GAUSSIAN PROCESS REGRESSION – A MACHINE LEARNING APPROACH TO DERIVATIVE PRICING

by

Haibo Li (CID: 01441128)

Department of Mathematics
Imperial College London
London SW7 2AZ
United Kingdom

Thesis submitted as part of the requirements for the award of the MSc in Mathematics and Finance, Imperial College London, 2017-2018

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signature and date:

Haibo Li

11/09/2018

Acknowledgements

I would like to thank my supervisor at Imperial College, Dr. Thomas Cass for his constant help and support, as well as for his insightful suggestions.

I would also like to thank my manager at Citigroup, Karim Berradi for giving me the opportunity to do this Placement project on his team. I am most grateful to his help and numerous inspirational guidance during the past three months.

I also want to thank other members on the team: Håvard Sandvik, Nikos Karantzoulis, Stephane Guilherme Gomes, Mohamed Boualam, He Ren and Yacine Debbabi for their help in many aspects which made the past three months a very pleasant experience.

Contents

1 Introduction

One of the challenges when pricing some type of derivatives is performance in terms of speed. This is typically the case for structured products with a large number of underlyings. Usual numerical and programming techniques can be used to improve that performance, however in certain application where the derivatives need to be re-priced a large number of times, performance in terms of speed becomes extremely critical. One way to tackle this problem is using a trained machine learning model that can replicate prices. In this paper, one class of regression model, the Gaussian Process Regression(GPR) is investigated for the purpose of replicating the price of one particular structured product, namely Synthetic Collateralized Debt Obligation.

1.1 CDS, CDD and Synthetic CDO

The focus of this paper is not to present the mathematical sophistication behind the pricing models which have already been extensively studied in numerous works in the past. The following paragraphs will only briefly introduce the pricing mechanism of Credit Default Swap(CDS), Collateralized Debt Obligation(CDO) and Synthetic CDO. A thorough discussion about the mathematical details of the pricing formula is out of the scope of this paper. More details about credit derivatives can be found in Elouerkhaoui's book[1], Brigo's book[2] and O'Kane's book[3].

A CDS contract provides protection against default. Typically, in a CDS contract, there are protection buyer, protection seller and the reference entity. Like in an insurance contract, the protection buyer pays the premium leg to the protection seller given there is no default; the protection seller pays the protection leg to the protection buyer if the reference entity defaults before the maturity of the contract. Assume face value of \$1, the present value(PV) of a CDS is the difference between PV of the premium leg and protection leg. Exclude accrued coupon payment for simplicity, the PV of premium leg and protection leg are given by

$$\text{Premium PV} = S_0[\sum_{n=1}^{N} \Delta(t_{n-1}, t_n) Z(t, t_n) Q(t, t_n)], \tag{1.1}$$

and

$$\text{Protect PV} = (1-R)\int_t^T Z(t,s)(-dQ(t,s)), \tag{1.2}$$

where t is the effective date of the CDS contract; $Z(t,T)$ is the Libor discount curve; $Q(t,T)$ is the survival probability of the reference entity at time t to T; $t_n, n = 1, ..., N$ are the premium payment dates; S_0 represents $S(0,T)$, the fixed contractual spread at time 0 which matures at T; $\Delta(t_{n-1}, t_n)$ is the day count fraction between t_{n-1} and t_n; R is the expected recovery rate as a percentage. For a detailed discussion about CDS, please see [3, chapter 5].

A Collateralized Debt Obligation(CDO) is a structured asses-backed security whose payoff is specified by the total loss of the portfolio. A CDO is typically divided into different tranches with different seniority based on level of risk. A synthetic CDO is a CDO with underlying portfolio made of reference CDSs. A CDO tranche is defined by an attachment point K_1 and a detachment point K_2, which determine the range of loss that will affect the payoff. Denote the cumulative percentage default loss on the preference portfolio at time T as $L(T)$ and the CDO tranche loss at time T as $L(T, K_1, K_2)$, then we have

$$L(T, K_1, K_2) = \frac{\max(L(T) - K_1, 0) - \max(L(T) - K_2, 0)}{K_2 - K_1}. \tag{1.3}$$

The tranche premium leg is a series of cash flows paid by the tranche protection buyer to the seller. The amount of the payment is determined by the tranche spread S, which is agreed when signing the contract and the remaining tranche principal $\mathbb{E}[1 - L(t_i, K_1, K_2)]$. The present value of tranche premium leg at time zero is

$$\text{Premium PV} = S\sum_{i=1}^{N_T} \Delta(t_{i-1}, t_i)Z(t_i)\mathbb{E}[1 - L(t_i, K_1, K_2)], \tag{1.4}$$

where $Z(t_i)$ is the Libor discount curve; $t_i, n = 1, ..., T_N$ are the premium payment dates; $\Delta(t_{i-1}, t_i)$ is the day count fraction between t_{i-1} and t_i. Here we assume that the premium paid at time t_i is on the tranche notional value at t_i. One possible approximation would be taking the tranche notional as the average tranche notional since last premium payment. Then one gets

$$\text{Premium PV} = S\sum_{i=1}^{N_T} \Delta(t_{i-1}, t_i)Z(t_i)\mathbb{E}\left[1 - \frac{L(t_{i-1}, K_1, K_2) + L(t_i, K_1, K_2)}{2}\right], \tag{1.5}$$

The protection leg pays the tranche loss at the time of loss. The amount of loss on the tranche over a small period of time is given by $dL(t, K_1, K_2)$.The present value of the protection leg at time zero is

$$\text{Protection PV} = \int_0^T Z(s)\mathbb{E}[dL(s, K_1, K_2)]. \tag{1.6}$$

Analogously to the pricing equations of CDS, if we define the survival probability of tranche as

$$Q(t, K_1, K_2) = \mathbb{E}[1 - L(t, K_1, K_2)], \tag{1.7}$$

then Premium PV and Protection PV are

$$\begin{aligned} \text{Premium PV} &= \frac{S}{2}\sum_{i=1}^{N_T} \Delta(t_{i-1}, t_i)Z(t_i)[Q(t_{i-1}, K_1, K_2) + Q(t_i, K_1, K_2)], \\ \text{Protection PV} &= \int_0^T Z(s)(-dQ(s, K_1, K_2)). \end{aligned} \tag{1.8}$$

Then the present value at time zero of the tranche for the protection seller is

$$\begin{aligned} V(K_1, K_2) =& \frac{S}{2}\sum_{i=1}^{N_T} \Delta(t_{i-1}, t_i)Z(t_i)[Q(t_{i-1}, K_1, K_2) + Q(t_i, K_1, K_2)] \\ &- \int_0^T Z(s)(-dQ(s, K_1, K_2)). \end{aligned} \tag{1.9}$$

For a detailed discussion about pricing mechanism for CDOs, please see chapter 12 in [3].

The loss distribution of a CDO tranche will depend on the correlation of the underlying credits. When the correlation is zero, the underlying credits are independent which means they do not tend to survive or default together. When the correlation is high, the underlying credits become more likely to survive or default together. There are several methods to model the credit correlation, see [1] and [3, Part II].

Pricing a structured product like a synthetic CDO tranche with a lot of underlying CDSs could be challenging. One has to first get the implied default probabilities for CDSs from observed CDS quotes, then do risk-neutral pricing. The whole procedure could be time-consuming. To provide an alternative way of pricing those structured products, this paper proposes a machine learning method – Gaussian Process Regression.

1.2 Machine Learning for pricing – Gaussian Process Regression

In this paper, an alternative approach to pricing structured products – Gaussian Process Regression(GPR) – is proposed. The basic idea of GPR is to measure the similarity within the feature space spanned by all the inputs from the traditional pricing techniques, then replicate new prices based on the similarity to the prices in training data. A detailed explanation about how this works will be provided in section 2. Detailed discussions about examining several model selection techniques in the Gaussian Process settings including different kernel functions and objective functions for training purposes will be presented in section 3. Section 4 contains two dimension reduction methods and comparison between their results. The thesis will be concluded by section 5 with conclusion, discussion and further study.

Modelling regression problems using Gaussian process is well known in the geostatistics field[4, 5], but the studied has only focused on small input dimensions e.g. 2-dimensional and 3-dimensional input spaces. GPR under machine learning context was first studied by Williams and Rasmussen in [6]. They also described how to optimize parameters in the covariance matrix, which will be discussed in the following section. Other previous works are also focused on lower dimensional input space[7, 8, 9], so it will be of interest to investigate how GPR can be used in replicating synthetic CDO prices with much higher dimensional input space. The data used in numerical implementation is Citigroup proprietary data with input dimension of 1118. The numerical results are all produced from this dataset unless otherwise stated.

2 Gaussian Process

There are broadly two common approaches when it comes to supervised machine learning. One is to assume a specific class of functions one wants the model to learn, then train the model to learn the parameters that describe that specific function. The other approach is to assign a *prior* probability to all the possible functions, then choose the one that maximise the *likelihood* of training data to get the *posterior* distribution of functions. The first approach has a problem that one has to decide the richness of the class of functions that the model is trying to learn for each specific task. But sometimes, the target function that one aims to learn ends up in a totally different class of functions from the model estimation. For instance, the model may instantiate a linear function estimator for a highly non-linear target. In this case, the prediction result when evaluate at new data will inevitably be subprime or sometimes, very poor. The second approach seems to be intractable in the sense that there are just too many functions that one can assign probability to. This is where Gaussian Process can be of great use. In this section, we first define Gaussian processes, then show how they can be used to tackle the regression problem.

2.1 Introduction to Gaussian Process

Readers are assumed to be familiar with the definition and properties of multivariate Gaussian distributions and stochastic processes. One can define a Gaussian process by the following definition.

Definition 2.1. (Gaussian Process) A *Gaussian* process $\{X_i\}_{i\in\mathcal{I}}$ indexed by an index set $\mathcal{I}$ is a family of random variables X_i's, all defined on the same probability space, such that any finite subset $\mathcal{F} \subset \mathcal{I}$, the random vector $X_{\mathcal{F}} := \{X_i\}_{i\in\mathcal{F}}$ has a multivariate Gaussian distribution.[10]

Because of their analytical tractability, it is convenient to model finite collection of real-valued functions using multivariate Gaussian distributions. In practice, one can think of a Gaussian Process as a very long multivariate gaussian vector indexed by some index space (e.g. time, space, hyperspace . . .). The training data will be some dimensions of this vector that one has already observed, and one wants to make prediction of dimensions that hasn't been observed.

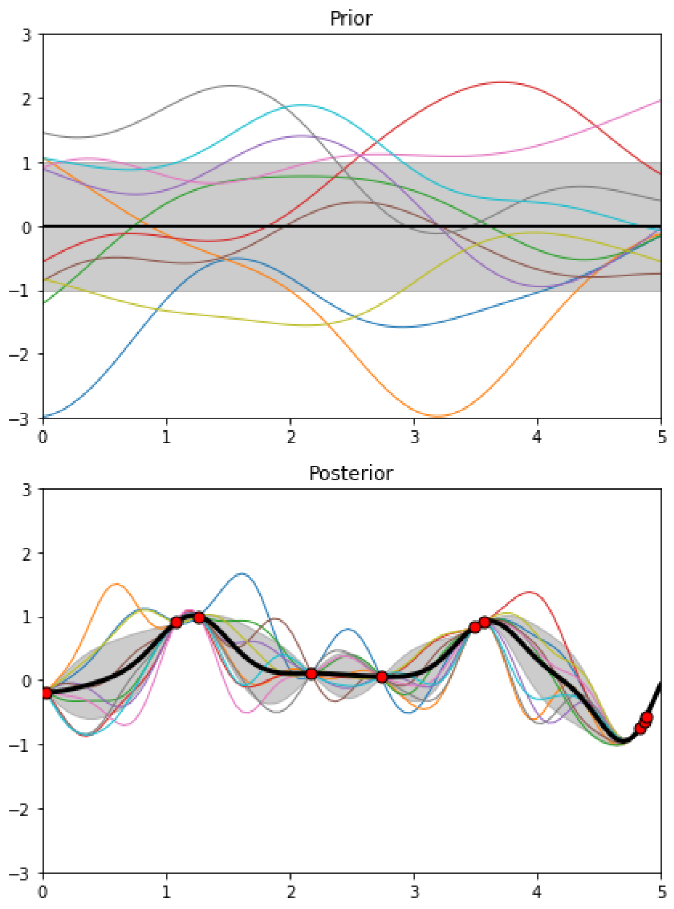

Figure 1: The upper panel is the *prior* of a Gaussian Process with its parameters set at initial value. The bold black line is the mean, which in our example is constantly zero. The colored lines are sample functions drawn from this Gaussian Process. Grey area is one standard deviation away from the mean at each input point. The lower panel is the *posterior*. The dots on the black line are our observations. Colored lines are sample functions drawn from the *posterior* distribution. Grey area is one standard deviation away from the mean. Notice that once we have our observation, we are more certain about the function value around observed points. This is illustrated by the shrinking of the grey area in the lower panel.

Before observing any values from a Gaussian Process, one only has a *prior* distribution over functions specified by that Gaussian Process which may include all the continuous functions. Once some data points have been observed, the possible functions are now reduced to those that go through the observed points. Because of the multivariate gaussian distribution property of any finite subset of a Gaussian Process, prediction on unobserved data can be calculated by conditioning on the observed points. *Prior* combined with observation gives the *posterior* distribution over the function that one wants to model.

When there are more data points coming in, the uncertainty from the posterior distribution will be decreased. Hence, the prediction will become more and more confident when the model has observed more and more data. For traditional parametric methods, the model will tend to overfit the training data when the size of training set grows. However, for Gaussian Processes, since one are not assuming any specific form of the predicted function, one does not have to worry about overfitting of training data. There are still some flexibility left (specified by the variance of the posterior Gaussian distribution), even a lot of training data points have been observed.

2.2 Gaussian Process Regression

So far, we have introduced Gaussian Processes. In order to solve the regression problem, one needs to first define a *prior* distribution of the target functions by a Gaussian process, then apply Bayesian inference[12] to get the *posterior* distribution of our predictions. The Gaussian Process Regression can be interpreted in two ways: 1) Weight-space View, and 2) Function-space View.

2.2.1 Weight-space View

The weight-space view of Gaussian Process starts with the standard linear regression model but from a Bayesian perspective. Recall the Bayesian linear regression model of input $\mathbf{x}$ and targets y is

$$y = \mathbf{x}^T \mathbf{w} + \epsilon, \tag{2.1}$$

where $\mathbf{w}$ is the weight and ϵ is Gaussian noise with distribution $\mathcal{N}(0, \sigma_n^2)$. This formulation gives the *likelihood* of the training data

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{X}^T \mathbf{w}, \sigma_n^2 I). \tag{2.2}$$

In the Bayesian setting, one needs to specify a *prior* over the parameters $\mathbf{w}$ before we do any inference from the training data[13, sec 2.1]. We assume a zero mean Gaussian distribution for $\mathbf{w}$, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma)$. Then by Bayes' rule

$$posterior = \frac{likelihood \times prior}{marginal\ likelihood}, \tag{2.3}$$

we have

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}, \tag{2.4}$$

where *marginal likelihood* is a constant and has the expression of

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})d\mathbf{w}. \tag{2.5}$$

Plug in *likelihood* and *prior* to get

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \bar{\mathbf{w}})^T(\sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \Sigma^T)(\mathbf{w} - \bar{\mathbf{w}})\right), \tag{2.6}$$

where $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \Sigma^{-1})^{-1}\mathbf{X}\mathbf{y}$. Notice that this is the probability density function of the *posterior* of $\mathbf{w}$. We have the *posterior* probability is

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \mathcal{N}(\bar{\mathbf{w}}, \bar{\Sigma}) \tag{2.7}$$

where $\bar{\Sigma} = (\sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \Sigma^{-1})^{-1}$. The prediction on a new data point $\mathbf{x}^*$ is the weighted average from all possible values of parameters by their *posterior* distribution. Thus the predictive distribution for y^* is

$$\begin{aligned} p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) &= \int p(y^*|x^*, \mathbf{w})p(x|\mathbf{W}, \mathbf{y})d\mathbf{w} \\ &= \mathcal{N}\left(\sigma_n^{-2}x^{*T}\bar{\Sigma}\mathbf{X}\mathbf{y}, x^{*T}\bar{\Sigma}x^*\right). \end{aligned} \tag{2.8}$$

The Bayesian linear model may perform poorly on higher dimensional data because of its lack of complexity. One way to tackle this problem is to introduce a kernel function applying to the input data before doing linear regression. This approach will lead to the weight-space interpretation of Gaussian Process.

Apply kernel function $\phi(\mathbf{x})$ to the input, and the model becomes

$$y = \phi(\mathbf{x})^T\mathbf{w} + \epsilon. \tag{2.9}$$

Apply the same inference process as the standard Bayesian linear model above and replace $\mathbf{X}$ by $\Phi(\mathbf{X})$, $\mathbf{x}^*$ by $\phi(\mathbf{x}^*)$in eq.2.8, we get

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(\sigma_n^{-2}\phi(x^*)^T\bar{\Sigma}\Phi\mathbf{y}, \phi(x^*)^T\bar{\Sigma}\phi(x^*)\right). \tag{2.10}$$

Rewrite the above equation to get

$$\begin{aligned} p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(&(\phi^*)^T\Sigma\Phi(K+\sigma_n^2 I)^{-1}\mathbf{y}, \\ &(\phi^*)^T\Sigma\phi^* - (\phi^*)^T\Sigma\Phi(K+\sigma_n^2 I)^{-1}\Phi^T\Sigma\phi^*), \end{aligned} \tag{2.11}$$

where $\phi^* = \phi(\mathbf{x})^*$ and K is defined as $\Phi^T\Sigma\Phi$. Notice that in eq.2.11, one can interpret $(\phi^*)^T\Sigma\Phi$, $(\phi^*)^T\Sigma\phi^*$ and $\Phi^T\Sigma\phi^*$ as inner products with respect to Σ. If we define $\psi(\mathbf{x}) = \Sigma^{1/2}\phi(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x}) \cdot \psi(\mathbf{x}')$, then we have $(\phi^*)^T\Sigma\Phi = k(\mathbf{x}^*, \mathbf{X})$, $(\phi^*)^T\Sigma\phi^* = k(\mathbf{x}^*, \mathbf{x}^*)$ and $\Phi^T\Sigma\phi^* = k(\mathbf{X}, \mathbf{x}^*)$. Then eq. 2.11 becomes

$$\begin{aligned} p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(&k(\mathbf{x}^*, \mathbf{X})(K+\sigma_n^2 I)^{-1}\mathbf{y}, \\ &k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{X})(K+\sigma_n^2 I)^{-1}k(\mathbf{X}, \mathbf{x}^*)), \end{aligned} \tag{2.12}$$

which is the predictive distribution of y^* in Gaussian Process.

2.2.2 Function-space View

Another interpretation of Gaussian Process Regression is through function-space view. If one considers a Gaussian Process as a continuous stochastic process, then it defines a probability distribution for functions [11]. Since a multivariate Gaussian distribution can be completely specified by its mean and covariance, by definition 2.1, then a Gaussian Process can be fully specified by its mean function and covariance function [13]. If one defines the mean function as $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real-valued Gaussian Process $f(\mathbf{x})$ as

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right] \end{aligned} \tag{2.13}$$

and then one can write the Gaussian Process $f(\mathbf{x})$ as

$$f(\mathbf{x}) \ \sim \ \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.14}$$

Suppose we choose a finite subset of random variables $\mathbf{f}$ from the Gaussian Process $f(\mathbf{x})$, with the corresponding index set $\mathcal{X}$, then by definition of Gaussian Process, we know the distribution of $\mathbf{f}$:

$$p(\mathbf{f} \mid \mathcal{X}) \ = \ \mathcal{N}(\mathbf{m}, K), \tag{2.15}$$

where $\mathcal{N}(\mathbf{m}, K)$ denotes a multivariate Gaussian distribution with mean $\mathbf{m}$ and covariance K. In our task, the index set $\mathcal{X}$ can be a subset of $\mathbb{R}^d$, where d is the dimension of our input space. Usually we take the mean function to be a constant function equals to zero for notational simplicity. In practice, one can subtract the mean to achieve this.

If the target function $f(\mathbf{x})$ is estimated as a Gaussian process, then one needs to check if the *consistency* requirement of estimator is fulfilled. One can easily do this by the marginalization property of multivariate Gaussian distributions. The marginalization property tells us that if

$$(\mathbf{f}_1, \mathbf{f}_2) \ \sim \ \mathcal{N}(\mathbf{m}, K),$$

then we also have

$$\begin{aligned} \mathbf{f}_1 \ &\sim \ \mathcal{N}(\mathbf{m}_1, K_{11}) \\ \mathbf{f}_2 \ &\sim \ \mathcal{N}(\mathbf{m}_2, K_{22}). \end{aligned}$$

where K_{11} and K_{22} are sub-matrices of K. The proof is in the following:

Proof. Prove the marginal density $p(\mathbf{f}_1)$ follows Gaussian distribution. First, by definition of marginalization, we have

$$p(\mathbf{f}_1) = \int p(\mathbf{f}_1, \mathbf{f}_2) d\mathbf{f}_2,$$

where

$$p(\mathbf{f}_1, \mathbf{f}_2) = \frac{1}{(2\pi)^{n/2}\sqrt{\det K}} \exp(E)$$

and E is given by

$$\begin{aligned} E = &-\frac{1}{2}\left(\mathbf{f}_2 - (\mathbf{m}_2 - \boldsymbol{\Lambda}_{22}^{-1}\boldsymbol{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right)^T \boldsymbol{\Lambda}_{22} \left(\mathbf{f}_2 - (\mathbf{m}_2 - \boldsymbol{\Lambda}_{22}^{-1}\boldsymbol{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right) \\ &+ \frac{1}{2}\left(\mathbf{f}_1^T\boldsymbol{\Lambda}_{12}\Lambda_{22}^{-1}\Lambda_{21}\mathbf{f}_1 - 2\mathbf{f}_1^T\boldsymbol{\Lambda}_{12}\Lambda_{22}^{-1}\Lambda_{21}\mathbf{m}_1 + \mathbf{m}_1^T\boldsymbol{\Lambda}_{12}\Lambda_{22}^{-1}\Lambda_{21}\mathbf{m}_1\right) \\ &- \frac{1}{2}\left(\mathbf{f}_1^T\Lambda_{11}\mathbf{f}_1 - 2\mathbf{f}_1^T\Lambda_{11}\mathbf{m}_1 + \mathbf{m}_1^T\Lambda_{11}\mathbf{m}_1\right) \\ = &-\frac{1}{2}\left(\mathbf{f}_2 - (\mathbf{m}_2 - \boldsymbol{\Lambda}_{22}^{-1}\boldsymbol{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right)^T \boldsymbol{\Lambda}_{22} \left(\mathbf{f}_2 - (\mathbf{m}_2 - \boldsymbol{\Lambda}_{22}^{-1}\boldsymbol{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right) \\ &- \frac{1}{2}(\mathbf{f}_1 - \mathbf{m}_1)^T \left(\boldsymbol{\Lambda}_{11} - \boldsymbol{\Lambda}_{12}\boldsymbol{\Lambda}_{22}^{-1}\boldsymbol{\Lambda}_{21}\right)(\mathbf{f}_1 - \mathbf{m}_1) \end{aligned}$$

where $\mathbf{\Lambda}$ is the information matrix and

$$\mathbf{\Lambda} = K^{-1} = \begin{pmatrix} \mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{12} \\ \mathbf{\Lambda}_{21} & \mathbf{\Lambda}_{22} \end{pmatrix}.$$

Using the *matrix inversion lemma* (see Appendix), we have

$$K_{11}^{-1} = \mathbf{\Lambda}_{11} - \mathbf{\Lambda}_{12}\mathbf{\Lambda}_{22}^{-1}\mathbf{\Lambda}_{21},$$

combined with the line above to get

$$p(\mathbf{f}_1, \mathbf{f}_2) = \frac{1}{(2\pi)^{n/2}\sqrt{\det K}} \exp(E_1) \exp(E_2),$$

where

$$\begin{aligned} E_1 &= -\frac{1}{2}\left(\mathbf{f}_2 - (\mathbf{m}_2 - \mathbf{\Lambda}_{22}^{-1}\mathbf{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right)^T \mathbf{\Lambda}_{22}\left(\mathbf{f}_2 - (\mathbf{m}_2 - \mathbf{\Lambda}_{22}^{-1}\mathbf{\Lambda}_{21}(\mathbf{f}_1 - \mathbf{m}_1))\right) \\ E_2 &= -\frac{1}{2}(\mathbf{f}_1 - \mathbf{m}_1)^T \left(\mathbf{\Lambda}_{11} - \mathbf{\Lambda}_{12}\mathbf{\Lambda}_{22}^{-1}\mathbf{\Lambda}_{21}\right)(\mathbf{f}_1 - \mathbf{m}_1). \end{aligned}$$

Since E_2 is independent of $\mathbf{f}_1$, we have

$$p(\mathbf{f}_1) = \frac{1}{(2\pi)^{n/2}\sqrt{\det K}} \int \exp(E_1) d\mathbf{f}_2 \exp(E_2).$$

Integral of a probability density function is one, we get

$$\int \exp(E_1) d\mathbf{f}_2 = (2\pi)^{n_2/2}\sqrt{\det \mathbf{\Lambda}_{22}^{-1}}$$

plug this into the line above to get

$$p(\mathbf{f}_1) = \frac{\sqrt{\det \mathbf{\Lambda}_{22}^{-1}}}{(2\pi)^{n_1/2}\sqrt{\det K}} \exp\left(-\frac{1}{2}(\mathbf{f}_1 - \mathbf{m}_1)^T K_{11}^{-1}(\mathbf{f}_1 - \mathbf{m}_1)\right).$$

Again, by the *matrix inversion lemma*, we have

$$\begin{aligned} \det K &= \det K_{11} \det(K_{22} - K_{21}K_{11}^{-1}K_{12}) \\ \mathbf{\Lambda}_{22}^{-1} &= K_{22} - K_{21}K_{11}^{-1}K_{12}. \end{aligned}$$

Plug those results into the previous line, we get

$$p(\mathbf{f}_1) = \frac{1}{(2\pi)^{n_1/2}\sqrt{\det K_{11}}} \exp\left(-\frac{1}{2}(\mathbf{f}_1 - \mathbf{m}_1)^T K_{11}^{-1}(\mathbf{f}_1 - \mathbf{m}_1)\right).$$

which proves that $\mathbf{f}_1 \sim \mathcal{N}(\mathbf{m}_1, K_{11})$. $\square$

Through *consistency* of Gaussian process, one knows if $(\mathbf{f}_1, \mathbf{f}_2) \sim \mathcal{N}(\mathbf{m}, K)$, then $\mathbf{f}_1 \sim \mathcal{N}(\mathbf{m}_1, K_{11})$ and $\mathbf{f}_2 \sim \mathcal{N}(\mathbf{m}_2, K_{22})$. Now one can use Gaussian process to model target function that one wants to estimate.

Firstly, model the target latent function $\mathbf{f}$ by a zero mean Gaussian Process, then $\mathbf{f}$ has prior distribution:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K), \tag{2.16}$$

where K is the covariance matrix.

Assume the special case where the target functions are noise-free. Suppose we have observed data $\{(\mathbf{x}_i, f_i) | i = 1, ..., n\}$, we want to infer the value of our target function at input points $\{\mathbf{x}_j | j = 1, ..., n_*\}$. We call $\mathbf{f}$ as training outputs and $\mathbf{f}_*$ as test outputs. From the prior distribution, we know $\mathbf{f}$ and $\mathbf{f}_*$ have joint distribution

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right). \tag{2.17}$$

$K(\mathbf{X}, \mathbf{X})$ denotes the covariance matrix of n training data points, $K(\mathbf{X}_*, \mathbf{X}_*)$ denotes the covariance of n_* test data points and $K(\mathbf{X}_*, \mathbf{X})$ denotes the cross-covariance of training and test data points. To get the distribution of our prediction on the test data set, we just condition the joint Gaussian prior distribution on the training data set. Then we get

$$\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{m}_*, V_*), \tag{2.18}$$

where

$$\begin{aligned} \mathbf{m}_* &= K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{f}, \\ V_* &= K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*). \end{aligned} \tag{2.19}$$

Proof. For notational simplicity, let's denote $K(\mathbf{X}, \mathbf{X})$ as K, $K(\mathbf{X}, \mathbf{X}_*)$ and $K(\mathbf{X}_*, \mathbf{X})$ as K_* and $K(\mathbf{X}_*, \mathbf{X}_*)$ as K_{**}.

The conditional probability density function of $\mathbf{f}_*$ given $\mathbf{f}$ is

$$p(\mathbf{f}_* | \mathbf{f}) = \frac{p(\mathbf{f}, \mathbf{f}_*)}{\int p(\mathbf{f}, \mathbf{f}_*) d\mathbf{f}}, \tag{2.20}$$

where $\int p(\mathbf{f}, \mathbf{f}_*)d\mathbf{f}$ is a normalization constant. Denote the normalization constant as E, we have

$$p(\mathbf{f}_*|\mathbf{f}) = \frac{1}{E}\exp\left(-\frac{1}{2}\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}^T \begin{bmatrix}K & K_*\\ K_* & K_{**}\end{bmatrix}^{-1} \begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}\right). \tag{2.21}$$

Denote

$$\begin{bmatrix}K & K_*\\ K_* & K_{**}\end{bmatrix}^{-1} = \begin{bmatrix}V & V_*\\ V_* & V_{**}\end{bmatrix}, \tag{2.22}$$

then we get

$$\begin{aligned}p(\mathbf{f}_*|\mathbf{f}) &= \frac{1}{E}\exp\left(-\frac{1}{2}\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}^T \begin{bmatrix}V & V_*\\ V_* & V_{**}\end{bmatrix} \begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}\right)\\ &= \frac{1}{E}\exp\left(-\frac{1}{2}[\mathbf{f}^TV\mathbf{f} + \mathbf{f}^TV_*\mathbf{f}_* + \mathbf{f}_*^TV_*\mathbf{f} + \mathbf{f}_*^TV_{**}\mathbf{f}_*]\right).\end{aligned} \tag{2.23}$$

Complete the square to get

$$\begin{aligned}p(\mathbf{f}_*|\mathbf{f}) = \frac{1}{E}\exp(-[\frac{1}{2}(\mathbf{f}_* + V_{**}^{-1}V_*\mathbf{f})^TV_{**}(\mathbf{f}_* + V_{**}^{-1}V_*\mathbf{f})\\ \frac{1}{2}\mathbf{f}^TV\mathbf{f} - \frac{1}{2}\mathbf{f}^TV_*V_{**}^{-1}V_*\mathbf{f}]).\end{aligned} \tag{2.24}$$

Take out the terms that are independent of $\mathbf{f}_*$ from the exponential and put them into the normalization constant to get updated constant E', then we get

$$p(\mathbf{f}_*|\mathbf{f}) = \frac{1}{E'}\exp\left(-\frac{1}{2}(\mathbf{f}_* + V_{**}^{-1}V_*\mathbf{f})^TV_{**}(\mathbf{f}_* + +V_{**}^{-1}V_*\mathbf{f})\right). \tag{2.25}$$

Recognise this is in the form of Gaussian probability density function with

$$\begin{aligned}\mu &= -V_{**}^{-1}V_*\mathbf{f},\\ \sigma^2 &= V_{**}^{-1}.\end{aligned} \tag{2.26}$$

By *matrix inverse lemma*(see Appendix), we have

$$\begin{bmatrix}K & K_*\\ K_* & K_{**}\end{bmatrix} = \begin{bmatrix}(V - V_*V_{**}^{-1}V_*)^{-1} & -(V - V_*V_{**}^{-1}V_*)^{-1}V_*V_{**}^{-1}\\ -V_{**}^{-1}V_*(V - V_*V_{**}^{-1}V_*)^{-1} & (V_{**} - V_*V^{-1}V_*)^{-1}\end{bmatrix}. \tag{2.27}$$

Combining the previous line to get

$$\begin{aligned}\mu &= -V_{**}^{-1}V_*\mathbf{f} = K_*K^{-1}\mathbf{f},\\ \sigma^2 &= V_{**}^{-1} = K_{**} - K_*K^{-1}K_*,\end{aligned} \tag{2.28}$$

which concludes the proof for eq.2.18 and 2.19. □

The prediction on the test set is a *posterior* Gaussian distribution with mean $\mathbf{m}_*$ and covariance $\mathbf{V}_*$. Notice that in the prediction, rather than giving specific parametric relationship between the test input $\mathbf{X}_*$ and the prediction value, the model just gives a Gaussian distribution inferred from the prior distribution and observation. If one wants to get an exact value of $\mathbf{f}_*$ for prediction, one can sample from the posterior distribution or simply take the mean $\mathbf{m}_*$ as our prediction. For more on this, please see section 2.3. One can also get the variance v_* of each individual element in vector $\mathbf{f}_*$ from the covariance matrix $\mathbf{V}_*$. The smaller the variance is, the more confident we are about the prediction.

In most real-life application, it is more realistic to incorporate noise in the target values. We can denote the noisy target values as $\mathbf{y}$ in training set. Then write the noisy value as $\mathbf{y} = f(\mathbf{x}) + \epsilon$. Assuming identical, independent Gaussian noise ϵ with mean 0 and variance σ_n^2, then the covariance on the noisy observation is

$$\text{cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij}$$

where δ_{ij} is a Kronecker delta which is 1 is $i = j$ and 0 otherwise. The above relationship is vector form is

$$\text{cov}(\mathbf{y}) = K(X, X) + \sigma_n^2 \mathbf{I}, \tag{2.29}$$

where the identity matrix $\mathbf{I}$ comes from the independence assumption of the noise. Then we can write the joint prior distribution of noisy training data and test data as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 \mathbf{I} & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right). \tag{2.30}$$

Then by the same derivation to eq. 2.18 and 2.19, we get the prediction for test data. The posterior distribution is

$$\mathbf{f}_* | X_*, X, \mathbf{y} \sim \mathcal{N}(\widehat{\mathbf{m}_*}, \widehat{\mathbf{V}_*}), \tag{2.31}$$

where

$$\begin{aligned} \widehat{\mathbf{m}_*} &= K(X_*, X)[K(X, X) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{y}, \\ \widehat{\mathbf{V}_*} &= K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 \mathbf{I}]^{-1} K(X, X_*). \end{aligned} \tag{2.32}$$

This is equivalent to the predictive distribution from weight-space view in eq.2.12.

Notice that though we assume the mean function as constantly zero in the prior distribution, we get non-zero mean function in the posterior distribution for the test data set. This can be interpreted as after observing the training set, the model has more information about the distribution of the test set. Also, different from a parametric model, the Gaussian Process Regression model needs to store all the training data for prediction. For a parametric model, the goal is to find the predefined parametric relationship between features and targets, once done that, the model doesn't need the training data for prediction. The features, $\mathbf{x}_i^*|i = 1, ..., n^*$, will give all the information we need for prediction $\mathbf{y}_i^*$'s. However, in Gaussian Process Regression model, the prediction eq. 2.31 and eq. 2.32 need the information from training set to get the posterior distribution of the test set.

2.3 Predicition from Gaussian Process Regression

Form eq. 2.31, we know that the posterior distribution of test data is Gaussian with mean and variance given in eq.2.32. But if one wants an explicit prediction for the target function rather than a distribution, what is the value then? One intuitive answer would be the mean of posterior distribution. In practice, this prediction is appropriate in most real-life tasks. Rasmussen and Williams provides in-depth explanation in [13, sec 2.4], we present a summary of the rationale in the following.

To find a optimal prediction value, we need a way to measure our performance. Let's define a *loss function*, $\mathcal{L}(\hat{y}, y)$, which specifies the loss between the prediction, $\hat{y}$ and the true value, y. The loss function can be *mean squared error* or *mean absolute error* which are both symmetric loss functions.

One wants to give a prediction value that produces the minimum loss possible. But how do we achieve that when we don't know the true value? According to [13, sec 2.4], we can minimize the expected loss, by averaging the loss with respect to the posterior distribution, which is

$$\tilde{R}_{\mathcal{L}}(\hat{y}|\mathbf{x}_*) = \int \mathcal{L}(y_*, \hat{y}) p(y_*|\mathbf{x}_*) dy_*. \tag{2.33}$$

Our best prediction is the one that minimizes this loss, i.e.

$$y_{optimal}|\mathbf{x}_* = \underset{\hat{y}}{\operatorname{argmin}} \tilde{R}_{\mathcal{L}}(\hat{y}|\mathbf{x}_*). \tag{2.34}$$

For *mean squared error* loss function, the minimum occurs at the mean of y_*. For *mean* *absolute error* loss function, the minimum occurs at the median of y_*. In our case, the distribution of y_* is Gaussian, so the mean and median coincide. Furthermore, for other symmetric loss function and symmetric posterior distribution, the optimal prediction occurs at the mean of the distribution. For asymmetric loss functions, the optimal prediction can be computed from eq. 2.33 and eq. 2.3. For more detail, see [14].

3 Model Selection

When modelling the target function as a zero mean Gaussian process, i.e.

$$f(\mathbf{x}) \;\sim\; \mathcal{GP}\left(\mathbf{0}, k(\mathbf{x},\mathbf{x}')\right).$$

the main task will be to find a good configuration of covariance matrix, generated by kernel function k and a set of *hyperparameters* that gives the best performance. The model selection is a combination of choosing a good kernel function and optimizing *hyperparameters* of the kernel function. In this section, we will discuss several aspects of model selection, namely marginal likelihood, covariance matrix, kernel functions, *hyperparameters* optimization.

3.1 Marginal Likelihood of the training data

From eq. 2.16, the noise-free *prior* follows multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $K(X, X)$. By Bayes' theorrm, the marginal likelihood of the noisy training data is the integral of the likelihood times prior, which is

$$p(\mathbf{y}|X) \;=\; \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|X)d\mathbf{f}. \tag{3.1}$$

The term *marginal* likelihood means the marginalization over the function values $\mathbf{f}$. Assume identical, independent Gaussian noise, then we have $\mathbf{y} = f(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma_n^2)$. Let $K_y = K + \sigma_n^2 I$, then by eq. 2.29. one gets

$$\mathbf{y} \;\sim\; \mathcal{N}\left(\mathbf{0}, K_y\right). \tag{3.2}$$

Since the likelihood of a Gaussian distribution involves exponential term, one can take the logarithm of it. The log likelihood of $\mathbf{y}$ given $\mathbf{X}$ and θ is

$$\log p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi, \tag{3.3}$$

where θ is parameters in kernel function k.

3.2 Covariance Matrix and Kernel functions

A kernel function k (also called a covariance function) governs the behaviour and property of a zero-mean Gaussian process. Since for any element K_{ij} in the covariance matrix K, we know $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, the relationship of covariance matrix K and kernel function k is

$$K = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{3.4}$$

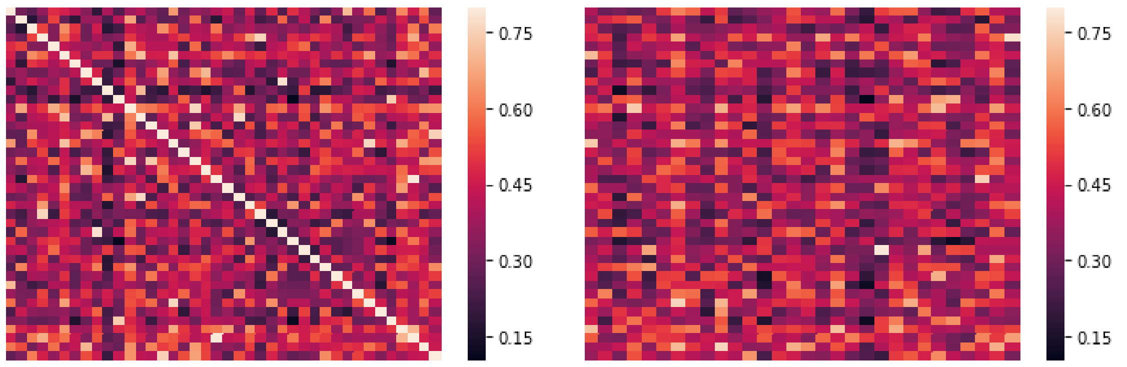

Figure 2: Heat-map of Covariance matrices. The left panel is the covarianve matrix K of the training data $\mathbf{X}$. The diagonal component K_{ii} is the variance of $\mathbf{X}_i$, the off-diagonal component K_{ij} is the covariance between $\mathbf{X}_i$ and $\mathbf{X}_j$. The right panel is the cross-covariance K^* of $\mathbf{X}$ and testing data $\mathbf{X}^*$. K^*_{ij} is the covariance between $\mathbf{X}_i$ and $\mathbf{X}^*_j$.

A kernel function in some sense measures the similarity between two data points in the input space. The basic assumption in GPR is that two points that are close in the input space should be more likely to have similar target values.

Since a covariance matrix is always positive-semidefinite[1] and symmetric, for a function to be qualified as a kernel function, it must be positive-semidefinite[2] and symmetric in the sense that $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$[13, sec 4.1].

3.2.1 Several Common Kernel functions

To understand what kind of function structure that Gaussian process can express, one has to first understand properties of embedded kernel function of the GP. Kernel function map $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ into $\mathbb{R}$. The similarity of $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$ is measured by the result from the kernel function.

A stationary kernel function is a function whose value only depends on the difference between $\mathbf{x}$ and $\mathbf{x}'$. One can write k to take only single argument $\mathbf{r} = \mathbf{x} - \mathbf{x}'$.

Radial Basis Function Kernel

The mostly commonly used stationary kernel function is the *Radial Basis Function*(RBF) kernel, which has the form

$$k_{RBF} = \sigma^2 \exp\left(-\frac{\mathbf{r}^2}{2l^2}\right), \tag{3.5}$$

where the parameter l is the *characteristic length-scale* and σ controls the magnitude of function value. The RBF kernel function is also known as the *quared exponential*(SE) function. This kernel function has derivatives at all orders, which means Gaussian process with this kernel will be very smooth. For detailed explanation, please see [13, sec 4.2]. Sometimes, target functions in real wrold application do not have this smoothness, so another family of stationary kernel functions, namely the *Matérn class* kernel functions are introduced by Stein in [15].

[1]A symmetric $n \times n$ real matrix $\mathbf{M}$ is said to be positive-semidefinite(PSD) if $\mathbf{v}^T\mathbf{M}\mathbf{v}$ is non-negative for all $\mathbf{v}$ in $\mathbb{R}^n$.

[2]A *kernel* is said to be positive-semidefinite if

$$\int k(\mathbf{x}, \mathbf{x}')f(\mathbf{x})f(\mathbf{x}')d\mu(\mathbf{x})d\mu(\mathbf{x}') \geq 0,$$

for all $f \in L_2(\mathcal{X}, \mu)$, $\mathcal{X}$ is the input space and μ is a measure.

Matérn Class of Kernel Functions

This class of kernel functions was named after the work of Matérn. The general form of *Matern class* kernel functions is given by

$$k_{Matern}(\mathbf{r}) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\mathbf{r}}{l} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}\mathbf{r}}{l} \right), \tag{3.6}$$

where ν and l are positive parameters and K_ν is a second kind modified Bessel function[16, sec 9.6]. When $\nu \to \infty$, we obtain the SE kernel function. For smaller ν, the resulting Gaussian process has a rougher path than those from a SE kernel. The most commonly exploited cases in machine learning is when $\nu = 3/2$ and $5/2$ [13, sec 4.2]. These two kernel functions are of the forms

$$\begin{aligned} k_{Matern32}(\mathbf{r}) &= \left(1 + \frac{\sqrt{3}\mathbf{r}}{l}\right) \exp\left(-\frac{\sqrt{3}\mathbf{r}}{l}\right), \\ k_{Matern52}(\mathbf{r}) &= \left(1 + \frac{\sqrt{5}\mathbf{r}}{l} + \frac{5\mathbf{r}^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}\mathbf{r}}{l}\right). \end{aligned} \tag{3.7}$$

For $\nu \geq 7/2$, Rasmussen and Williams have proved the resulting Gaussian process is very smooth in [13, sec 4.2].

Stationary kernels will give the same value as long as the difference between $\mathbf{x}$ and $\mathbf{x}'$ is the same. Hence, stationary kernels are translation invariant. When one wants to incorporate effects of translation in feature space, non-stationary kernel functions should be introduced.

Linear Kernel Functions

A linear kernel function has the form

$$k_{Linear} = \frac{\mathbf{x} \cdot \mathbf{x}'}{l}, \tag{3.8}$$

where l is the *characteristic length-scale. Priors* drawn form a Gaussian process governed by a Linear kernel will be linear functions, see figure 3. The expressive ability of Linear kernel alone is not very interesting, but combined with other non-linear kernel. the resulting GP can capture global trend as well as local variation of the target function.

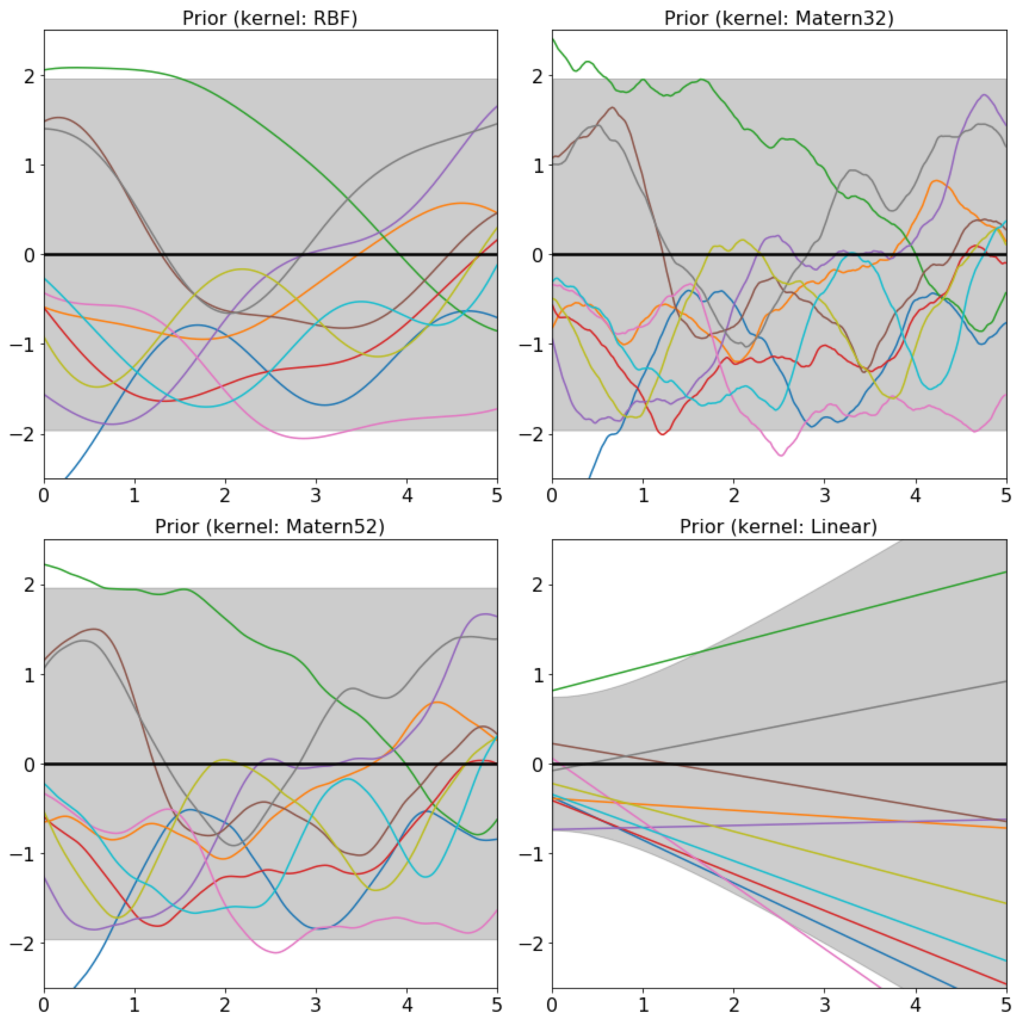

Figure 3: GP Priors with different kernels. For staiontionary kernels e.g. RBF, Matern32 and Matern53, variance at each point(covariance with itself) is constant for fixed parameters. For non-stationary kernel, e.g. Linear kernel, variance increases as data points move away from the origin. We can also see that sample functions drawn from *prior* distribution of different kernels show different level of smoothness. Sample functions draw from Linear kernel are linear functions.

GP *priors* and *posteriors* with different kernel functions are shown in Figure 3 and 4.

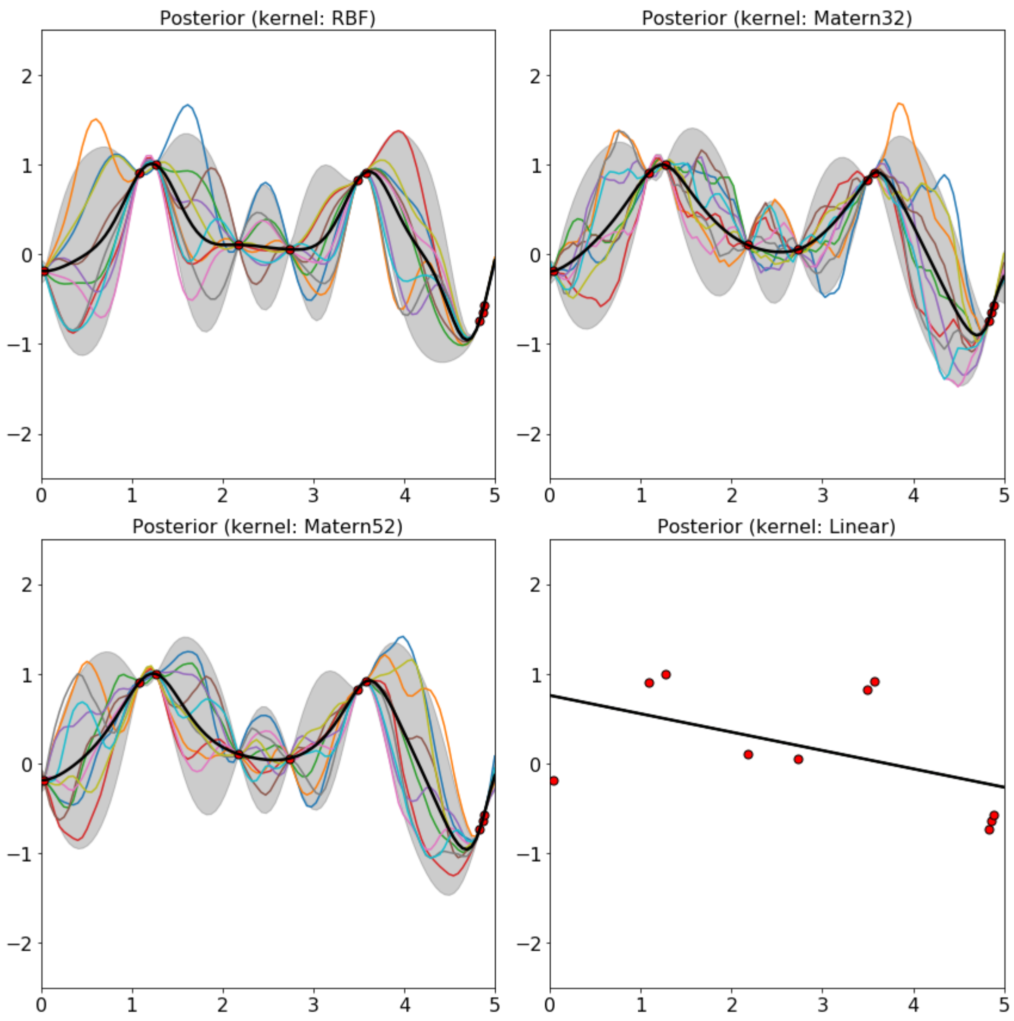

Figure 4: GP Posteriors with different kernels. The red dots are the training points. The black line is the mean function, the grey area is 95% confidence intervel.

Combining Kernel Functions

For most real-life tasks e.g. estimating the price of derivatives, using vanilla kernel functions described above may not generate ideal result. We can construct new kernel functions from these vanilla kernel functions by addition, multiplication, convolution and other methods. Adding two kernel functions is equivalently modelling the resulting Gaussian process by the sum of two independent Gaussian processes[13, sec 4.2].

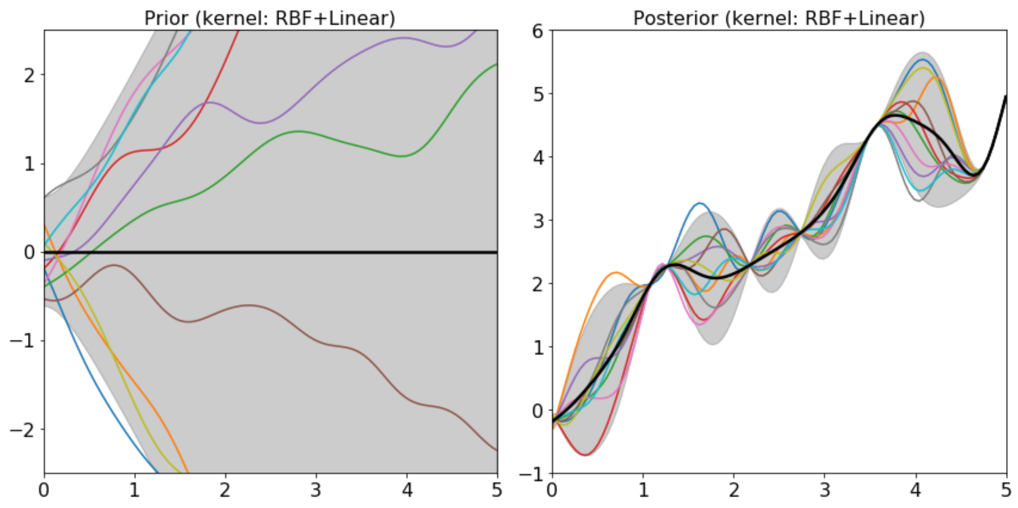

Figure 5: GP *prior* and *posterior* with RBF+Linear kernel. The mean function from the *postetior* distribution has a upwards trend and local variation.

3.2.2 Optimizing Kernel Parameters

Each kernel function has a set of parameters which determines properties of the kernel function. In Gaussian process regression, since these parameters specify distributions of the parameters of target function, so we call them *hyperparameters* of the model. In the training process of our model, we want to find a optimal set of *hyperparameters* in the sense that the log likelihood of our training data define in eq. 3.3 is maximised. We will discuss in detail about training methods in the following subsection.

3.2.3 Numerical Results of Different Kernels

Table 1 shows prediction performance of GPRs governed by different kernel functions namely RBF, Matern32, Matern52, RBF+Linear, Matern32+Linear and Matern52+Linear. The target function is the price of synthetic CDO. We divide the training set into 4 batches of 2048 points in each batch for computational efficiency. The testing set size is 998. The feature space $\mathcal{X}$ is $[0,1]^{1118}$. We measure the absolute value of difference between prediction and target value from 10 to 10000. The results are shown as the percentage of the entire testing set. GPR with Matérn32 kernel function has the most prediction within absolute difference of 10 out of all other kernel functions. This can be explained by the

	Training data absolute difference				Testing data absolute difference			
	10	100	1000	10000	10	100	1000	10000
RBF	59.31	78.02	95.31	99.28	27.66	36.97	50.50	75.85
Matern32	**60.77**	78.89	94.43	99.46	**30.76**	40.58	**58.02**	**80.66**
Matern52	59.85	78.65	95.00	99.38	30.26	38.78	53.31	77.56
RBF+Linear	50.42	79.90	**97.91**	**100.00**	28.36	39.78	53.81	75.55
Matern32+Linear	48.88	77.16	96.48	99.96	27.15	39.68	53.51	75.45
Matern52+Linear	54.24	**80.21**	96.72	99.77	29.36	**41.28**	54.61	77.15

Table 1: Results for kernels with single value length-scale.

smoothness by the target function that we are trying to estimate. Functions modelled GPR with Matérn32 kernel are rougher than those by GPR with RBF and Matérn52 function. When added Liner kernel function, the results do improve a little within absolute difference of 1000 and 10000 on training set, this means our model can explain more of the global trend of the target function compared to the one without Linear kernel. However, the performance get worse within the absolute difference of 10. This is maybe because the target function does not have a clear linear relationship with any of the dimensions from feature space, when fitted by a Linear kernel, the model is not able to give an accurate prediction.

3.2.4 Automatic Relevance Determination

In the previous discussion, we used single scalar value for the *characteristic length-scale l* in hyperparameter θ for kernels functions. In practice, for multi-dimensional feature space, a universal *characteristic length-scale* for all dimensions may not perform well. Instead, We can assign a *characteristic length-scale* for each dimension. For instance, a RBF and

Matérn kernel function in this form are

$$\begin{aligned}
k_{RBF\_ARD}(\mathbf{r}) &= \sigma^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{\mathbf{r}_i^2}{l_i^2}\right), \\
k_{Matern32\_ARD}(\mathbf{r}) &= \left(1+\sqrt{3}\sum_{i=1}^{d}\frac{|r_i|}{l_i}\right)\exp\left(-\sqrt{3}\sum_{i=1}^{d}\frac{|r_i|}{l_i}\right), \\
k_{Matern52\_ARD}(\mathbf{r}) &= \left(1+\sqrt{5}\sum_{i=1}^{d}\frac{|r_i|}{l_i}+\frac{5}{3}\sum_{i=1}^{d}\frac{\mathbf{r}_i^2}{l_i^2}\right)\exp\left(-\sqrt{5}\sum_{i=1}^{d}\frac{|r_i|}{l_i}\right).
\end{aligned} \tag{3.9}$$

where $\mathbf{r} \in \mathbb{R}^d$, d is the number of dimensions in feature space. ARD stands for automatic relevance determination[22, Neal]. From eq.3.9, we can see that the inverse of l_i will determine how sensitive the covariance is to the change in $\mathbf{r}_i$. For large value of l_i, the inverse is close to zero which will make the value of covariance all-most invariant to the change in $\mathbf{r}_i$. This effect will determine the relevance of i^{th} dimension of input to the covariance given enough training data, hence, the name – automatic relevance determination.

	Training data absolute difference				Testing data absolute difference			
	10	100	1000	10000	10	100	1000	10000
RBF	47.94	74.13	94.06	99.96	27.45	41.58	65.83	88.38
Matern32	**58.51**	**83.42**	**98.35**	99.98	**30.76**	44.59	68.34	88.08
Matern52	56.10	81.23	97.52	99.98	29.56	**45.29**	68.44	87.88
RBF+Linear	31.30	56.18	85.61	99.63	23.75	39.48	65.23	89.78
Matern32+Linear	45.19	75.07	96.25	**100.00**	28.36	41.98	**68.74**	**90.28**
Matern52+Linear	50.92	80.51	97.86	**100.00**	29.46	42.79	64.73	86.97

Table 2: Results for ARD kernels

Table 2 shows prediction performance of GPRs governed by different kernel functions with ARD *characteristic length-scale*. All models are trained with 4 batches training data with each batch of 2048 points. We iterate the optimization 400 times through the entire training set. The actual results on the training set is not improved from those by non-ARD kernels shown in table 1, this may be caused by the relatively small amount of training data points(2048 each batch) compared to the number of parameters that we have to optimize(more than 1000). However, from table 2 and figure 6, we can see the results on absolute difference from 100 to 10000 of testing data are improved for all kernels when using ARD. This is because ARD kernels can give a more accurate measure of similarity between training and testing points than those without ARD.

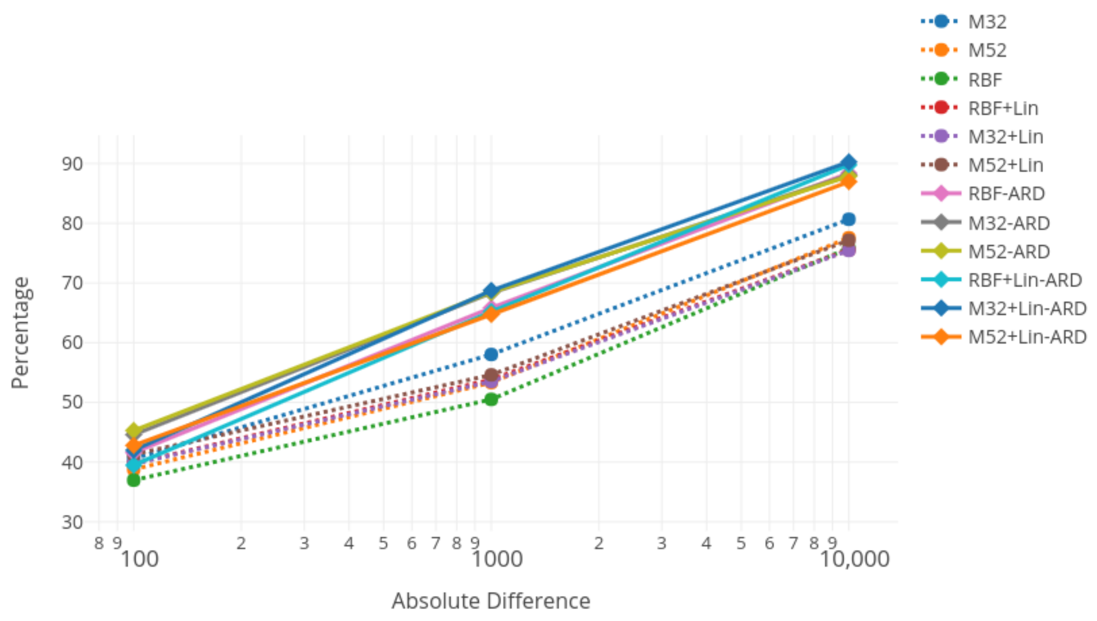

Figure 6: Absolute difference on testing set with ARD kernels. Kernels with ARD consistently give better results than kernels without ARD.

3.3 Training Methods

Once we have chosen our kernel function, the remaining work of model selection is to optimise the *hyperparameters* in the chosen kernel function. In this subsection, we will suppose three objective functions for training and explain how to use gradient decent based algorithm for optimisation.

3.3.1 Maximum Likelihood(ML)

Recall from eq. 3.3, we define the marginal likelihood of training data

$$\log p(\mathbf{y}|\mathbf{X},\theta) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi, \tag{3.10}$$

where $K_y = K_\mathbf{f} + \sigma^2 I$ is the covariance matrix for the noisy targets value $\mathbf{y}$. K_f is the covariance matrix of latent function $\mathbf{f}$ and σ^2 is the variane of i.i.d. Gaussian noise. We can interpret the first term in the marginal likelihood $-\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y}$ as the data-fit since it is the only term involves target values. The second term $-\frac{1}{2}\log|K_y|$ is the complexity penalty since it only depends on the covariance matrix. The third term $-\frac{n}{2}\log 2\pi$ is the normalization constant[13, Rasmussen and Williams, sec 5.4]. As *length-scale* increases, K_y^{-1} decreases and $\log|K_y|$ increases. So we want to find a set θ such that

$$\hat{\theta} = \underset{\theta\in\boldsymbol{\Theta}}{\operatorname{argmax}} \log p(\mathbf{y}|\mathbf{X},\theta). \tag{3.11}$$

In practice, in order to find $\hat{\theta}$ that maximises the log marginal likelihood, we first take partial derivatives of the the log marginal likelihood with respect to each element of the hyperparameter vector θ. Here, we use vector θ to denote all the hyperparameters in our chosen kernel function. Let θ_i be i^{th} element in θ, then the partial derivative of log marginal likelihood w.r.t. θ_i is

$$\begin{aligned}
\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\mathbf{X},\theta) &= -\frac{1}{2}\mathbf{y}^T\frac{\partial K^{-1}}{\partial\theta_i}\mathbf{y} - \frac{1}{2}\frac{\partial\log|K|}{\partial\theta_i} \\
&= -\frac{1}{2}\mathbf{y}^T\left(-K^{-1}\frac{\partial K}{\partial\theta_i}K^{-1}\right)\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_i}\right) \\
&= \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial\theta_i}K^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_i}\right) \\
&= \frac{1}{2}\mathrm{tr}\left((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_i}\right), \text{ where } \alpha = K^{-1}\mathbf{y}.
\end{aligned} \tag{3.12}$$

Notice that $\frac{\partial K}{\partial\theta_i}$ denotes the matrix of element-wise derivatives. Then in each iteration, we update our the parameters θ_i^n by

$$\theta_i^n = \theta_i^{n-1} - \eta\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\mathbf{X},\theta) \tag{3.13}$$

where η is a predefined learning rate.

The main computational complexity of computing this partial derivative lies in the computing of K^{-1}, which is of order $\mathcal{O}(n^3)$ for a n dimensional matrix K. After we have computed partial derivatives for every hyperparameters, we update each of them using a gradient decent based optimizer. Many well known packages in major programming languages have those optimizer built-in. We use Python for our implementation. One can find several more efficient optimizers such as RMSPropOptimizer and AdamOptimizer in Python. In practice, we will set our objective function to be the negative log likelihood, then we use optimizer to minimise our objective function w.r.t. hyperparameters. For those optimizers mentioned above, We can also use adaptive learning rate during optimising operation for better performance when approaching to the minimal point. By adaptive learning rate, we mean learning rate decays by a certain method. For instance, the time-based decay is specified by $\eta = \eta_0/(1 + k * t)$, where η_0 is the initial learning rate, k is a hyperpamater that governs the decay speed and t is the iterations that have passed[17, Bengio].

3.3.2 LOO-CV Based Objective Functions

We can also define our objective function as the leave-one-out cross validation (LOO-CV) based predictive measure. Cross Validation (CV) based predictive measure has been successfully used in many model selection tasks in machine learning by Cawley and Talbot in [18] and Sundararajan and Keerthi in [19].

Geisser's surrogate Predictive Probability(GPP)

Predictive Sample Reuse(PSR) methods was introduced by Geisser[20] to be applied for both model selection and parameters optimisation problem. The basic idea is to define a partition of training data $P(N, n, \Gamma)$ such that such that P^i_{N-n} is the predictive probability of i^{th} subset of size n in the partition. Leave-one-out cross validation is the special case when we take the size of element in the partition to be $n = 1$.

The predictive log likelihood when leaving out training case i is

$$\log p(y_i|\mathbf{X}, \mathbf{y}_{-i}, \theta) = -\frac{1}{2}\log \sigma_i^2 - \frac{(y_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2}\log 2\pi, \tag{3.14}$$

where $\mathbf{y}_{-i}$ means all targets value except the i^{th} value, μ_i and σ_i are calculated according to eq. 2.32. The training data are taken to be $(\mathbf{X}_{-i}, \mathbf{y}_{-i})$. We take our objective function to be the average negative predictive log likelihood, which is given by

$$G(\mathbf{X}, \mathbf{y}, \theta) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y_i|\mathbf{X}, \mathbf{y}_{-i}, \theta) \tag{3.15}$$

This objective function is known as Geisser's surrogate Predictive Probability(GPP), first supposed by Geisser and Eddy in [21]. It is also called *pseudo*-likelihood by Rasmussen and Williams in [13, sec. 5.4].

The main computational cost in calculating GPP is in the calculation of the predictive mean μ_i and predictive variance σ_i^2, which are dominated by the inversion of the matrix K in eq. 2.32. The expression for μ_i and σ_i^2 was calculated in [19], and they are

$$\begin{aligned} \mu_i &= y_i - [K^{-1}\mathbf{y}]_i/[K^{-1}]_{ii}, \\ \sigma_i^2 &= 1/[K^{-1}]_{ii}. \end{aligned} \tag{3.16}$$

Plug in these expressions into eq. 3.14 and eq. 3.15 to get the objective function. The next step is to calculate the partial derivatives of the objective function with respect to hyperparameters. We first use the expression in eq. 3.16 to calculate the partial derivatives of the predictive mean and variance

$$\begin{aligned} \frac{\partial \mu_i}{\partial \theta_j} &= \frac{[Z_j\alpha]_i}{[K^{-1}]_{ii}} - \frac{\alpha_i[Z_jK^{-1}]_{ii}}{[K^{-1}]_{ii}^2}, \\ \frac{\partial \sigma_i^2}{\partial \theta_j} &= \frac{[Z_jK^{-1}]_{ii}}{[K^{-1}]_{ii}^2}, \end{aligned} \tag{3.17}$$

where $\alpha = K^{-1}\mathbf{y}$ and $Z_j = K^{-1}\frac{\partial K}{\partial \theta_j}$. Using chain rule and eq. 3.14 to calculate the partial derivatives of eq.3.15

$$\begin{aligned} \frac{\partial G(\mathbf{X},\mathbf{y},\theta)}{\partial \theta_j} &= -\frac{1}{n}\sum_{i=1}^{n} \frac{\partial \log p(y_i|\mathbf{X},\mathbf{y}_{-i},\theta)}{\partial \mu_i}\frac{\partial \mu_i}{\partial \theta_j} + \frac{\partial \log p(y_i|\mathbf{X},\mathbf{y}_{-i},\theta)}{\partial \sigma_i^2}\frac{\partial \sigma_i^2}{\partial \theta_j} \\ &= -\frac{1}{n}\sum_{i=1}^{n}\left(\alpha_i[Z_j\alpha]_i - \frac{1}{2}\left(1+\frac{\alpha_i^2}{[K^{-1}]_{ii}}\right)[Z_jK^{-1}]_{ii}\right)/[K^{-1}]_{ii}. \end{aligned} \tag{3.18}$$

With partial derivatives calculated, we can use gradient decent based optimizer to minimise our objective function $G(\mathbf{X},\mathbf{y},\theta)$, thus find the optimal $\hat{\theta}$ such that

$$\hat{\theta} = \underset{\theta\in\boldsymbol{\Theta}}{\operatorname{argmax}} G(\mathbf{X},\mathbf{y},\theta). \tag{3.19}$$

Geisser's Predictive mean square Error (GPE)

Another LOO-CV based objective function is Geisser's Predictive mean square Error (GPE)[21, Geisser and Eddy][19, Sundararajan and Keerthi]. GPE is define by

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[(y_i-\hat{y}_i)^2], \tag{3.20}$$

where $\hat{y}_i$ is the predicted value from the model using data set $(\mathbf{X}_{-i}.\mathbf{y}_{-i})$ and y_i is the true target value. GPE measures the average expected square error derived from LOO-CV. The objective function corresponding to GPE is

$$G_E(\mathbf{X},\mathbf{y},\theta)=\frac{1}{n}\sum_{i=1}^{n}\int(y_i-\hat{y}_i)^2 p(\hat{y}_i|\mathbf{X},\mathbf{y}_{-i},\theta)d\hat{y}_i, \tag{3.21}$$

where $\hat{y}_i$ follows a Gaussian distribution with mean μ_i and variance σ_i^2 given by eq. 3.16. Thanks to the analytical tractability of Gaussian distribution, we can simplify the objective function as

$$G_E(\mathbf{X},\mathbf{y},\theta)=\frac{1}{n}\sum_{i=1}^{n}(y_i-\mu_i)^2+\sigma_i^2. \tag{3.22}$$

We can see from the expression of G_E, this objective function will aim to minimise the deviation between the predictive mean and the true value as well as the predictive variance. In the most ideal case, G_E should be very closed to zero, meaning that expected error of training set derived based on LOO-CV should be very small.

We can also use chain rule and eq. 3.14 to calculate the partial derivatives of GPE objective function

$$\frac{\partial G_E(\mathbf{X},\mathbf{y},\theta)}{\partial\theta_j}=\frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i-\mu_i)^2}{\partial\mu_i}\frac{\partial\mu_i}{\partial\theta_j}+2\sigma_i\frac{\partial\sigma_i}{\partial\theta_j}. \tag{3.23}$$

Plug in eq. 3.17 to get

$$\frac{\partial G_E(\mathbf{X},\mathbf{y},\theta)}{\partial\theta_j}=\frac{1}{n}\sum_{i=1}^{n}\left([Z_jK^{-1}]_{ii}\left(1+2\frac{\alpha_i^2}{[K^{-1}]_{ii}}\right)-2\alpha_i[Z_j\alpha]_i\right)/[K^{-1}]_{ii}, \tag{3.24}$$

where $\alpha=K^{-1}\mathbf{y}$ and $Z_j=K^{-1}\frac{\partial K}{\partial\theta_j}$.

Once we have calculated the partial derivatives, we will follow the similar steps in the maximum likelihood case to find the optimal *hyperparameter* $\hat{\theta}$ such that

$$\hat{\theta}=\underset{\theta\in\mathbf{\Theta}}{\operatorname{argmax}}G_E(\mathbf{X},\mathbf{y},\theta). \tag{3.25}$$

The main difference is that maximum likelihood gives us the probability of training data given the assumption of model while the two LOO-CV based methods estimate the predictive probability and error which gives us a measure of how good our assumption is.

3.3.3 Numerical Results

The three objective proposed above have similar computational expense, all dominated by the inversion of covariance matrix K, which is $\mathcal{O}(n^3)$. However, the two LOO-CV based methods have additional $\mathcal{O}(n^2)$ cost for the entire process of calculating the predictive probability and error. This difference will emerge when the train set is large and when we run a large number of iterations. For 4 batches of 2048 points each, ML takes about one third less time compared to the two LOO-CV based methods. Exact numbers are in Table 6.

Table 3, 4 and 5 show that results from these three methods vary between different kernels. For the data set used, the best performance comes from the GPP method with Matérn32 kernel, which is consistent with results from Table 1 and 2. Out of the three methods, the GPE method is consistently the worst performer with all kernels. One can explain this result by reparameterizing the objective function $G_E(\mathbf{X}, \mathbf{y}, \theta)$.

Start by denoting the covariance matrix by $K = \sigma^2 \bar{K}$, where σ is a variance parameter. Then we have $\bar{K}^{-1} = \sigma^2 K^{-1}$. By eq. 3.16 and 3.22, we get the reparameterized GPE objective function

$$\begin{aligned} G_E(\mathbf{X}, \mathbf{y}, \theta) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_i)^2 + \frac{1}{n}\sum_{i=1}^{n}\sigma_i^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{[K^{-1}\mathbf{y}]_i}{[K^{-1}]_{ii}}\right)^2 + \frac{1}{n}\sum_{i=1}^{n}\frac{1}{[K^{-1}]_{ii}} \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{[\bar{K}^{-1}\mathbf{y}]_i}{[\bar{K}^{-1}]_{ii}}\right)^2 + \frac{\sigma^2}{n}\sum_{i=1}^{n}\frac{1}{[\bar{K}^{-1}]_{ii}}. \end{aligned} \tag{3.26}$$

Since σ is independent of $\bar{K}$, when minimising the objective function, σ will be pushed towards zero. Then the covariance matrix K will be pushed towards zero too, which is not true[19]. This is why results from GPR are bad for all kernel functions.

	Training data absolute difference				Testing data absolute difference			
	10	100	1000	10000	10	100	1000	10000
ML	**47.94**	**74.13**	**94.06**	**99.96**	**27.45**	**41.58**	**65.83**	**88.38**
GPP	42.86	68.66	89.23	99.89	24.45	38.28	60.72	85.57
GPE	15.25	31.98	60.23	81.73	13.03	26.55	52.40	77.86

Table 3: Different training methods for RBF kernel

	Training data absolute difference				Testing data absolute difference			
	10	100	1000	10000	10	100	1000	10000
ML	58.51	83.42	**98.35**	**99.98**	30.76	**44.59**	**68.34**	**88.08**
GPP	**63.22**	**83.58**	97.64	99.83	**31.56**	43.59	64.03	84.97
GPE	17.57	36.76	65.73	85.42	14.03	29.46	54.91	79.26

Table 4: Different training methods for Matern32 kernel

	Training data absolute difference				Testing data absolute difference			
	10	100	1000	10000	10	100	1000	10000
ML	**56.10**	**81.23**	**97.52**	**99.98**	29.56	**45.29**	**68.44**	**87.88**
GPP	57.65	78.82	96.00	99.88	**30.36**	40.98	62.73	84.77
GPE	29.24	57.01	78.88	95.47	20.24	36.77	59.12	83.77

Table 5: Different training methods for Matern52 kernel

	RBF	Matern32	Matern52
ML	5.89s	5.70s	5.93s
GPP	9.01s	8.85s	8.79s
GPE	9.14s	9.04s	8.92s

Table 6: Training time per iteration of entire dataset from different methods. Algorithm is implemented using Python 3.5 and Tensorflow packge. Run on machine with CPU: Intel Xeon E5-2620 v4 2.10GHz, RAM: 32.0GB.

4 Data Analysis

4.1 Data Description

The data used during implementation is propriety data from Citigroup. The feature space contains 1290 dimensions of numeric values and the target value is the price of synthetic CDO. Since all feature dimensions are numeric, this paper will not discuss handling with missing features and converting non-numerical features to numeric features. After removing feature dimensions with variance less than 0.01[3], there are still 1118 dimensions left. Within some of these dimensions, there are large differences between the minimum and maximum values. So before doing any specific data processing, one should first rescale all feature dimensions into the interval of $[0, 1]$ for better computational performance. By rescaling all dimensions into uniform interval, it can effectively avoid model misspecification because of large difference in numeric ranges of feature values[26]. Another benefit of rescaling all feature values into uniform interval lies in the more reasonable initialization of parameters. The *characteristic length-scale* in kernel functions can be conveniently initialized to be 0.5 or 1.0 for all dimensions, then perform optimization. After doing this pre-processing, the remaining section will be focused on dimension reduction techniques.

4.2 Dimension Reduction

Due to the computational expense of calculating the inverse of covariance matrix, which is of order $\mathcal{O}(n^3)$ for a n-dimensional positive semi-definite matrix, it is hard to work with large data set in practice. However, for high dimensional input, one must train the model with a reasonably large amount of data so that on average each dimension is adequately represented. Also, as mentioned above, one has to optimize more parameters for high dimension data if using ARD kernels. This also requires training the model with a large amount of data for parameters to converge. But it is restricted by $\mathcal{O}(n)$ computational complexity of covariance matrix inversion. To tackle this dilemma, one needs to reduce

[3]The variance of a random variable measures how far a set of observations are spread out from the mean. A constant random variable has zero variance. By removing dimensions with near zero variance, we discard those dimensions that are almost constant in all data points.

the dimension of feature space while retaining the information from the original data. In this subsection, two dimension reduction techniques will be presented, namely Principal Component Analysis(PCA) and Autoencoder(AE).

4.2.1 Principal Component Analysis

Principal Component Analysis(PCA) is used extensively in statistics for data representation and dimension reduction. The main purpose of PCA is to find a new coordinate system in which we can express the original data with less dimensions without significant loss of information. The following paragraph will briefly explain how PCA works. For a detailed discuss about PCA, please see [23, Jolliffe].

To perform PCA, one will first need to standardize the data by subtracting mean and dividing standard deviation. Then calculate the sample covariance matrix of the input data $\mathbf{Q} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T$, where each column of $\mathbf{X}$ is one data point in original input space, $\mathbb{R}^d$. Then find the eigenvalues $\lambda_1, \cdots, \lambda_d$ of $\mathbf{Q}$ and corresponding orthogonal eigenvectors $\mathbf{u}_1, \cdots, \mathbf{u}_d$. Once all the eigenvalues have been calculated, one needs to sort the eigenvalues in descending order and corresponding eigenvectors. The eigenvector corresponding to the largest eigenvalue is called first *principal component* and so on. To reduce dimensions, one discards those eigenvectors corresponding to eigenvalues that are smaller than a predefined threshold(small eigenvalues correspond to eigenvectors containing less information about the original data). One can also precede by choosing the first d eigenvalues until

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \geq \epsilon, \tag{4.1}$$

where ϵ is the percentage of variance one wants to keep from the original data. Now there are $k(< d)$ eigenvectors to form the feature matrix $\mathbf{U}$ which is of shape $k \times d$. The dimension reduction is done by multiplying $\mathbf{X}$ by $\mathbf{U}$ from left,

$$\mathbf{X}' = \mathbf{U} \cdot \mathbf{X} \tag{4.2}$$

where $\mathbf{X}'$ is a $k \times n$ matrix with each column a transformed data point in reduced input space $\mathbb{R}^k$.

4.2.2 Autoencoder

An autoencoder is a type of feed forward neural network that is mainly used for efficient data encoding[25, chapter 14]. The most simple case of autoencoder contains three layers – namely, input layer, hidden layer and output layer. This type of autoencoder can be divided into two parts: an encoder function and a decoder function. An encoder function takes the original data as input and transform it to a transformed space. Denote the encoder as $\mathbf{h} = f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d, \mathbf{h} \in \mathbb{R}^m$. A decoder function takes $\mathbf{h}$ as input and outputs a reconstruction of the original data $\mathbf{x}' = g(\mathbf{h})$, where $\mathbf{x}' \in \mathbb{R}^d$. One can train the model by minimize the reconstruction error between $\mathbf{x}$ and $\mathbf{x}'$. Commonly used reconstruction error measure is the Mean Squared Error loss function

$$L(\mathbf{x}, \mathbf{x}') = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{x}'_i)^2. \tag{4.3}$$

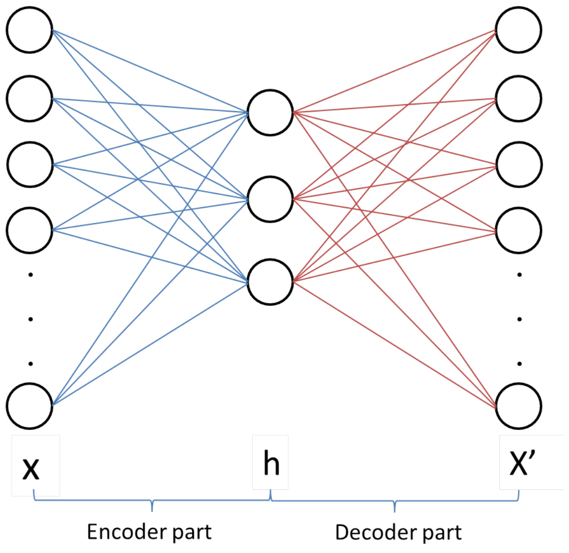

Figure 7: Undercomplete Autoencoder Training Scheme. In an undercomplete autoencoder, number of neurons in the hidden layer is smaller than the number of input dimension.

When $m < d$, it means the encoder function transforms the original data into a lower dimensional space and the decoder function can recover the original data from the transformed space within a tolerable error range. This type of autoencoders is called *undercomplete autoencoders*(hidden layer dimension fewer than input dimension) and is widely used as a dimension reduction tool. By setting the hidden layer size smaller than the input dimension and minimizing the reconstruction error, it forces the autoencoder to learn meaningful transformation of original features. When the activation function used in the encoder part is linear, i.e. $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, and the hidden layer is fully connected, the subspace spanned by the autoencoder by the encoder function is the same as the one spanned by principal components from PCA. Detailed discussion and proof are provided by Plaut in [24].

When the activation function is non-linear, e.g. the logistic function

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{1 + \exp[-(\mathbf{w} \cdot \mathbf{x} + b)]}, \\ g(\mathbf{h}) &= \frac{1}{1 + \exp[-(\mathbf{w}' \cdot \mathbf{h} + b')]}, \end{aligned} \tag{4.4}$$

where $\mathbf{w}, b$ and $\mathbf{w}', b'$ are weights and bias of the encoder and decoder function, the autoencoder can learn a more useful non-linear generalization of the one generated by PCA. After the training process, one can discard the decoder part and only apply the encoder function to the whole data set for dimension reduction purpose.

Apart from dimension reduction, one can also add denoising feature to the autoencoder by feed the model with *noisy* data, then try to reconstruct the *clean* data. This type of autoencoders are called *denoising autoencoders.*

In a denoising autoencoder, a random noise, usually Gaussian noise, is added to the original data $\mathbf{y}$ to get noisy data $\hat{\mathbf{y}}$. Then apply the encoder function and decode function to get $\hat{\mathbf{h}} = f(\mathbf{y})$ and $\hat{\mathbf{y}}' = g(\hat{\mathbf{h}})$. The loss function is defined as Mean Squared Error between the reconstructed $\hat{\mathbf{y}}'$ and the original data $\mathbf{y}$. The training process it to minimize $L(\mathbf{y}, \hat{\mathbf{y}}')$.

4.3 Numerical Results

While traditional dimension reduction techniques like PCA are methods, undercomplete autoencoder is a family of methods. One can configure different autoencoders by number of nodes in the hidden layer or number of hidden layers. One can immediate see this

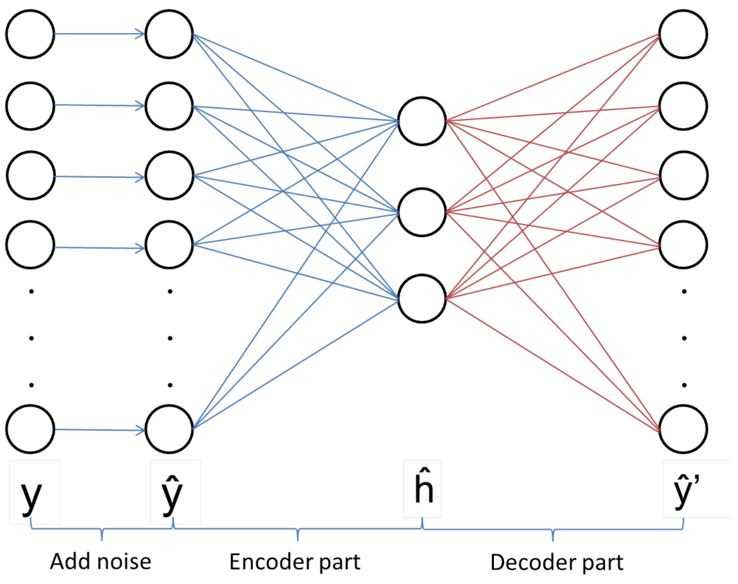

Figure 8: Denoising Autoencoder Training Scheme. Manually add noise to the original data before feeding into the AE.

can go on forever. Like the configuration of a normal Neural Network, the configuration of an undercomplete autoencoder for a specific dataset is often based on experience and experiments. For computational tractability, the implementaion will use only one hidden layer in this paper. The hidden layer is set to be a dense layer, which means every nodes in a layer is fully connected to every nodes from the previous layer. The empirically-derived rule-of-thumb for sizes of hidden layers is 2^n. In this implementation, it starts with 256 nodes, approximately one-fourth of the original dimension and gradually decrease to 16. Training results for autoencoders with different sizes are shown in the following figures.

From figure 9 and 10, when further compressing the data to lower dimension, the AE is actually able to reconstruct the original data within smaller error range. This means the AE with non-linear activation function can effectively learn meaningful features in a much lower dimensional space that can recover the original information within little error.

Feed the reduced data into our GPR model with RBF kernel and Matern32 kernel. Results from differnt methods are shown in table 7. Consistently with previous results, Matern32 kernel continues to outperform the RBF kernel in all reduced data set. This means that indexed by the reduced feature space, our target function is still not smooth enough to be modeled by a GP with RBF kernel. The column $PCA99\%$ means the PCA

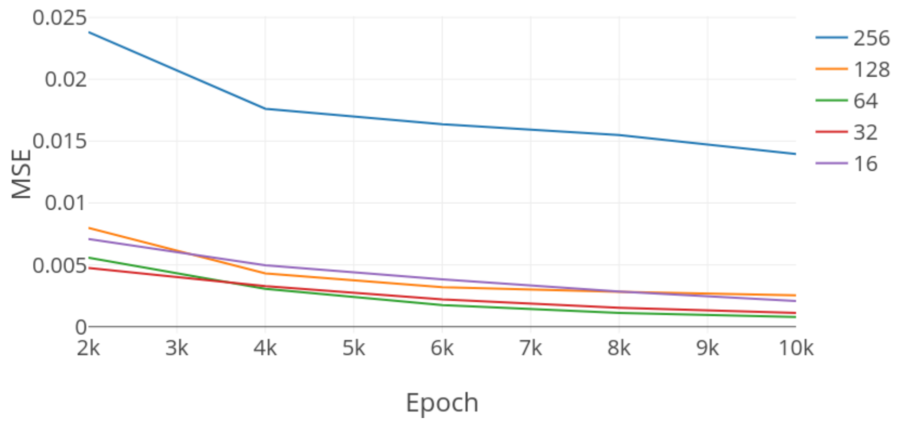

Figure 9: Training Process of Different AE Size. All sizes converge at similar speed, the MSE differences between 4000 and 10000 iterations are subtle. AEs with fewer hidden nodes(128,64,32,16) are able to reconstruct the original data with less error than AE with 256 hidden nodes.

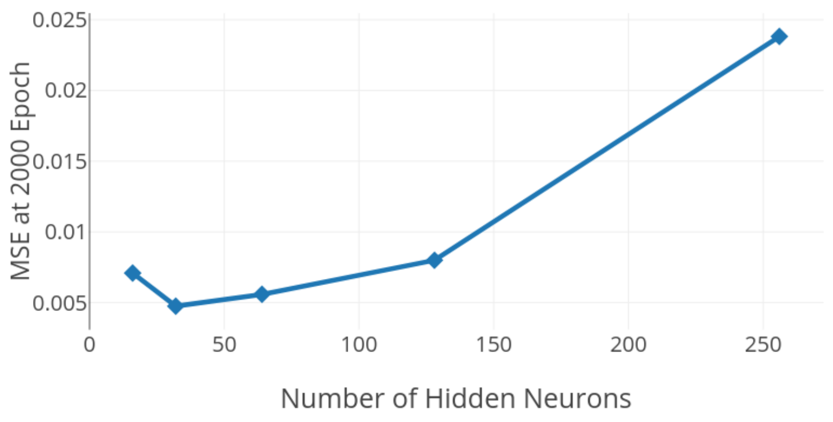

Figure 10: Training Results at 2000 Epochs. Reconstuction error is at lowest when the AE has 32 hidden nodes.

remains 99% of the variance from the original data when performing PCA, which will reduce feature dimensions to 225. The results from $PCA99\%$ is worse than those from AE with different sizes. This is because PCA only performs linear transformation from the original feature space to the reduced space. For the data set we use, the reduced space from PCA method does not contain as much information as those spanned from AEs. On the other hand, the non-linear activation function embedded in AEs is able to learn useful non-linear transformation from the original feature space that can be used to reconstruct the original information with little error. Results from GPR prediction shows that this non-linear transformation does not necessarily benefit from the increasing in the number of hidden nodes. This means for the used data set, non-linear mapping into a smaller space is able to retrain more information than a larger one. For the AEs used, the one with 64 hidden nodes performs the best.

		256		128		64		32		16		PCA 99%	
		RBF	Matern32	RBF	Matern32	RBF	Matern32	RBF	Matern32	RBF	Matern32	RBF	Matern32
Training	10	31.31	16.60	32.60	57.17	34.63	58.73	24.17	**58.85**	41.27	54.43	22.68	23.08
data	100	54.31	37.51	57.07	77.45	58.70	**79.28**	54.04	78.80	64.00	75.99	48.24	51.97
absolute	1000	76.39	64.70	78.34	93.46	79.50	**94.82**	74.44	94.48	84.00	91.38	69.51	71.11
difference	10000	91.82	82.24	92.49	98.88	93.85	98.90	92.04	98.82	94.53	**99.24**	88.39	87.71
Testing	10	18.22	10.49	18.95	26.59	18.03	**26.77**	12.88	25.94	20.42	24.75	12.83	13.43
data	100	30.63	23.46	33.58	**39.56**	33.21	38.64	27.69	39.28	32.38	37.63	28.59	27.60
absolute	1000	52.25	39.56	54.00	60.07	53.54	**60.07**	47.01	57.96	49.68	56.30	47.29	45.45
difference	10000	78.29	72.59	77.92	81.14	78.29	80.40	76.91	**82.34**	77.09	80.13	75.08	72.68

Table 7: Autoencoder and PCA results of RBF and Matern32 kernels; the numbers, 128, 64, ..., mean the number of neurons in the hidden layer of an undercomplete autoencoder. PCA 99% means sum of chosen eigenvalues is 99% of the sum of total eigenvalues. Results from AEs are consistently better than those from PCA. This means linear transformation used in PCA is not able to retain as much information as non-linear transformation in AEs. Adding the hidden nodes in AE does not necessarily improve prediction performance from GPR model.

5 Conclusion, Discussion and Further Study

This section will wrap up what has been shown in previous sections and discuss other applications of GP for further study.

This paper starts with brief introduction to pricing formulas for synthetic CDO – a structured credit product. Then it proposes an alternative way of replicating prices using a Bayesian learning model – Gaussian Process Regression. In section 2, it has shown Gaussian process from a extended weight-space view and a more intuitive function-space view. For a zero mean Gaussian process as proposed in this paper, the behaviour of the GP is fully specified by its covariance matrix produced by kernel functions. There are numerous choices for kernels function. Section 3 has discussed three stationary kernel functions namely, Radial Basis Function(RBF) and Matérn family kernel functions(Matérn32 and Matérn52), and one non-stationary kernel function – Linear kernel function. Section 3 has also proposed three objective functions for parameter optimization – namely, Maximum Likelihood(ML), Geisser's surrogate Predictive Probability(GPP) and Geisser's Predictive mean square Error(GPE). In terms of computation complexity, ML objective functions requires less time for each iteration during training than other two objective functions. Numerical results are shown in Section 3.3.3. For the used dataset, Matérn32 kernel function combined with GPP objective function gives the best predictive results.

The expensive computational complexity($\mathcal{O}(n^3)$) of GPR model during training process prohibits us from using a larger, more representative data set. To tackle this problem, we have used two dimension reduction techniques – namely, Principal Component Analysis(PCA) and Autoencoder(AE) and compared the results on our dataset in Section 4.3. When applying the reduced data to our GPR model, results from AEs with different number of hidden nodes are consistently better than those from PCA. This is because the non-linear transformation performed by the activation function of AEs can produce a reduced space retaining more meaningful information from the original dataset than the linear transformation in PCA.

The autoencoder is a family of methods. One can construct complex hidden layers and nodes configuration as well as training loss function when dealing with a specific data set. For computational tractability, this paper has used one dense hidden layer for the implementation. However, one can build an autoencoder tailored for a specific data set for better performance.

It has been shown in Section 3.2 that sample paths of GPs governed by a Matérn32 kernel function are rougher than those by a RBF kernel function. Prediction results also validate that Matérn32 kernel performs better than RBF kernel on the dataset used in this paper. However, when applied to a another dataset, one needs to first analyse the distribution of features and the behaviour of the target function so that one can construct a suitable kernel function. It is also worth exploring that different kernel functions can be applied to different dimensions of the feature space, based on their different distribution and the smoothness of target values with respect to that dimension.

Since differentiation is a linear operation, the derivative of a Gaussian process is another Gaussian process. Thus one can model the derivative of the target function as a GP and make inference from that. This can be applied to modelling underlyings of a structured or exotic products for hedging and risk management purposes. The covariance of partial derivative with function value and covariance between partial derivatives can be specified by the kernel function k in the following form

$$\begin{aligned} \operatorname{cov}\left(f_i, \frac{\partial f_j}{\partial x_{d_j}}\right) &= \frac{\partial k(\mathbf{x}_i, \mathbf{x}_j)}{\partial x_{d_j}} \\ \operatorname{cov}\left(\frac{\partial f_i}{\partial x_{d_i}}, \frac{\partial f_j}{\partial x_{d_j}}\right) &= \frac{\partial^2 k(\mathbf{x}_i, \mathbf{x}_j)}{\partial x_{d_i} \partial x_{d_j}}, \end{aligned} \tag{5.1}$$

where d_j denotes the j^{th} dimension in the feature space[13, sec 9.4]. If certain dimensions in the feature space are of interest, one can just remove columns in the covariance matrix that are irrelevant. Inference and predictions when modelling with partial derivative can be done similarly to the procedure described in this paper.

Appendix

Matrix Inversion Lemma

Matrix Inversion Lemma also known as the Woodbury, Sherman & Morrison formula[28]. states that

$$(Z + UWV^T)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^T Z^{-1}U)^{-1}V^T Z^{-1},$$

assuming the relevant matrices inverses all exist. Here Z is $n \times n$, W is $m \times m$ and U, V are $n \times m$.

The equation for determinants is

$$|Z + UWV^T| = |Z||W||W^{-1} + V^T Z^{-1}U|.$$

Let the invertible $n \times n$ matrix K and its inverse V be partitioned into

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}, \qquad V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix},$$

where K_{ij} and V_{ij} are submatrices. Then we have

$$\begin{aligned} V_{11} &= K_{11}^{-1} + K_{11}^{-1}K_{12}MK_{21}K_{11}^{-1} \\ V_{12} &= -K_{11}^{-1}K_{12}M \\ V_{21} &= -MK_{12}K_{11}^{-1} \\ V_{22} &= M \end{aligned}$$

where $M = (K_{22} - K_{21}K_{11}^{-1}K_{12})^{-1}$.

References

[1] Youssef Elouerkhaoui. *Credit Correlation - Theory and Practice.* Springer. 2017

[2] Damiano Brigo and Fabio Mercurio. *Interest Rate Models - Theory and Practice.* Springer, Berlin, Heidelberg. 2006

[3] Dominic O'Kane. *Modelling Single-name and Multi-name Credit Derivatives.* Wiley Finance series. 2008

[4] G. Matheron. *The Intrinsic Random Functions and Their Applications.* Advances in Applied Probability, 5:439-468. 1973.

[5] A. G. Journel and C. J. Huijbregts. *Mining Geostatistics.* Academic Press. 1978.

[6] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Regression.* Advances in Neural Information Processing Systems 8, pages 514-520. MIT Press 1996.

[7] Christopher K. I. Williams. *Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond.* Learning in Graphical Models, Springer Netherlands, pages 599–621. 1998.

[8] David J. c. MacKay. *Gaussian Processes - A Replacement for Supervised Neural Networks?.* http://www-clmc.usc.edu/publications/M/mackay-LectureNotesGP.pdf. 1997.

[9] Carl E. Rasmussen. *Gaussian Processes in Machine Learning.* Advanced Lectures on Machine Learning. ML 2003. Lecture Notes in Computer Science, vol 3176. Springer, Berlin, Heidelberg, pages 63-71. 2003

[10] Steven P. Lalley. *Introduction to Gaussian Process.* https://galton.uchicago.edu/ lalley/Courses/386/GaussianProcesses.pdf

[11] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, New York, third edition, 1991.

[12] William M. Bolstad and James M. Curran. *Introduction to Bayesian Statistics.* John Wiley & Sons,third edition, pages 109-128. 1991.

[13] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Process for Machine Learning*. MIT Press, 2006.

[14] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, New York. Second edition. 1985.

[15] M. L. Stein. *Interpolation of Spatial Data*. Springer-Verlag, New York. 1999.

[16] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, New York. 1965.

[17] Yoshua Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*. https://arxiv.org/pdf/1206.5533v2.pdf, 2012

[18] G. C. Cawley and N. L. C. Talbot. *Fast Exact Leave-one-out Cross Validation of Sparse Least Squares Support Vector Machines*. Neural Networks, 17(10): 1467-1475, 2004

[19] S. Sundararajan and S. S. Keerthi. *Predictive Approaches for Choosing Hyperparameters in Gaussian Process*. Neural Computation, 13:1103-1118, 2004

[20] S. Geisser. *The Predictive Sample Reuse Method with Applications*. Journal of the American Statistical Association, 70, 320-328, 1975

[21] S. Geisser. and W. F. Eddy. *A Predictive Approach to Model Selection*. Journal of the American Statistical Association, 74, 153-160, 1979

[22] M. Neal. *Bayesian Learning for Neural Networks*. Springer, New York. Lecture Notes in Statistics 118. 1996

[23] I. T. Jolliffe. *Principal Component Analysis*. Springer, New York. second edition. 2002

[24] Elad Plaut. *From Principal Subspaces to Principal Components with Linear Autoencoders*. https://arxiv.org/pdf/1804.10253.pdf. 2018

[25] Ian Goodfellow, Yoshua Bengio and Aaron Courvillet. *Deep Learning*. MIT Press. http://www.deeplearningbook.org. 2016

[26] S. B. Kotsiantis, D. Kanellopoulos and P. E. Pintelas. *Data Preprocessing for Supervised Leaning*. International Journal of Computer Science Volume 1 Number 1, ISSN 1306-4428. 2006

[27] I. T. Jolliffe and J. Cadima. *Principal component analysis: a review and recent developments*. Philosophical transactions Series A, Mathematical, physical, and engineering sciences. 2016;374(2065):20150202. doi:10.1098/rsta.2015.0202.

[28] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Second edition. 1992.