely positive or negative correlation between the probability of the positive class ('class 1') and the value of that input variable. We infer

Table 5.2 Standard qualitative description of the significance of an input variable

Range of ϵ/ϵ_0	Verbal Description
0 – 0.25	Compared with the null model (baseline), this variable contains strong evidence whether the sample belongs to ... class.
0.25 – 0.8	Compared with the null model (baseline), this variable contains some evidence of class label assignment.
0.8 – 1	This variable provides little evidence of class label assignment given a baseline error rate of ...%.
1 – ∞	The classification performance (in terms of error rate) based on this variable alone is even worse than that of the naïve baseline classifier.

monotonicity from the posterior latent function $f(x)$ of the one-dimensional squared exponential (SE) classifiers.[1]

Strictly speaking, since the GP posterior is a distribution over latent functions $f(x)$, we should compute the probability of $f(x)$ being monotonic. Nevertheless, in practice we take a less rigorous but much simpler route. We consider the gradients of the posterior mean $\mathbb{E}[f(x)]$ at all training points $x_1, \ldots, x_N$, and claim $f(x)$ is monotonic if the gradients are all positive (or negative). Training points that take extreme values on the x-axis are not counted in this process, as the posterior mean tends to return to neutral in the absence of training data. An example is shown in Fig. 5.2.



(a) Monotonic (b) Not monotonic

Fig. 5.2 Monotonicity detection in 1-D SE models.

[1]Note the argument of latent function f is scalar x, not vector $\mathbf{x}$, since we are only considering one-dimensional GP classifiers.

5.5 Additive Component Analysis

The Additive Component Analysis section tries to decouple the additive components that constitute the full model found by the search procedure.

To begin with, we obtain an ordering of summands based on their significance. It operates as follows. First, the full kernel is expanded into a sum-of-products form in the same way as in Section 5.3. Next, the additive component (*summand*) with the lowest cross-validated error is selected as the partial kernel. Then we repeatedly add one of the remaining summands to the partial kernel such that the *cumulative* error rate of the new kernel is minimised, until the full kernel is obtained. In each iteration, both the current summand and the resulting partial additive kernel are recorded for analysis. This procedure is outlined formally in Algorithm 5.1.

Algorithm 5.1 Additive component analysis

Require: k ▷ The kernel to be analysed
$S \leftarrow$ SUMOFPRODUCTS(k) ▷ Obtain list of summands
$T \leftarrow$ LIST(), $P \leftarrow$ LIST() ▷ Initialise lists of ordered summands and partial kernels
$p \leftarrow k_{\text{NULL}}$ ▷ Initialise partial sum kernel as NULL kernel
while LENGTH(S) > 0 **do**
    $Q \leftarrow$ ELEMENTWISEADD(S, p) ▷ Add partial kernel to each remaining summand
    $S \leftarrow$ SORT(S) by ERROR(Q)
    $s \leftarrow$ POP(S) ▷ Remove the first element
    $p \leftarrow p + s$
    APPEND(T, s), APPEND(P, p)
end while
return T, P

For example, kernel $\text{SE}_1 \times \text{SE}_2 + \text{SE}_3 + \text{SE}_4$ has three summands: $\text{SE}_1 \times \text{SE}_2$, SE_3 and SE_4. We first choose the one with the lowest error rate, say $\text{SE}_1 \times \text{SE}_2$. Next, both SE_3 and SE_4 can be added to $\text{SE}_1 \times \text{SE}_2$ to build the next partial kernel. Assume $\text{SE}_1 \times \text{SE}_2 + \text{SE}_4$ has a lower error rate than $\text{SE}_1 \times \text{SE}_2 + \text{SE}_3$, so SE_4 is selected as the second significant summand, and the resulting partial kernel is $\text{SE}_1 \times \text{SE}_2 + \text{SE}_4$. Finally, SE_3 is the least significant summand, and adding SE_3 to the partial kernel yields the full kernel at the beginning.

These summands and partial kernels are then described and plotted in the order of significance, as shown in Appendix A.

To conclude, we enumerate all additive components, all one-dimensional models, the full model and the baseline model in one comprehensive table, such as Table 5.3.

Table 5.3 A table generated by AutoGPC that summarises the classification performance of the full model, its additive components (if any), all input variables, and the baseline.

Dimensions	Kernel expression	NLML	Error
1, 3, 4	$\mathrm{SE}_1 + \mathrm{SE}_3 \times \mathrm{SE}_4$	**280.45**	**21.83%**
1	SE_1	381.22	25.00%
3, 4	$\mathrm{SE}_3 \times \mathrm{SE}_4$	422.79	30.80%
2	SE_2	347.18	33.71%
3	SE_3	366.12	33.71%
–	C *(Baseline)*	*373.79*	*34.39%*
4	SE_4	344.69	34.54%

Chapter 6

Datasets and Numerical Results

Both synthetic and real-world data have played a vital role in the development of AutoGPC. The synthetic datasets are usually smaller in size, and serve the purpose of unit testing. In this chapter, we list the real-world datasets that were used in the experiments, including the necessary preprocessing steps.

The main innovation of this project is to automatically generate natural-language analysis of classification datasets. A sample report generated by AutoGPC is attached in Appendix A. Classification performance is not the focus of this project. Therefore, we only quote a few numerical results in the second section of this chapter.

6.1 Datasets and Preprocessing

All of our real datasets were obtained from the UCI Machine Learning Repository [13]. The input dimensionality and number of training points of these datasets are tabulated in Table 6.1. The datasets were preprocessed mainly in order to remove corrupt entries. The preprocessing on each dataset is detailed as follows, following the suggestions in [13]:

Table 6.1 Datasets used in experiments and their sizes before and after preprocessing

		Raw		Cleaned	
Dataset	Source	D	N	D	N
Iris Dataset	[5]	4	150	4	100
Pima Indians Diabetes Dataset	[26]	8	768	4	724
BUPA Liver Disorders Dataset	[6]	5	345	5	345
Wisconsin Breast Cancer Dataset	[30]	9	699	5	683
Cleveland Heart Disease Dataset	[1]	13	303	6	297

- **Iris:** The original dataset contains three classes, each having 50 data points. We retained only the *Iris versicolor* and *Iris virginica* classes.
- **Pima:** Attributes 'plasma glucose', 'diastolic blood pressure' and 'body mass index' contain clearly corrupt records marked by zero. These data points were removed. The input space was reduced to four dimensions ('plasma glucose concentration', 'body mass index', 'diabetes pedigree function' and 'age') in order to accelerate experiments and allow faster development cycles. These four dimensions were chosen as they were found to be the most relevant by Mrkšić [18].
- **BUPA:** The last attribute 'selector field', although binary, is actually not an appropriate training target [6]. Instead, attribute 'drinks' should be dichotomised by whether or not it is greater than 5 to form a training target. All other attributes were used to form a 5-dimensional input space.
- **Wisconsin:** Missing values are denoted '?' in the original dataset, which we removed. For the same reason as Pima dataset, we reduced the input space to five dimensions ('clump thickness', 'uniformity of cell size', 'single epithelial cell size', 'bare nuclei' and 'normal nucleoli').
- **Cleveland:** 6 data points have missing values, which we excluded from our datasets. Binary and categorical input variables were discarded. The remaining input variables were 'age', 'trestbps', 'chol', 'thalach', 'oldpeak' and 'ca'.

6.2 Test Results

Experiments on the datasets yielded classification performance shown in Table 6.2, which is comparable to the state of the art. Only squared exponential (SE) kernels were used as base kernels. Note that the datasets were preprocessed as described above, so caution should be exercised when comparing our results with others' work. Due to random initialisation, the exact form of the best full model may vary slightly between trials.

Table 6.2 Classification performance of AutoGPC on real datasets

	Best 1-D model			Best full model		
Dataset	Kernel	NLML	Error	Kernel	NLML	Error
Iris	SE_4	21.31	6.00%	SE_4	21.31	6.00%
Pima	SE_1	308.93	24.45%	$\mathrm{SE}_1 + \mathrm{SE}_3 \times \mathrm{SE}_4$	280.45	21.83%
BUPA	SE_3	150.17	21.74%	$\mathrm{SE}_1 \times \mathrm{SE}_3 + \mathrm{SE}_2 \times \mathrm{SE}_5$	138.13	18.84%
Wisconsin	SE_2	96.80	7.61%	$(\mathrm{SE}_2 + \mathrm{SE}_3) \times (\mathrm{SE}_4 + \mathrm{SE}_5)$	62.52	2.63%
Cleveland	SE_6	139.92	25.57%	$\mathrm{SE}_4 \times \mathrm{SE}_6 + \mathrm{SE}_5$	120.95	22.88%

Chapter 7

Conclusion

In this project, we have built AutoGPC, an Automatic Statistician for general binary classification problems. The system is based on Gaussian processes, and is capable of constructing and describing the nonparametric model underlying the dataset. The resulting data analysis report is written in natural language, and is, to the best of our knowledge, the first of its kind for classification tasks.

We verified that it is feasible to achieve automatic structure discovery for binary classification by means of a greedy search of the compositional kernel space [18]. A beam search procedure was specified, and a set of empirical constraints were applied to hyperparameters to prevent overfitting and underfitting. The base kernels were restricted to one-dimensional squared exponential (SE) kernels, but could be easily extended to include other forms of one-dimensional kernels.

We designed a visualisation library that can create intuitive graphs for GP classification models up to 3-D, and for classification datasets up to any dimensionality. In addition, models or datasets with many input dimensions can be projected to a manually selected lower-dimensional subspace (1-, 2- or 3-D), where plotting is less tricky.

We created a systematic method for the interpretation and presentation of GP classification models. By considering the one-dimensional GP classifiers, we were able to infer the nature of the relationship between the class labels and individual input variables (e.g. monotonicity); by considering the sum-of-products decomposition of the full GP kernel, we could decouple different additive patterns and describe them separately. Cross-validated classification error provides a convenient measure of significance, which is used throughout the data analysis report to compare candidate kernels.

The tests on the five real-world datasets demonstrated that AutoGPC was indeed able to find a good model of the given dataset, with the performance on a par with the state of the art. These tests also confirmed that our pattern analysis and interpretation techniques were able to create high-quality natural-language description of a general GP classification model.

Future Work

While AutoGPC is able to create automatically generated data analysis reports for binary classification problems, it can certainly be extended in a few promising directions.

The base kernels could be redesigned. Currently the base kernels only include squared exponential kernels (SE). The support for periodic kernels (PER) was added in AutoGPC but not tested with real applications. Linear kernels (LIN) might be added in the future to distinguish between linear and nonlinear patterns in individual input variables. Nonstandard kernels might also be considered if they suit the classification job well.

Quantitative tests could be conducted on the performance of sparse GP methods in the context of classification and report generation. Sparse GP techniques such as [10] are already implemented in GPy [7]. They are advantageous when the training dataset has a large number of training points. It would be interesting to quantitatively compare their classification accuracy and robustness with those of the conventional 'full' GP methods, especially for the purpose of AutoGPC.

A more elegant approach could be taken towards occasional missing and spurious dataset entries. Currently we have to discard the entire data point for just one missing value, which means the valuable information contained in the other normal values is also wasted.

AutoGPC could be generalised to support multi-class problems. This would probably require an extension of the visualisation library and a revamp of the model description language.

Finally, the idea of Automatic Statistician may be applied beyond the realm of Gaussian processes. It is totally possible to develop an Automatic Statistician programme for neural networks, for example, if suitable constraints on the network and corresponding description strategy could be defined.

References

[1] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J.-J. Schmid, S. Sandhu, K. H. Guppy, S. Lee, and V. Froelicher, "International application of a new probability algorithm for the diagnosis of coronary artery disease," *The American Journal of Cardiology*, vol. 64, no. 5, pp. 304–310, Aug. 1, 1989.

[2] D. K. Duvenaud, "Automatic model construction with gaussian processes," PhD thesis, University of Cambridge, Jun. 2014.

[3] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, "Additive gaussian processes," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 226–234.

[4] D. Duvenaud, J. R. Lloyd, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani, "Structure discovery in nonparametric regression through compositional kernel search," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 1166–1174.

[5] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[6] R. Forsyth and R. Rada, *Machine learning: Applications in expert systems and information retrieval*. Halsted Press, 1986.

[7] GPy, *Gpy: A gaussian process framework in python*, http://github.com/SheffieldML/GPy, 2012.

[8] R. B. Grosse, R. R. Salakhutdinov, W. T. Freeman, and J. B. Tenenbaum, "Exploiting compositionality to explore a large space of model structures," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 2012.

[9] J. Heer, M. Bostock, and V. Ogievetsky, "A tour through the visualization zoo," *Communications of the ACM*, vol. 53, no. 6, pp. 59–67, Jun. 2010.

[10] J. Hensman, A. G. d. G. Matthews, and Z. Ghahramani, "Scalable variational gaussian process classification," in *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015, pp. 351–360.

[11] M. Kuss and C. E. Rasmussen, "Assessing approximate inference for binary gaussian process classification," *Journal of Machine Learning Research*, vol. 6, pp. 1679–1704, Dec. 2005.

[12] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse gaussian process methods: The informative vector machine," in *Advances in Neural Information Processing Systems 15*, MIT Press, 2003, pp. 609–616.

[13] M. Lichman, *UCI machine learning repository*, http://archive.ics.uci.edu/ml, 2013.

[14] J. R. Lloyd, D. Duvenaud, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani, "Automatic construction and natural-language description of nonparametric regression models," in *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, Jun. 21, 2014, pp. 1242–1250.

[15] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov 2008.

[16] D. J. C. MacKay, "Introduction to gaussian processes," in *Neural Networks and Machine Learning*, C. M. Bishop, Ed., Berlin: Springer, 1998.

[17] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, California: Morgan Kaufmann Publishers Inc., 2001, pp. 362–369.

[18] N. Mrkšić, "Kernel structure discovery for gaussian process classification," M.Eng. Dissertation, University of Cambridge, Cambridge, United Kingdom, Jun. 4, 2014.

[19] A. Naish-Guzman and S. Holden, "The generalized FITC approximation," in *Advances in Neural Information Processing Systems 20*, MIT Press, 2007, pp. 1057–1064.

[20] R. M. Neal, "Regression and classification using gaussian process priors," in *Bayesian Statistics 6*, J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, Eds., Oxford, United Kingdom: Oxford University Press, Aug. 12, 1999.

[21] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning*, ser. Lecture Notes in Computer Science 3176, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds., Berlin: Springer, 2004, pp. 63–71.

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

[23] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: MIT Press, 2006, xviii, 248 p.

[24] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 22, 2000.

[25] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, Apr. 3, 2009.

[26] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*, American Medical Informatics Association, 1988, p. 261.

[27] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 22, 2000.

[28] L. Todorovski and S. Dzeroski, "Declarative bias in equation discovery," in *Proceedings of the 14th International Conference on Machine Learning*, 1997, pp. 376–384.

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

[30] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology.," *Proceedings of the National Academy of Sciences*, vol. 87, no. 23, pp. 9193–9196, 1990.

Appendix A

Sample Report Generated by AutoGPC

The main achievement of this project is the automatic discovery and description of the pattern underlying a general binary classification dataset, which is best demonstrated with a solid example.

We include overleaf a full-length data analysis report generated automatically by AutoGPC. The report presents an automatic interpretation of the Pima Indian Diabetes Dataset [26], including the relevance of each input variable and the decomposition of the full model. The report is reader-friendly and includes quantitative and qualitative details as desired.

AutoGPC Data Analysis Report on Pima Dataset

The Automatic Statistician

24th May 2016

This report is automatically generated by the AutoGPC. AutoGPC is an automatic Gaussian process (GP) classifier that aims to find the structure underlying a dataset without human input. For more information, please visit `http://github.com/charles92/autogpc`.

The training dataset spans 4 input dimensions, which are variables 'plasma glucose', 'body mass index', 'pedigree function' and 'age'. There is one binary output variable y, where $y = 0$ (negative) represents 'not diabetic', and $y = 1$ (positive) represents 'diabetic'.

The training dataset contains 724 data points. Among them, 249 (34.39%) have positive class labels, and the other 475 (65.61%) have negative labels. All input dimensions as well as the class label assignments are plotted in Figure 1.

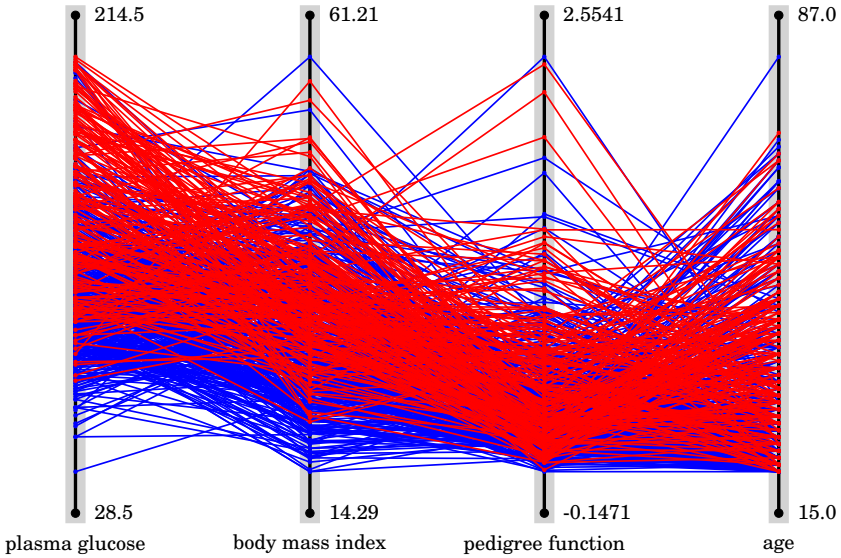


Figure 1: The input dataset. Positive samples ('diabetic') are coloured red, and negative ones ('not diabetic') blue.

1 Executive Summary

The best kernel that we have found relates the class label assignment to input variables 'plasma glucose', 'pedigree function' and 'age'. In specific, the model involves an additive combination

of variable 'plasma glucose' and a 2-way interaction between variables 'pedigree function' and 'age' ($\mathrm{SE}_1 + \mathrm{SE}_3 \times \mathrm{SE}_4$). This model can achieve a cross-validated error rate of 22.38% and a negative log marginal likelihood of 353.28.

We have also analysed the relevance of each input variable. They are listed below in Table 1 in descending order of inferred relevance (i.e. in ascending order of cross-validated error of the best one-dimensional GP classifier).

Table 1: Input variables

		Statistics			Classifier Performance		
Dimension	Variable	Min	Max	Mean	Kernel	NLML	Error
1	plasma glucose	44.00	199.00	121.88	smooth	381.22	**25.00%**
2	body mass index	18.20	67.10	32.47	smooth	347.18	33.71%
3	pedigree function	0.08	2.42	0.47	smooth	366.12	33.71%
–	*Baseline*	–	–	–	*constant*	*373.79*	*34.39%*
4	age	21.00	81.00	33.35	smooth	**344.69**	34.54%

In the rest of the report, we will first describe the contribution of each individual input variable (Section 2). This is followed by a detailed analysis of the additive components that jointly make up the best model (Section 3).

2 Individual Variable Analysis

First, we try to classify the training samples using only one of the 4 input dimensions. By considering the best classification performance that is achievable in each dimension, we are able to infer which dimensions are the most relevant to the class label assignment.

Note that in Table 1 the baseline performance is also included, which is achieved with a constant GP kernel. The effect of the baseline classifier is to indiscriminately classify all samples as either positive or negative, whichever is more frequent in the training data. The classifier performance in each input dimension will be compared against this baseline to tell if the input variable is discriminative in terms of class determination.

2.1 Variable 'plasma glucose'

Variable 'plasma glucose' has mean value 121.88 and standard deviation 30.73. Its observed minimum and maximum are 44.00 and 199.00 respectively. A GP classifier trained on this variable alone can achieve a cross-validated error of 25.00%.

Compared with the null model (baseline), this variable contains some evidence of class label assignment. There is a positive correlation between the value of this variable and the likelihood of the sample being classified as positive. The GP posterior trained on this variable is plotted in Figure 2.

2.2 Variable 'pedigree function'

Variable 'pedigree function' has mean value 0.47 and standard deviation 0.33. Its observed minimum and maximum are 0.08 and 2.42 respectively. A GP classifier trained on this variable alone can achieve a cross-validated error of 33.71%.

This variable provides little evidence of class label assignment given a baseline error rate of 34.39%. The GP posterior trained on this variable is plotted in Figure 3.

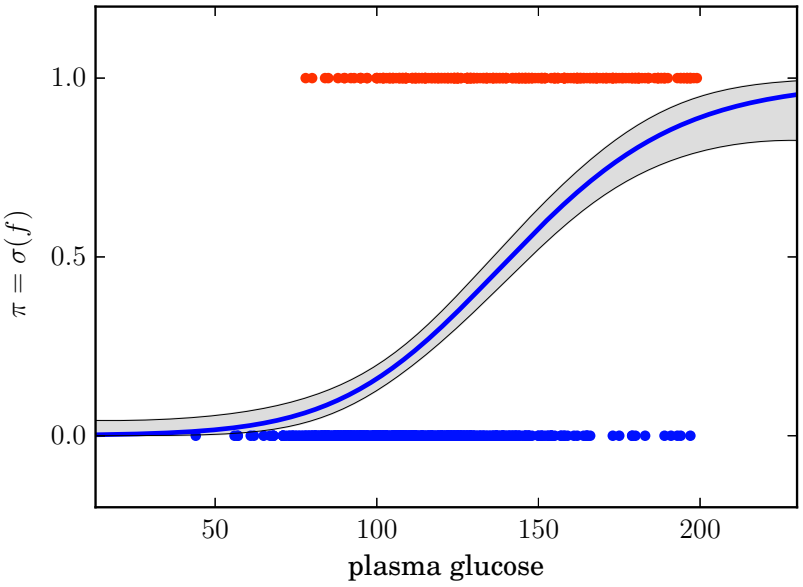


Figure 2: Trained classifier on variable ‘plasma glucose’.

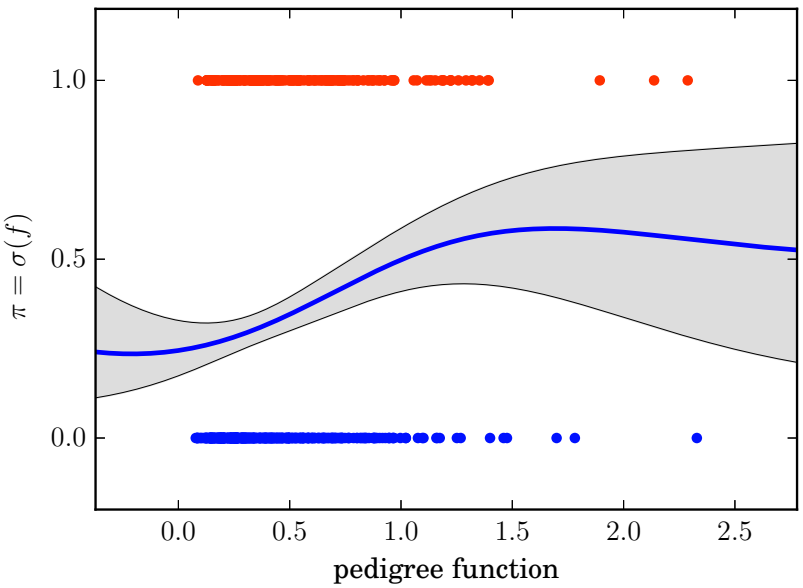


Figure 3: Trained classifier on variable ‘pedigree function’.

2.3 Variable ‘body mass index’

Variable ‘body mass index’ has mean value 32.47 and standard deviation 6.88. Its observed minimum and maximum are 18.20 and 67.10 respectively. A GP classifier trained on this variable alone can achieve a cross-validated error of 33.71%.

This variable provides little evidence of class label assignment given a baseline error rate of 34.39%. The GP posterior trained on this variable is plotted in Figure 4.

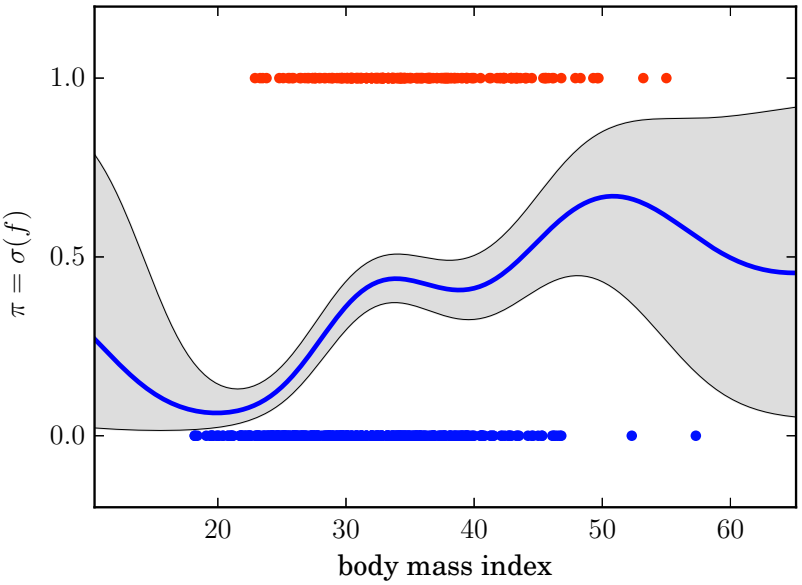


Figure 4: Trained classifier on variable ‘body mass index’.

2.4 Variable ‘age’

Variable ‘age’ has mean value 33.35 and standard deviation 11.76. Its observed minimum and maximum are 21.00 and 81.00 respectively. A GP classifier trained on this variable alone can achieve a cross-validated error of 34.54%.

The classification performance (in terms of error rate) based on this variable alone is even worse than that of the naïve baseline classifier. The GP posterior trained on this variable is plotted in Figure 5.

3 Additive Component Analysis

The pattern underlying the dataset can be decomposed into 2 additive components, which contribute jointly to the final classifier which we have trained. With all components in action, the classifier can achieve a cross-validated error rate of 22.38%. The performance cannot be further improved by adding more components.

In Table 2 we list the full additive model, all input variables, as well as more complex additive components (if any) considered above, ranked by their cross-validated error rate.

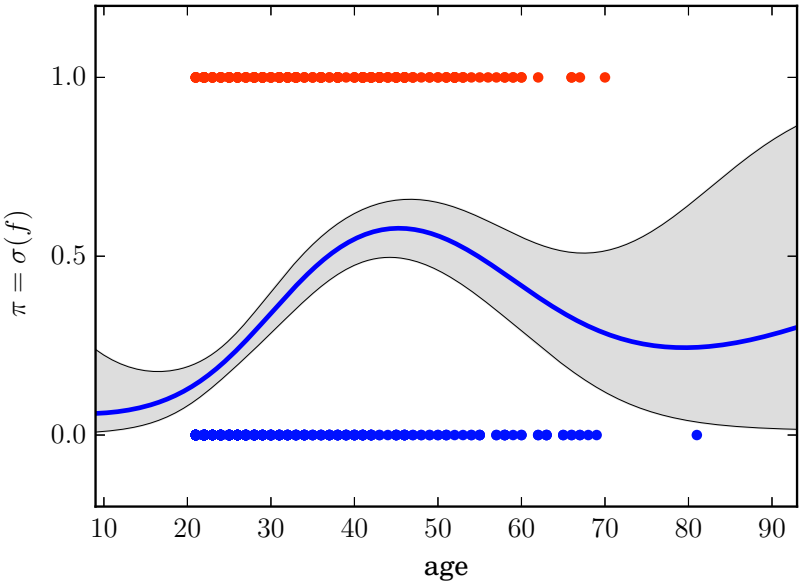


Figure 5: Trained classifier on variable 'age'.

Table 2: Classification performance of the full model, its additive components (if any), all input variables, and the baseline.

Dimensions	Kernel expression	NLML	Error
1, 3, 4	$\mathrm{SE}_1 + \mathrm{SE}_3 \times \mathrm{SE}_4$	**280.45**	**21.83%**
1	SE_1	381.22	25.00%
3, 4	$\mathrm{SE}_3 \times \mathrm{SE}_4$	422.79	30.80%
2	SE_2	347.18	33.71%
3	SE_3	366.12	33.71%
–	C *(Baseline)*	*373.79*	*34.39%*
4	SE_4	344.69	34.54%

3.1 Component 1

With only one additive component, the GP classifier can achieve a cross-validated error rate of 25.00%. The corresponding negative log marginal likelihood is 381.22. This component operates on variable ‘plasma glucose’, as shown in Figure 6.

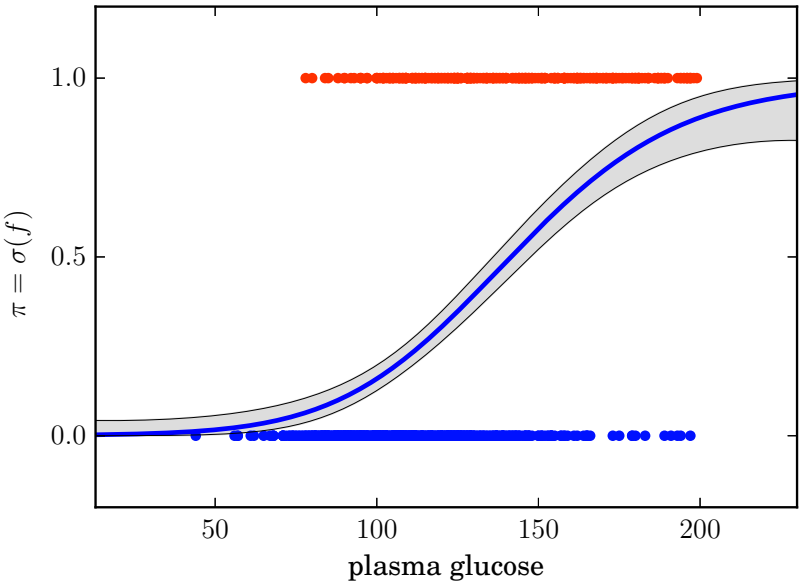
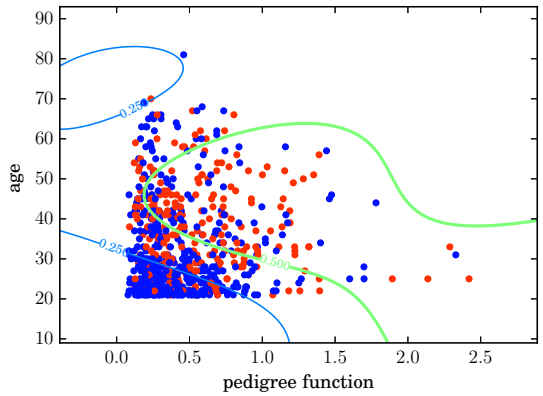


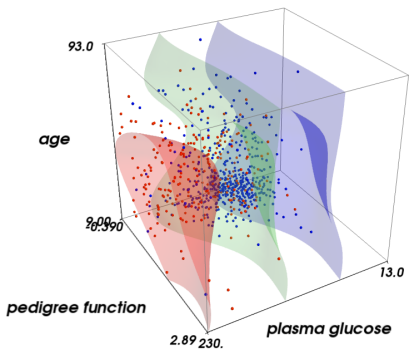
Figure 6: Trained classifier on variable ‘plasma glucose’.

3.2 Component 2

With 2 additive components, the cross-validated error can be reduced by 2.62% to 22.38%. The corresponding negative log marginal likelihood is 353.28. The additional component operates on variables ‘pedigree function’ and ‘age’, as shown in Figure 7.



(a) Current additive component involving variables ‘pedigree function’ and ‘age’.



(b) Previous and current components combined, involving variables ‘plasma glucose’, ‘pedigree function’ and ‘age’.

Figure 7: Trained classifier on variables ‘plasma glucose’, ‘pedigree function’ and ‘age’.

Appendix B

Risk Assessment

The Risk Assessment Form was submitted to the Department Safety Office on 7 October 2015, before the project started.

Given the nature of the project, the only potential hazard identified on the Form was computer use:

> 'Potential ergonomic concerns from extensive use of computers. Care should be taken to use correct posture and to take breaks regularly.'

This was indeed the main health and safety issue during the entire course of the project. In Michaelmas term, the author was able to utilise the office space of the Computational and Biological Learning Laboratory (CBL) where adjustable LCD monitors and external keyboards were available. It was thus much more comfortable than working directly on a laptop. Unfortunately, the office became fully occupied after the Christmas Vacation. Adding such free working space for fourth-year projects would be very helpful for future students.

There were no other health and safety issues.