Jomard
Publishing

A NEW MONTE CARLO BASED EXACT ALGORITHM FOR THE GAUSSIAN PROCESS CLASSIFICATION PROBLEM

Amir F. Atiya[1*]**, Ahmed H. Abdel-Gawad**[1]**, Hatem A. Fayed**[2]

[1]Department of Computer Engineering, Cairo University, Cairo, Egypt
[2]University of Science and Technology, Mathematics Program, Zewail City of Science and Technology, 6th of October City, Egypt

Abstract. Gaussian process is a very promising novel technology that has been applied to both the regression problem and the classification problem. While for the regression problem it yields simple exact solutions, this is not the case for the classification problem, because we encounter intractable integrals. In this paper we develop a new derivation that transforms the problem into that of evaluating the ratio of multivariate Gaussian orthant integrals. Moreover, we develop a new Monte Carlo procedure that evaluates these integrals. It is based on some aspects of bootstrap sampling and acceptance-rejection. The proposed approach has beneficial properties compared to the existing Markov Chain Monte Carlo approach, such as simplicity, reliability, and speed.

1 Introduction

Gaussian process classification (GPC) is a promising Bayesian approach to the classification problem. It is based on defining a so-called latent variable for every pattern, and setting the prior based on proximity relations among the patterns. Based on the prior and through a series of integrals, the posterior of a test pattern will determine its classification (see the reviews of Rasmussen & Williams (2005), Nickisch & Rasmussen (2008), and Seeger (2004)). Unfortunately, this resulting multi-integral formula is intractable, and can only be solved through some approximations or using lengthy algorithms. In spite of that its superior performance has been pointed out in several application studies, such as image analysis Batsch et al. (2019), medical data set classification Jenssen et al. (2007), remote sensing Bazi & Melgani (2010), and active learning Schreiter et al. (2015) and Elreedy et al. (2019). However, this computational issue hampers its wide applicability to large data sets. In this paper, we propose a new simplified, but exact, formula for the binary GPC problem. The proposed method is based on applying some substitutions and transformations that convert the problem into that of evaluating a ratio of two orthant integrals of a multivariate Gaussian density. Moreover, we develop a new Monte-Carlo-type algorithm to evaluate these orthant integrals. The proposed algorithm is based on some aspects of bootstrap sampling and acceptance-rejection.

The approaches in the literature for the GPC problem can be categorized into solutions based on analytical approximation of the integrals and solutions based on Monte Carlo sampling (see

the extensive review of Nickisch & Rasmussen (2008). Among the well-known proposed methods from the first category are the Laplace's approximation (Williams & Barber, 1998) and the expectation propagation (Minka, 2001). Also, some other efficient approximation-based methods include the work of Csáto et al. (2000), Opper & Winther (2000), Gibbs & Mackay (2000), Rifkin & Klautau (2004), Jaakkola & Haussler (1999), and Kim & Ghahramani (2006). Work on the second category, the Monte Carlo approach, has been more scarce. Almost all of the approaches are based on the Markov Chain Monte Carlo (MCMC) concept. Neal's so-called *Annealed Importance Sampling (AIS)* (Neal, 1998) uses an approximate posterior, rather than the prior, as a starting point for evaluating the marginal likelihood (to be described shortly). The *Hybrid Monte Carlo (HMC)* (Neal (1999) is based on the concept of "importance density". Murray et al. (2010) proposed the Elliptical Slice Sampler (ESS). It is based on sampling over an elliptical slice to efficiently obtain the step size. Titsias et al. (2009) introduced a novel MCMC approach by making use of a low dimensional vector of control variables (see also Titsias & Lawrence (2010)). Vehtari et al. (2000) proposed a novel way to choose effective starting values for the MCMC based on early stopping. Barber & Williams (1997) proposed a hybrid method, that uses partly an approximation and partly the MCMC procedure.

Nickisch & Rasmussen (2008)) provided a comprehensive review and comparison between the different GPC approximations and the MCMC approaches. They came to the conclusion that the MCMC-type approaches are superior in performance, but of course computationally more extensive. This is because they can obtain the exact solution of the integral formula, provided the size of the run is large enough. As such, they are *not* based on approximations, as in the first category of algorithms.

Kuss & Rasmussen (2005) compared between a number of GPC approximations. Rather than improving the approximation ability or the computation speed, some researchers considered short-cuts to the the GPC model itself to achieve better efficiency or performance, such as sparse GPC models (see Qi et al. (2010), Csáto & Opper (2002) Vanhatalo & Vehtari (2010), and Titsias (2009)), and low-dimensional manifold embedding (see Urtasun & Darrell (2007) and Nickisch & Rasmussen (2010)).

A majority of the work in the literature and the above reviewed methods consider the binary classification case. Nevertheless, some researchers extended The GPC problem to the multi-class case (for example Girolami & Rogers (2006), Hernández-Lobato et al. (2011) and Seeger & Jordan (2004)).

The method proposed in this paper fits more into the second category described above (i.e. exact Monte Carlo-based), but it is not based on the MCMC concept. It is guaranteed to converge as close as possible to the exact solution of the multi-integral formula, provided we use a large enough sample of generated points. The advantages of the proposed algorithm is that it does not require any parameter-tuning (other than specifying the number of Monte Carlo generated points), is consistent, and reliable. (In short it works all the time, we tested hundreds of problems, some as high as 2000 dimensional problems.) It also compares favorably in terms of speed and accuracy to the other MCMC approaches, especially for the evaluation of the marginal likelihood. The marginal likelihood is an expression for the likelihood of the data given the parameters. The hyperparameters are typically tuned by optimizing the marginal likelihood function. Because of repeated evaluations of the likelihood function, this step is the most time-consuming part, and the speed-up provided by the proposed algorithm will lead to a significant computational benefit. A beneficial aspect of the proposed integral formulation is that it gives many insights into the different influencing factors. For example, one can obtain the limiting behavior of the covariance matrix parameters, and therefore understand the classification behavior when moving their values in certain directions. Also, the other advantage of the integral formulation is that it is given in terms of multivariate Gaussian orthant integrals. These are well-researched integrals, and several approximations exist in the literature. So, this could possibly open the way for new competitive approximations to the GPC problem.

The paper is organized as follows. Next section we present an overview and definition of the GPC problem. In Section 3 we propose the new formulation that is obtained by simplifying the multi-integral formula. Section 4 presents an overview of the multivariate Gaussian integral that has to be evaluated. In Section 5 we propose the new Monte Carlo model for evaluating the integral. Section 6 provides the experimental study to assess the new model, and Section 7 is the conclusion.

2 The Gaussian Process Classification Problem

Gaussian process classifiers (GPC) are based on defining a "latent state" f_i for every training pattern. It is a central variable in the formulation which measures some sort of degree of membership to one of the classes. Let y_i denote the class membership of training pattern i, where $y_i = 1$ denotes Class 1 and $y_i = -1$ denotes Class 2. The latent variable f_i, whose range is from $-\infty$ to ∞, is mapped into class posterior probability through a monotone squashing function σ that has a range of $(0, 1)$, as follows.

$$J = P(y_i = 1|f_i) = \sigma(f_i) \tag{1}$$

There are two typical forms for σ in the GPC literature: the logit (or logistic function) and the probit (or cumulative Gaussian integral). As argued by Nickisch & Rasmussen (2008), both choices are effectively quite similar. In this work we consider only the probit function.

In what is next we will follow closely the terminology of Rasmussen & Williams (2005). Let us arrange the latent variables and the class memberships in one vector each: $\mathbf{f} = (f_1, \ldots, f_N)^T$, and $\mathbf{y} = (y_1, \ldots, y_N)^T$. Note that each index of the afforementioned vectors pertains to a specific training pattern, and N is the size of the training set. Let $\mathbf{x}_i$ be the feature vector of training pattern i. Moreover, let us arrange all training vectors $\mathbf{x}_i$ as rows in a matrix X. Let $\mathbf{x}_*$ denote the feature vector of the test pattern, whose class needs to be evaluated. Let its latent state be f_*.

The latent state vector $\mathbf{f}$ obeys an a priori density that is assumed to be a multivariate Gaussian (therefore the name Gaussian processes). From an a priori point of view, patterns that are close (in the features space) are more likely to belong to the same class. So this prior density is selected to reflect that property. Patterns with nearby feature vectors have highly correlated latent variables f_i, and as the patterns become more distant the correlation decays. The a priori density can be written as

$$p(\mathbf{f}|X) = \mathcal{N}(\mathbf{f}; 0, \Sigma) \tag{2}$$

where $\mathcal{N}(\mathbf{f}; \mu, \Sigma)$ denotes a Gaussian density of variable $\mathbf{f}$ having mean vector μ and covariance matrix Σ. The covariance matrix has elements that are a function of the distance between two feature vectors $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ and is so designed to achieve this aforementioned correlation behavior (see Rasmussen & Williams (2005) for a detailed discussion and examples of covariance functions). A particularly prevalent choice of the covariance matrix is the so called "RBF" covariance matrix, given by:

$$\Sigma_{ij} = \beta e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\alpha^2}} \tag{3}$$

The α and β parameters (called respectively the length scale and the latent function scale) are very influential in the performance of the classifier, and tuning them has to be done with care (see Sundararajan & Keerthi (2001)). More will be said later on them.

A test pattern's latent variable f_* will have similar correlation structure as the training patterns. Consider the augmented training latent state vector and test point latent state. It is given as Gaussian, as follows:

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix}; 0, \begin{bmatrix} \Sigma & \Sigma_{X\mathbf{x}_*} \\ \Sigma_{X\mathbf{x}_*}^T & \Sigma_{\mathbf{x}_*\mathbf{x}_*} \end{bmatrix} \right) \tag{4}$$

where $\Sigma_{X\mathbf{x}_*}$ is the covariance between the training latent variables and the test latent variable (it is a vector), and $\Sigma_{\mathbf{x}_*\mathbf{x}_*}$ is the variance of f_*. A key to estimating the class membership of the test point is to evaluate the probability density of its latent state f_*, conditional on all the information that is available from the training set:

$$p(f_*|X, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f}|X, \mathbf{y}) d\mathbf{f} \tag{5}$$

where $p(\mathbf{f}|X, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X)}{p(\mathbf{y}|X)}$ is the posterior of the latent variables. Of course this posterior density reflects how the prior density is altered to take into account the observed class memberships $\mathbf{y}$.

Then, we compute the probability of Class 1 averaged over the conditional density of f_*:

$$J_* \equiv p(y_* = +1|X, \mathbf{y}, \mathbf{x}_*) = \int \sigma(f_*) p(f_*|X, \mathbf{y}, \mathbf{x}_*) df_* \tag{6}$$

where we used the fact that $\sigma(f_*)$ signifies the conditional given in Eq. (1). We get

$$p(f_*|X, \mathbf{y}, \mathbf{x}_*) = \frac{\int p(f_*|X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|X) d\mathbf{f}}{p(\mathbf{y}|X)} \tag{7}$$

where

$$p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{N} p(y_i|f_i) = \prod_{i=1}^{N} \sigma(y_i f_i) = \prod_{i=1}^{N} \int_{-\infty}^{y_i f_i} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx, \tag{8}$$

where we used the fact that σ is the probit function (integral of the Gaussian function). Note that $P(y_i = -1|f_i) = 1 - P(y_i = 1|f_i) = 1 - \sigma(f_i) = \sigma(-f_i) = \sigma(y_i f_i)$ because of the point symmetry of σ. Also,

$$p(f_*|X, \mathbf{x}_*, \mathbf{f}) = \mathcal{N}(f_*; \mathbf{a}^T \mathbf{f}, \sigma_*^2) \tag{9}$$

where

$$\mathbf{a} = \Sigma^{-1} \Sigma_{X\mathbf{x}_*} \tag{10}$$

$$\sigma_*^2 = \Sigma_{\mathbf{x}_*\mathbf{x}_*} - \Sigma_{X\mathbf{x}_*}^T \Sigma^{-1} \Sigma_{X\mathbf{x}_*} \tag{11}$$

We utilized formulas expressing the conditional of a multidimensional Gaussian distribution, applied to Eq. (4).

All past formulas follow from straightforward probability manipulations, and they are described clearly in Rasmussen & Williams (2005), p. 16 Eq. 2.19. Equation (6), representing the posterior probability corresponding to the test pattern $\mathbf{x}_*$, is the main quantity needed to classify the pattern. Thus, $J_* \geq 0.5$ means that the pattern should be classified as Class 1, and otherwise it should be classified as Class 2.

Another important quantity that is needed is the so-called marginal likelihood L, defined as

$$L = p(\mathbf{y}|X) \tag{12}$$

It is an important quantity for the purpose of tuning the two hyperparameters (α and β). By maximizing the marginal likelihood, we arrive at hyperparameters that are most consistent with the observed data. As such, any method should also be able to efficiently evaluate the marginal likelihood. See Rasmussen & Williams (2005) for more information about the marginal likelihood.

3   The Proposed Simplification of the Multi-Integral

3.1   Variable Transformation

Evaluating Equations (6), (7) analytically is very hard to accomplish. The difficulty arises also when attempting to evaluate them numerically because of the high dimensionality of the integrals (for example for a problem with a training set of size 1000 we are dealing with more than a thousand-fold integral). Also, attempting a standard Monte Carlo evaluation leads to some practical problems, among them is the fact that $\prod_{i=1}^{N} \sigma(y_i f_i)$ turns out to be usually a very small number (with a negative exponent with a very large magnitude). So to summarize, we are dealing with a very hard problem if an exact solution is to be sought. Here we develop a procedure that transforms the problem into the more approachable form of integrals of multivariate Gaussian functions. Specifically, we perform some substitutions and transformations of variables that will convert the problem into evaluating orthant integrals of some multivariate Gaussian density. By orthant integral we mean an integral of a zero-mean multivariate Gaussian function over some quadrant, e.g. over $\mathbf{x} \geq 0$. The detailed steps are given below.

Substituting (8), (2), (9) and (7) into (6), we obtain:

$$
J_* = \frac{1}{p(\mathbf{y}|X)} \int\limits_{f_*=-\infty}^{\infty} \left[\int\limits_{u=-\infty}^{f_*} \frac{e^{-\frac{u^2}{2}}}{\sqrt{2\pi}} du \right] \int_f \left[\prod_{i=1}^{N} \int\limits_{z_i=-\infty}^{y_i f_i} \frac{e^{-\frac{z_i^2}{2}}}{\sqrt{2\pi}} dz_i \right]
$$
$$
\mathcal{N}(\mathbf{f}; 0, \Sigma) \, \mathcal{N}(f_*; \mathbf{a}^T \mathbf{f}, \sigma_*^2) df_1 df_2 .. df_N df_* \tag{13}
$$

Rearranging, we get

$$
J_* = \frac{\int\limits_{f_*=-\infty}^{\infty} \int\limits_{u=-\infty}^{f_*} \int\limits_{f_1=-\infty}^{\infty} \cdots \int\limits_{f_N=-\infty}^{\infty} \int\limits_{z_1=-\infty}^{y_1 f_1} \cdots \int\limits_{z_N=-\infty}^{y_N f_N} e^{-\frac{W}{2}} dz_1 ... dz_N df_1 ... df_N du df_*}{(2\pi)^{N+1} \sigma_* |\Sigma|^{\frac{1}{2}} p(\mathbf{y}|X)} \tag{14}
$$

where

$$
W = u^2 + \sum_{i=1}^{N} z_i^2 + \mathbf{f}^T \Sigma^{-1} \mathbf{f} + \frac{(f_* - \mathbf{a}^T \mathbf{f})^2}{\sigma_*^2} \tag{15}
$$

Rewriting W in matrix form:

$$
W = \begin{bmatrix} u & z_1 & \cdots & z_N \end{bmatrix} \begin{bmatrix} u \\ z_1 \\ \vdots \\ z_N \end{bmatrix} + \mathbf{f}^T \Sigma^{-1} \mathbf{f} +
$$
$$
\begin{bmatrix} f_* & f_1 & \cdots & f_N \end{bmatrix} B \begin{bmatrix} f_* \\ f_1 \\ \vdots \\ f_N \end{bmatrix}
$$

where $B = \frac{1}{\sigma_*^2} \begin{bmatrix} 1 \\ -\mathbf{a} \end{bmatrix} \begin{bmatrix} 1 & -\mathbf{a}^T \end{bmatrix}$. The problem with this integral (J_*) is that some of its variables (f_*, f_1, ..., f_N) occur in the limits of the integrals. A transformation can fix this

problem by using the substitution (see the preliminary work of Abdel-Gawad & Atiya (2008)):

$$\mathbf{v} \equiv \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2N+2} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & y_1 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & y_N \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ \mathbf{z} \\ f_* \\ \mathbf{f} \end{bmatrix} \tag{16}$$

We get

$$J_* = \frac{\int\limits_{v_1=0}^{\infty} \int\limits_{v_2=0}^{\infty} \cdots \int\limits_{v_{N+1}=0}^{\infty} \int\limits_{v_{N+2}=-\infty}^{\infty} \cdots \int\limits_{v_{2N+2}=-\infty}^{\infty} e^{-\frac{\mathbf{v}^T D \mathbf{v}}{2}} dv_1 \cdots dv_{2N+2}}{(2\pi)^{N+1} \sigma_* |\Sigma|^{\frac{1}{2}} p(\mathbf{y}|X)} \tag{17}$$

where

$$D = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & I & 0 & -C' \\ -1 & 0 & 1 + \frac{1}{\sigma_*^2} & -\frac{\mathbf{a}^T}{\sigma_*^2} \\ 0 & -C' & -\frac{\mathbf{a}}{\sigma_*^2} & I + \Sigma^{-1} + \frac{\mathbf{a}\mathbf{a}^T}{\sigma_*^2} \end{bmatrix} \tag{18}$$

where $C' = \begin{bmatrix} y_1 & & & \\ & y_2 & & \\ & & \ddots & \\ & & & y_N \end{bmatrix}$ and I is the identity matrix (in both cases in the formula it is

$N \times N$). The integration can then be put in the form:

$$J_* = \frac{1}{|D|^{\frac{1}{2}} |\Sigma|^{\frac{1}{2}} p(\mathbf{y}|X) \sigma_*} \int \mathcal{N}\left(\mathbf{v}; 0, D^{-1}\right) d\mathbf{v} \tag{19}$$

This integral above is called orthant normal integral where the orthant is defined over $\begin{bmatrix} v_1 \\ \vdots \\ v_{N+1} \end{bmatrix} \geq$

0 and $-\infty < \begin{bmatrix} v_{N+2} \\ \vdots \\ v_{2N+2} \end{bmatrix} < \infty$. Let us denote this orthant by *orth*. Consider now the term

$p(\mathbf{y}|X)$. Integrating (7) w.r.t. f_* from $-\infty$ to ∞, and using the fact that $\int p(f_*|X, \mathbf{y}, x_*) df_* = 1$ we get

$$\begin{aligned} p(\mathbf{y}|X) &= \int\limits_{-\infty}^{\infty} \int_{\mathbf{f}} p(f_*|X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|X) d\mathbf{f} \, df_* \\ &= \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx \int_{\mathbf{f}} p(f_*|X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{y}|\mathbf{f}) \\ & \quad p(\mathbf{f}|X) d\mathbf{f} \, df_* \end{aligned} \tag{20}$$

The integral $\int\limits_{-\infty}^{\infty} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx$ in the previous equation is inserted on purpose. It equals 1 so it will not alter the formula. The above integral will be the same as (19) except that the limits

will be over the orthant defined by $\begin{bmatrix} v_2 \\ \vdots \\ v_{N+1} \end{bmatrix} \geq 0$ and $-\infty < \begin{bmatrix} v_1 \\ v_{N+2} \\ \vdots \\ v_{2N+2} \end{bmatrix} < \infty$. The reason is

that the above analysis that led to (19) will apply here except that one of the integral's limits is different (the Gaussian integral with variable x that is inserted in (20), here the upper limit is ∞ instead of f_*). Let us denote this orthant by $orth+$. The expression for the posterior will then be given as.

$$J_* = p(y_* = 1 | X, \mathbf{y}, \mathbf{x}_*) = \frac{\int_{orth} \mathcal{N}\left(\mathbf{v}; 0, D^{-1}\right) d\mathbf{v}}{\int_{orth+} \mathcal{N}\left(\mathbf{v}; 0, D^{-1}\right) d\mathbf{v}} = \frac{I_1}{I_2} \tag{21}$$

3.2 Further Reduction:

The limits for the portion $(v_{N+2}, \ldots, v_{2N+2})^T$ in Expression (21) are from $-\infty$ to ∞, so these variables can be integrated out. This means that the expression can further be reduced from a $2N + 2$ dimensional integral to an $N + 1$ dimensional integral. The detailed steps of this reduction are given below.

Let $\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_{N+1} \end{bmatrix}$ and $\mathbf{v}' = \begin{bmatrix} v_{N+2} \\ \vdots \\ v_{2N+2} \end{bmatrix}$. We can write:

$$I_1 = k_3 \int\limits_{\mathbf{v} \geq 0} \int\limits_{-\infty \leq \mathbf{v}' \leq \infty} \frac{e^{-\frac{1}{2}\left(\mathbf{v}^T A_{11}\mathbf{v} - 2\mathbf{v}^T A_{12}\mathbf{v}' + \mathbf{v}'^T A_{22}\mathbf{v}'\right)}}{(2\pi)^{\frac{N+1}{2}} |A_{22}|^{-\frac{1}{2}}} d\mathbf{v}' d\mathbf{v} \tag{22}$$

where the D matrix defined in (18) is written as $D = \begin{bmatrix} A_{11} & -A_{12} \\ -A_{12} & A_{22} \end{bmatrix}$, $A_{11} = I$, $A_{12} = \begin{bmatrix} 1 & 0 \\ 0 & C' \end{bmatrix}$, $A_{22} = \begin{bmatrix} 1 + \frac{1}{\sigma_*^2} & -\frac{\mathbf{a}^T}{\sigma_*^2} \\ -\frac{\mathbf{a}}{\sigma_*^2} & I + \Sigma^{-1} + \frac{\mathbf{a}\mathbf{a}^T}{\sigma_*^2} \end{bmatrix}$, and $k_3 = (2\pi)^{-\frac{N+1}{2}} |A_{22}|^{-\frac{1}{2}} |D|^{\frac{1}{2}}$. Some manipulations lead to:

$$\begin{aligned} I_1 &= k_3 \int\limits_{\mathbf{v} \geq 0} \int\limits_{-\infty \leq \mathbf{v}' \leq \infty} \frac{e^{-\frac{1}{2}\left(\mathbf{v}' - A_{22}^{-1}A_{12}\mathbf{v}\right)^T A_{22}\left(\mathbf{v}' - A_{22}^{-1}A_{12}\mathbf{v}\right)}}{(2\pi)^{\frac{N+1}{2}} |A_{22}|^{-\frac{1}{2}}} d\mathbf{v}' \\ &\cdot\ e^{-\frac{1}{2}\left(\mathbf{v}^T A_{11}\mathbf{v} - \mathbf{v}^T A_{12}A_{22}^{-1}A_{12}\mathbf{v}\right)} d\mathbf{v} \end{aligned}$$

Now the inside integral w.r.t. $\mathbf{v}'$ equals 1. We get

$$I_1 = k_3 (2\pi)^{\frac{N+1}{2}} |A|^{-\frac{1}{2}} \int\limits_{\mathbf{v} \geq 0} \mathcal{N}\left(\mathbf{v}; 0, A^{-1}\right) d\mathbf{v}, \tag{23}$$

where

$$A = I - A_{12}A_{22}^{-1}A_{12} \tag{24}$$

A similar formula applies for I_2 but with integration limits given by $-\infty < v_1 < \infty$ and $\begin{bmatrix} v_2 \\ \vdots \\ v_{N+1} \end{bmatrix} \geq 0$

Some further simplification (see the Appendix for a detailed proof) leads to

$$A^{-1} \equiv R \;\; = \;\; \begin{pmatrix} 1 + \Sigma_{\mathbf{x}_*\mathbf{x}_*} & \Sigma^T_{X\mathbf{x}_*} C' \\ C' \Sigma_{X\mathbf{x}_*} & C'(I + \Sigma)C' \end{pmatrix} \tag{25}$$

$$= \;\; I + A_{12}\Sigma' A_{12} \tag{26}$$

where Σ' is the composite covariance function (of training patterns plus testing pattern, see Eq. (4)):

$$\Sigma' = \begin{pmatrix} \Sigma_{\mathbf{x}_*\mathbf{x}_*} & \Sigma^T_{X\mathbf{x}_*} \\ \Sigma_{X\mathbf{x}_*} & \Sigma \end{pmatrix}. \tag{27}$$

The final classification posterior probability is thus given by

$$J_* = p(y_* = 1|X, \mathbf{y}, \mathbf{x}_*) = \frac{\int_{orth} \mathcal{N}\left(\mathbf{v}; 0, I + A_{12}\Sigma' A_{12}\right) d\mathbf{v}}{\int_{orth+} \mathcal{N}\left(\mathbf{v}; 0, I + A_{12}\Sigma' A_{12}\right) d\mathbf{v}} = \frac{I_1}{I_2} \tag{28}$$

where $orth$ means the integration over $\mathbf{v} \geq 0$, and $orth^+$ is the integration over the region given

by $-\infty < v_1 < \infty$ and $\begin{bmatrix} v_2 \\ \vdots \\ v_{N+1} \end{bmatrix} \geq 0$ As we have demonstrated in Eq. (20), the term $p(\mathbf{y}|X)$,

which represents the marginal likelihood as described in Eq. (12) and beyond, is basically I_2, or the denominator of the expression for J_*. Thus,

$$L = p(\mathbf{y}|X) \;\; = \;\; \int_{orth+} \mathcal{N}\left(\mathbf{v}; 0, I + A_{12}\Sigma' A_{12}\right) d\mathbf{v} \tag{29}$$

$$= \;\; \int_{orth} \mathcal{N}\left(\mathbf{v}'; 0, C'(I + \Sigma')C'\right) d\mathbf{v}' \tag{30}$$

The last identity is obtained by noting that the limit for v_1 in Eq. (29) stretches from $-\infty$ to ∞ and therefore v_1 can be integrated out. The vector $\mathbf{v}' \equiv (v_2, \ldots, v_{N+1})^T$ corresponds to the variables of the training set, while v_1 corresponds to the test pattern.

3.3   Some Insights

The denominator, as mentioned, represents the marginal likelihood $p(\mathbf{y}|X)$. Essentially, the marginal likelihood, as also observed from the integral, measures how well the class memberships y_i fit with the covariance structure. If the patterns of 1's and -1's multiplied with the components of Σ' (through A_{12}) (in Eq. (29)) emphasize the large covariance elements (through agreeing class memberships, i.e. $y_i y_j = 1$) and deemphasizing the small covariance elements (through disagreeing class memberships, i.e. the multiplied by the factor $y_i y_j = -1$), then we have a relatively large marginal likelihood, and therefore a fairly consistent model. The reason why the positive orthant integral will have higher value for large positive covariances (rather than negative) is that the largest principal components will then be oriented closer to the orthant.

The classification posterior probability J_* is the ratio of the integral over an orthant, and the integral is over $orth+$ which basically extends over two orthants. This makes the expression appropriately smaller than 1. Observing the numerator, we find that the predominant signs (in y_i and therefore also A_{12}) that multiply large covariance elements (in $\Sigma_{X\mathbf{x}_*}$) will determine if the numerator has a large value (therefore the pattern should be classified as Class 1), or a small value (therefore the pattern should be classified as Class 2). Note that the identity matrix component of the covariance matrix in Eqs. (26) and (29) is some kind of regularizing factor that acts to "fatten" density.

Let us now consider the effects of the hyperparameters. Consider the RBF kernel (Eq. (3)). The analysis here complements and in many ways confirms the insightful analysis of

Nickisch & Rasmussen (2008). When either the latent function scale $\beta \longrightarrow 0$ or the length scale $\alpha \longrightarrow 0$ then the covariance matrix in the multivariate Gaussian tends to the identity matrix. In case of identity matrix the orthant integral equals 2^{-d} where d is the dimension. So, we end up with $L = 2^{-N}$ and $J_* = \frac{1}{2}$. If $\beta \longrightarrow \infty$ then the identity matrix part of the covariance matrix becomes negligible compared to the other part, see Eq. (26), and we get a formula similar to Eq. (28), but without the identity matrix added in the covariance expression. If $\alpha \longrightarrow \infty$, then we get the following formula. To avoid distraction to side issues, the proof is not given here.

$$J_* = \frac{\int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} \sigma\left(\sqrt{\beta}u\right)^{N_1+1} \left[1 - \sigma\left(\sqrt{\beta}u\right)\right]^{N_2} du}{\int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} \sigma\left(\sqrt{\beta}u\right)^{N_1} \left[1 - \sigma\left(\sqrt{\beta}u\right)\right]^{N_2} du} \tag{31}$$

where σ is the cumulative Gaussian integral (i.e. the integral of the one-dimensional Gaussian density), and N_1 (N_2) is the number of Class 1 (Class 2) training patterns. This formula, consisting of elements of cumulative Gaussian integral and the Gaussian function, can be further simplified using the methodology of Fayed & Atiya (2014b) and Fayed et al. (2015a). Essentially, the formula above gives some kind of a "soft" counting procedure, without regards to the distances involved. One can see that all test patterns will be classified as only one specific class (the one that wins the counting game).

4   The Multivariate Gaussian Integral

As can be seen the final equations (28), (29), and (30) for the posterior probability and the marginal likelihood are given in terms of two multivariate Gaussian integrals. The difficulty is that these are very high dimensional integrals (the dimension equals the size of the training set), making that a formidable problem. Consider that we would like to apply the standard approach of generating many points according to the multivariate density, and then computing the fraction of points that fall in the considered orthant (area of integration). Such high order integrals are typically a very small number, for example 10^{-50} (both integrals I_1 and I_2 give such orders of magnitudes, but their ratio is a sensible number). So even if we generate trillions of points, essentially no point will happen fall in the area of integration.

There is a large literature on the multivariate Gaussian integral. Interest in the problem started around the forties of last century, and research is continuing since then (see the reviews of Gupta (1963a) and Gupta (1963b), Johnson et al. (1994), and Genz & Bretz (2009)), as it is applicable in many problems such as the first passage time problem for stochastic processes (Atiya & Metwally, 2005). Essentially the state of the art is that a closed form formula exists only for a dimension up to three for the centered case (i.e. the integrals for the zero mean case where the limits are from 0, as is our case, see Eriksson (1990)), and no closed form formulas exist for the non-centered case. There are some special constructions of the covariance matrices for which simplified formulas exist (for any arbitrary dimension). There has also been some series expansions for the centered case in terms of the elements of the covariance matrix (Kendall, 1941; Moran, 1948), in terms of the elements of the inverse of the covariance matrix (Ribando, 2006), or by using Fourier series expansion Fayed & Atiya (2014a) and Fayed et al. (2015b). These formulas, while very elegant and insightful, are intractable for dimensions larger than ten, because of the huge number of combinations of powers of the N^2 variables of the covariance matrix. A parallel track in the attempt to tackle the multivariate Gaussian integral is by applying some efficient numerical integration techniques (see for example Schervish (1984)). Due to the exponential nature of these methods, they are applicable for a dimension up to around 20. Another track considers improvised Monte Carlo methods (Deák, 1986; Genz, 1993; Breslaw, 1994) and Hajivassiliou et al. (1991). Again, these methods have not been demonstrated on high dimensional problems.

5   New Monte Carlo Method

5.1   The Proposed Method

The proposed new Monte Carlo method tackles the high-dimensional multivariate Gaussian integral, and thereby simultaneously evaluates the posterior probability and the marginal likelihood. It combines aspects of rejection sampling and bootstrap sampling. The general idea is to first generate samples for the first variable v_1. Subsequently, we reject the points that fall outside the integral limits (for v_1). Then we replenish in place of the discarded points by sampling with replacement from the existing points (i.e. the points that have been accepted). Next, we move on to the second variable, v_2, and generate points using the conditional distribution $p(v_2|v_1)$. Again, we reject the points of v_2 that fall outside the integration limit, and replenish by sampling with replacement. We continue in this manner until we reach the final variable v_N. The integral value is then estimated as the product of the acceptance ratios of the N variables. Unlike MCMC-type methods, we do not need to perform additional cycles. We cycle only once through the N variables, each time generating a number M of points.

Here are the detailed steps of the algorithm:

1. For $i = 1$ to N perform the following:

2. If $i = 1$ then generate M points $v_1(m)$ from $p(v_1)$. Otherwise, generate M points $v_i(m)$ according to the conditional density function:

$$p \equiv p(v_i|\mathbf{v}_{1:i-1}(m)) \tag{32}$$

   where $\mathbf{v}_{1:i-1}(m)$ is the mth string of points (of variables v_1 to v_{i-1}) already generated in the previous steps.

3. Reject the points $v_i(m)$ that are outside the area of integration, i.e. reject the points $v_i(m) \leq 0$. Assume that there are $M_1(i)$ accepted points and $M_2(i) \equiv M - M_1(i)$ rejected point.

4. Replenish in place of the rejected points, by sampling a number $M_2(i)$ points by replacement from among the accepted points.

5. Once reaching the last dimension $i = N$, we stop, computing the multivariate integral as

$$I = \prod_{i=1}^{N} \left(\frac{M_1(i)}{M} \right) \tag{33}$$

5.2   The Rationale of the Algorithm

The proof that the proposed algorithm leads to an estimate for the multivariate orthant probability is essentially by construction, and this is described here. The multivariate integral can be written as:

$$\begin{aligned} I &= p(v_1 \geq 0, v_2 \geq 0, \ldots, v_N \geq 0) \tag{34} \\ &= p(v_N \geq 0|v_1 \geq 0, \ldots, v_{N-1} \geq 0) \ldots p(v_2 \geq 0|v_1 \geq 0)p(v_1 \geq 0) \tag{35} \end{aligned}$$

Since we generate points according to the distribution in (32), which conditions only on the surviving points that were not being rejected in previous rounds, the generated points will obey the distribution $p(v_i|v_1 \geq 0, \ldots, v_{i-1} \geq 0)$. From this distribution $M_1(i)$ points (out of M) will have $v_i \geq 0$ and will be accepted. As such, the ratio $M_1(i)/M$ is an estimate of the probability $p(v_i \geq 0|v_1 \geq 0, \ldots, v_{i-1} \geq 0)$. Using Eq.(35) we obtain the product formula (33) for the overall multivariate integral. Please note that the bootstrap sampling step does not alter the distribution of the generated points.

5.3   Example:

Consider for example that we would like to compute

$$I = \int_{0.7}^{\infty} \int_{1.4}^{\infty} \int_{1.2}^{\infty} \mathcal{N}(\mu, \Sigma) dv_1 dv_2 dv_3 \tag{36}$$

1. We generate M points from $p(v_1)$ (let $M = 10$). Let these points be 2.1, 0.2, 0.6, 1.8, 2.2, 0.8, 1.3, -0.3, 1.4, 1.6. We reject the points 0.2, 0.6, 0.8, -0.3, as they are less than 1.2 and hence are outside the area of integration. We keep the six accepted points 2.1, 1.8, 2.2, 1.3, 1.4, 1.6, and remove the rejected points. In place of the rejected points, we sample a similar number (i.e. four) from among the accepted points, with replacement. Assume we have done that and we have obtained: 2.1, 1.3, 1.8, 2.1. So, overall we have the following points ($v_1(m)$'s): 2.1, 1.8, 2.2, 1.3, 1.4, 1.6, 2.1, 1.3, 1.8, 2.1.

2. Generate $M \equiv 10$ points from $p(v_2|v_1(m))$. Specifically, we generate one point $v_2(1)$ using the density $p(v_2|v_1 = 2.1)$, then one point $v_2(2)$ using the density $p(v_2|v_1 = 1.8)$, and so on. (Note that $p(v_2|v_1)$ is obtained using simple manipulation of the multivariate Gaussian density and it becomes a univariate Gaussian density.) Let the generated points be $v_2(m) = 2.5, 1.5, 2.0, 1.7, 1.1, 1.5, 1.7, 1.3, 1.9, 2.0$. We reject the points 1.1 and 1.3 as they are below the integration limit of 1.4. Then we sample two more points by replacement in place of these rejected points. We continue in this manner for the remaining dimension. Assume that we rejected three points in this case.

3. The integral estimate is the product of the acceptance ratios, i.e. it equals $\left(\frac{6}{10}\right)\left(\frac{8}{10}\right)\left(\frac{7}{10}\right)$.

5.4   On the Convergence of the Proposed Algorithm

Because the integral is typically a very small number, we will consider here the logarithm of the integral as the target value we would like to estimate, i.e., from Eq.(35):

$$\log(I) = \log\Big[p(v_N \geq 0|v_1 \geq 0, \ldots, v_{N-1} \geq 0)\Big] + \cdots + \log\Big[p(v_2 \geq 0|v_1 \geq 0)\Big] + \log\Big[p(v_1 \geq 0)\Big] \tag{37}$$

Assume for the time being that the values generated are independent. In every step of the algorithm we generate M Bernoulli trials, where each has a probability of $P_i \equiv p(v_i \geq 0|v_1 \geq 0, \ldots, v_{i-1} \geq 0)$ in landing in the integral's sought interval and being accepted for the subsequent steps. Let us analyze the bias and the variance.

$$E\left[\log\left(\frac{M_i}{M}\right)\right] \approx \log(P_i) + \frac{1}{P_i}E\left[\frac{M_i}{M} - P_i\right] - \frac{E\left[\frac{M_i}{M} - P_i\right]^2}{2P_i^2} \tag{38}$$

where the expression in the RHS originates from a Taylor series expansion of $\log(M_i/M)$ around the value $\log(P_i)$, and keeping up to quadratic terms. The expectation in the second term in the RHS equals zero (because it is a binomial process, so the expectation of M_i equals MP_i). The last term in the RHS can also be evaluated, and we obtain the bias as

$$\text{Bias} \equiv E\left[\log\left(\frac{M_i}{M}\right)\right] - \log(P_i) = -\frac{(1-P_i)}{2MP_i} + O(M^{-2}) \tag{39}$$

$$= O(M^{-1}) \tag{40}$$

which means that we can have the bias as close as possible to zero, as the number of generated points becomes very large.

Concerning the variance, we get

$$\text{Var}\left[\log\left(\frac{M_i}{M}\right)\right] = \frac{1 - P_i}{MP_i} + O(M^{-2}) = O(M^{-1}) \tag{41}$$

For the overall integral, we get

$$\log(I) = \sum_{i=1}^{N} \log(P_i) \tag{42}$$

with the bias and variance becoming

$$\text{Bias}(\hat{I}) = -\frac{1}{M}\left[\sum_{i=1}^{N}\frac{(1 - P_i)}{2P_i}\right] + O(M^{-2}) \tag{43}$$

$$\text{Var}(\hat{I}) = \frac{1}{M}\left[\sum_{i=1}^{N}\frac{(1 - P_i)}{P_i}\right] + O(M^{-2}) \tag{44}$$

Thus, the mean square error (MSE) goes to zero as $M \longrightarrow \infty$. Note that P_i is the outcome of a one-dimensional integral, so it is expected to be in the middle range of $(0, 1)$.

As a benchmark comparison, consider the basic Monte Carlo integration algorithm, where we generate a number of points according to the multivariate Gaussian distribution and evaluate the fraction of points falling in the area of integration. In that case the mean square error is $(1 - I)/(MI) + O(M^{-2})$, where I is the value of the multivariate integral. One can see that the MSE is very large because typically I is infinitessimally small.

When we derived the above formula, we assumed that the samples are independent, as an approximation. Strictly speaking they are not, because of the following reason. Consider two points $v_j(1)$ and $v_j(2)$ generated according to $p(v_j|v_1 \geq 0, \ldots, v_{j-1} \geq 0)$. Tracking backwards from their values at dimension j and going upstream through the conditioned variables, we could find one variable, say v_{j-k}, that is a common conditioned variable to the two generated points $v_j(1)$ and $v_j(2)$ (i.e. a common ancestor). This is because of the bootstrap sampling procedure. However, we argue that the dependence will be fairly small. This is because equally-valued samples will get completely dispersed when we generate samples for the next variable, and so the dependence will decay fast. So, the net effect of this dependence is to have a somewhat higher MSE, but it would still be the same order, i.e. $O(M^{-1})$, and with higher coefficient. (It is akin to estimating the mean of a variable using generated points having a banded covariance matrix, the MSE will still be $O(M^{-1})$.)

5.5   On Generating from the Distribution $p(v_i|\mathbf{v}_{1:i-1}(m))$

In Step 2 in the algorithm described in Subsection 5.1, we need to generate from the conditional Gaussian distribution. This can be accomplished using the well known identity (assume mean($\mathbf{v}$)=0):

$$p(v_i|\mathbf{v}_{1:i-1}) = \mathcal{N}\left(v_i, \mathbf{b}_i^T\mathbf{v}_{1:i-1}, \sigma_i^2\right) \tag{45}$$

where

$$\mathbf{b}_i = R_{1:i-1,1:i-1}^{-1}R_{1:i-1,i} \tag{46}$$

$$\sigma_i^2 = R_{i,i} - R_{i,1:i-1}R_{1:i-1,1:i-1}^{-1}R_{1:i-1,i} \tag{47}$$

and where $R \equiv I + A_{12}\Sigma'A_{12}$ is the covariance matrix pertaining to the multivariate Gaussian (see Eqs. (25) and (28)), and the notation $A_{i:j,k:l}$ means the submatrix constructed from A by taking rows i to j and columns k to l.

We have to compute these variables in Eqs. (46) and (47), including inverting a matrix every step, i.e. N times. We present here a computationally more efficient algorithm based on a recursive computation of the quantities in Eqs. (46) and (47). Assume that we have performed the computations at Step i, i.e. that $\mathbf{b}_i$, σ_i^2 and $Q_i \equiv R_{1:i-1,1:i-1}^{-1}$ are available. Proceeding to the next step $i+1$, we first tackle Q_{i+1}. Using the partitioned matrix inversion (Horn & Johnson, 1985), we get

$$
Q_{i+1} = \begin{pmatrix} R_{1:i-1,1:i-1} & R_{1:i-1,i} \\ R_{1:i-1,i}^T & R_{ii} \end{pmatrix}^{-1}
$$
$$
= \begin{pmatrix} R_{1:i-1,1:i-1}^{-1} + \frac{1}{k}R_{1:i-1,1:i-1}^{-1}R_{1:i-1,i}R_{1:i-1,i}^T R_{1:i-1,1:i-1}^{-1} & -\frac{1}{k}R_{1:i-1,1:i-1}^{-1}R_{1:i-1,i} \\ -\frac{1}{k}R_{1:i-1,i}^T R_{1:i-1,1:i-1}^{-1} & \frac{1}{k} \end{pmatrix}
\tag{48}
$$

where $k = R_{ii} - R_{1:i-1,i}^T R_{1:i-1,1:i-1}^{-1} R_{1:i-1,i}$. Notice that k equals σ_i^2, which is available from the previous step. This way of updating the inverse of the covariance matrix has been commonly used in the signal processing community, and it was even introduced in Gaussian process regression by Csató & Opper (2002), Van Vaerenbergh et al. (2012), and Pérez-Cruz et al. (2013). Substituting from Eq. (46), we get

$$
Q_{i+1} = \begin{pmatrix} Q_i + \frac{\mathbf{b}_i\mathbf{b}_i^T}{\sigma_i^2} & -\frac{\mathbf{b}_i}{\sigma_i^2} \\ -\frac{\mathbf{b}_i^T}{\sigma_i^2} & \frac{1}{\sigma_i^2} \end{pmatrix}
\tag{49}
$$

We also get

$$
\mathbf{b}_{i+1} = Q_{i+1}R_{1:i,i+1}
\tag{50}
$$

$$
\sigma_{i+1}^2 = R_{i+1,i+1} - R_{1:i,i+1}^T \mathbf{b}_{i+1}
\tag{51}
$$

In summary, using these recursive formulas we can compute the moments for the conditional distribution using $O(N^2)$ instead of $O(N^3)$ operations, thus providing some computational savings.

5.6 Summary of the Algorithm

The algorithm turns out to be very simple, and can be coded easily. It is important to start with the training set, then proceed with the test set. So, basically we will rename the variables, such that $\mathbf{v}_{1:N}$ represents the training set, and $\mathbf{v}_{N+1:N+NTEST}$ represents the test set. Also, for convenience, the covariance matrix of (25) will be rearranged and will be made to include all test patterns, to become

$$
R = \begin{pmatrix} C'(I + \Sigma)C' & \Sigma_{X_*X_*}^T C' \\ C'\Sigma_{X_*X_*} & I + \Sigma_{X_*X_*} \end{pmatrix}
\tag{52}
$$

where X_* is the matrix of test patterns. As can be seen in the algorithm, the training set computations have to be done once, and need not be repeated for every test pattern, making the algorithm of incremental nature. The algorithm is described follows:

Algorithm GPC-MC

1. $i = 1$: Set $Q_2 = \frac{1}{R_{1,1}}, \sigma_1^2 = R_{1,1}$. Generate M points $v_1(m)$ according to $\mathcal{N}(v_1, 0, \sigma_1^2)$. Compute

$$
\hat{P}_1 = \frac{\#\left(v_1(m) \text{ s. t. } v_1(m) \geq 0\right)}{M}
\tag{53}
$$

where the latter expression means the fraction of points that are ≥ 0. Remove the points $v_1(m) < 0$. Sample by replacement from among the remaining points to keep the total number of points equal M. Rename the variables, so that $v_1(m)$ are the new kept points.

2. For $i = 2$ to N do the following:

   (a) Compute the matrices:

   $$\mathbf{b}_i = Q_i R_{1:i-1,i} \tag{54}$$

   $$\sigma_i^2 = R_{i,i} - R_{1:i-1,i}^T \mathbf{b}_i \tag{55}$$

   $$Q_{i+1} = \begin{pmatrix} Q_i + \frac{\mathbf{b}_i \mathbf{b}_i^T}{\sigma_i^2} & -\frac{\mathbf{b}_i}{\sigma_i^2} \\ -\frac{\mathbf{b}_i^T}{\sigma_i^2} & \frac{1}{\sigma_i^2} \end{pmatrix} \tag{56}$$

   (b) Generate M points $v_i(m)$ according to $\mathcal{N}(v_i, \mathbf{b}_i^T \mathbf{v}_{1:i-1}(m), \sigma_i^2)$, $m = 1, \ldots, M$. Compute

   $$\hat{P}_i = \frac{\#\left(v_i(m) \text{ s. t. } v_i(m) \geq 0\right)}{M} \tag{57}$$

   (c) Remove the points $v_i(m) < 0$. In their place, sample by replacement from among the remaining points to keep the total number of points equal M. Rename the variables, so that $v_i(m)$ are the new kept points.

3. The log marginal likelihood function is given by the following sum over the training set probabilities:

   $$LogL = \sum_{i=1}^{N} \log\left(\hat{P}_i\right) \tag{58}$$

4. For $i = N + 1$ to $N + NTEST$ (the test patterns) do the following:

   (a) Compute the matrices:

   $$\mathbf{b}_i = Q_{N+1} R_{1:N,i} \tag{59}$$

   where Q_{N+1} represents the covariance matrix inverse, obtained at the last training pattern. It will not be be updated further during the test.

   $$\sigma_i^2 = R_{i,i} - R_{1:N,i}^T \mathbf{b}_i \tag{60}$$

   (b) Generate M points $v_i(m)$ according to $\mathcal{N}(v_i, \mathbf{b}_i^T \mathbf{v}_{1:N}(m), \sigma_i^2)$, $m = 1, \ldots, M$. Compute

   $$\hat{P}_i = \frac{\#\left(v_i(m) \text{ s. t. } v_i(m) \geq 0\right)}{M} \tag{61}$$

   Note that $\hat{P}_i$ is the sought *test pattern posterior probability*. Note also that we do not need to perform the bootstrap sampling step here for the test.

Note that the proposed algorithm, after it is applied to the training set, has its samples obey the posterior distribution. So these samples can be saved for any future evaluation of a test pattern.

5.7 Another Variant

A possibly more efficient modification is to have some kind of soft count, instead of the hard count used in Eq.(57) (or Eq. (33)). We know that each point, while moving from Step $i - 1$ to Step i, is generated from a Gaussian density. The probability of landing in the positive side can be computed by simply applying the cumulative Gaussian integral for each point (i. e. $\sigma(u) \equiv \int_{-\infty}^{u} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx$). In that case, instead of Eq. (57) we apply the following:

$$\hat{P}_i = \frac{\sum_{m=1}^{M} \sigma\left(\frac{\mathbf{b}_i^T \mathbf{v}_{1:i-1}(m)}{\sigma_i}\right)}{M} \tag{62}$$

This applies similarly to Eq. (61). Of course, this does not relieve us from having to generate the points. We have to do that while moving forward till reaching the last dimension. In essence, all other steps are similar to the original version of the algorithm. Only the count is different.

6 Simulation Experiments

6.1 Experiment 1: Testing the New Monte Carlo Integration Method

In this experiment we test the convergence properties of the new Monte Carlo multivariate Gaussian integration approach. The goal here is to test the efficacy of this new method irrespective of its use in Gaussian process classifiers. The application of this method to the Gaussian process classifier hinges mainly on its success as a stand-alone integration approach. Once we establish this fact, we will have assurances that it would work well in the Gaussian process classifier setting.

To be able to judge the new method's approximation error, we have to use examples where the "ground truth", i.e. the real integral value, is known. We identified a special form where this can be obtained. The experiments will be performed on this special form, described below.

The covariance matrix equals 1 on the diagonal and equals $d_i d_j$ off the diagonal (at the $(i,j)^{th}$ position), where $\mathbf{d} = (d_1, \ldots, d_N)^T$ is some vector with $|d_i| < 1$. In such a situation, the orthant probability can be reduced to a simple one-dimensional integration, as follows:

$$\int_0^{\infty} \mathcal{N}(\mathbf{v}, \mu, \Sigma) d\mathbf{v} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} \prod_{i=1}^{N} \sigma\left(\frac{d_i u}{\sqrt{1 - d_i^2}}\right) du \tag{63}$$

where σ is the cumulative Gaussian function (i.e. the one-dimensional integration of the Gaussian density function). This formula was proposed by Das (1956), Dunnet & Sobel (1955), and Ihm (1959). It has been also generalized to different forms by Marsaglia (1963) and Webster (1970), and also used in combination of Monte Carlo sampling by Breslaw (1994). Some special cases of this formula even yield some closed-form solutions.

In this experiment we considered different dimensions for our space. Specifically, we considered the dimensions $N = 50$, $N = 200$, and $N = 500$. In addition, for each dimension we considered 50 different problems, where each problem has a different $\mathbf{d}$ vector (whose components are generated from a uniform distribution in $[-1, 1]$). To evaluate how the approximation error varies with the number of Monte Carlo samples M, we ran each of these problems for various values of M. Because the value of the integral is usually an infinitessimal value, a sensible approach is to consider the logarithm of the estimated integral, and compare it to the logarithm of the true integral. For example a typical integral value for a 500-dimensional problem could be 10^{-200}. The logarithm becomes about -461. As an error measure, we used the following *mean*

Table 1: The Mean Absolute Percentage Error (MAPE, in %) of the Log Multivariate Gaussian Integral Estimate (and its Standard Error in Brackets) against the Dimension of the Problem and the Number of Monte Carlo Samples (the Numbers are in *Percent* so they are Multiplied by 100)

No. MC samples	Problem 1 ($N = 50$)	Problem 2 ($N = 200$)	Problem 3 ($N = 500$)
3,000,000	0.012 (0.0011)	0.015 (0.0065)	0.043 (0.0299)
1,000,000	0.016 (0.0020)	0.019 (0.0068)	0.045 (0.0298)
300,000	0.041 (0.0050)	0.032 (0.0072)	0.050 (0.0297)
100,000	0.072 (0.0094)	0.039 (0.0070)	0.059 (0.0303)
30,000	0.141 (0.0160)	0.078 (0.0065)	0.080 (0.0289)
10,000	0.245 (0.0268)	0.101 (0.0093)	0.107 (0.0322)

absolute percentage error, defined as:

$$MAPE = \frac{100}{NR} \sum_{i=1}^{NR} \frac{|\log(I) - \log(\hat{I})|}{|\log(I)|} \tag{64}$$

where I is the true integral value, $\hat{I}$ is the integral value estimated by the algorithm, and NR represents the number of runs (i.e. the number of different **d** vectors tested, in our case $NR = 50$).

Note that we have to be careful when evaluating the true integral numerically using Eq. (63). If we multiply the terms first and then integrate numerically, we end up with very small numbers, leading to a large error. We overcame this difficulty, by successive normalization by the maximum value after each multiplication. Then we evaluate the integral and multiply back the normalization terms that we divided by.

Table 1 shows the MAPE error measure (average over each of the 50 tested problems) for each of the tested values of N (dimension) and M (number of Monte Carlo runs). Note that these are *percent* errors, so they are multiplied by 100. Displayed in the table is also the standard error (over the 50 tested problems). One can observe that the developed algorithm evaluates the orthant probabilities with good accuracy. As expected the accuracy tends to improve for larger M. However, the relation between the accuracy and the dimension is less straightforward to describe. Even though by Eqs. (43) and (44) one might expect that for large N there will be more terms and hence a higher error, in practice the P_i's are more important influencing factors. One can also see that the algorithm succeeded for even the case of 500 dimensional problems, even though such high dimensions are quite formidable problems. Most of the algorithms for orthant probability estimation test on problems with only tens of dimension. e Note that because of memory limitations, in case of a large number M of Monte Carlo samples it may not be practical to propagate all samples together. A more practical approach is to rerun the problem several times, each with a smaller M. For example assume that we would like to use 2,000,000 samples. In that case we apply ten runs, each with $M = 200,000$. We then average the integral estimates obtained.

6.2 Experiment 2: Testing the New Monte Carlo Method on Gaussian Process Classification Problems

The next group of experiments aims to verify that the proposed Monte Carlo method, in a Gaussian process classification setting, does converge to the true solution. The problem we face is that in general there is no way to know the true solution, and so it could be hard to verify this claim. However, we identified a special group of problems where the "ground truth" could be obtained. This is if we take the distance kernel function to be of the dot product form. This

means that the covariance matrix equals

$$\Sigma = X X^T \tag{65}$$

This is the so-called linear kernel. It is a legitimate kernel function as it represents a similarity between the patterns, and is positive semidefinite. For single-feature classification problems it can be shown after a few of lines of derivation that the class 1 posterior probability of a pattern, as given by Eq. (28), becomes

$$J_* = \frac{\int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} \sigma(xu) \prod_{i=1}^{N} \sigma(y_i x_i u)\, du}{\int_{-\infty}^{\infty} e^{-\frac{u^2}{2}} \prod_{i=1}^{N} \sigma(y_i x_i u)\, du} \tag{66}$$

where x_i and x are the feature values of respectively the ith training pattern and the test pattern, and y_i denotes the class membership for pattern i. The marginal likelihood is simply the denominator in Eq. (66). In this group of problems the covariance function becomes of the form discussed in the Experiment 1, where we can make use of the one-dimensional integration method of Eq, (63) to evaluate the integrals. The fact that we are dealing with a one-dimensional feature space does not necessarily make the problem any easier. We are still dealing with the same formulas and with the same very high-dimensional integrals. The dimension of the feature vector impacts only the covariance matrix, it will just have different entries.

We generated a number of training and testing patterns from a one-dimensional (i.e. single-feature) two-class Gaussian problem. To ensure that the proposed model can handle different types of problems, we considered a variety of training/testing set sizes, a variety of means/variances for the class-conditional densities (in order to account for a variety of different class overlaps). Specifically, we considered the four problems shown below. Let N and $NTEST$ be the sizes of respectively the training set and test set, and let μ_i and σ_i be respectively the mean and standard deviation of the class conditional density for class i.

- Problem 1: $N = 100$, $NTEST = 50$, $\mu_1 = 0$, $\mu_2 = 1$, $\sigma_1 = 0.2$, $\sigma_2 = 0.3$.

- Problem 2: $N = 200$, $NTEST = 100$, $\mu_1 = 0$, $\mu_2 = 1$, $\sigma_1 = 2$, $\sigma_2 = 1$.

- Problem 3: $N = 400$, $NTEST = 200$, $\mu_1 = 0$, $\mu_2 = 1.5$, $\sigma_1 = 0.5$, $\sigma_2 = 0.75$.

- Problem 4: $N = 800$, $NTEST = 400$, $\mu_1 = 0$, $\mu_2 = 1$, $\sigma_1 = 1$, $\sigma_2 = 0.75$.

When constructing the covariance matrix, we made a point to shuffle the training patterns of both classes. This is *important* for achieving better accuracies/speeds. The reason will be mentioned at the end of this subsection. To obtain statistically more reliable numbers, from each of the above problems we applied the proposed algorithm a number $N_R \equiv 20$ different times. Since the estimated class 1 posterior probabilities of the different test patterns are in a well-known range from 0 to 1, it is sufficient to use an absolute error metric, so we used the mean absolute error (MAE), defined as follows:

$$MAE = \frac{1}{NTEST} \sum_{j=1}^{NTEST} |J_{*j} - \hat{J}_{*j}| \tag{67}$$

where J_{*j} is an evaluation of the class 1 posterior probability for test pattern j using an exact numerical integration procedure (the true value), obtained by the formula of Eq. (66), and $\hat{J}_{*j}$ is the estimate using the proposed Monte Carlo procedure. Table 2 shows the obtained MAE values, averaged over the 20 runs, for a variety of numbers of Monte Carlo samples.

Concerning the log marginal likelihood, we evaluated it using the proposed Monte Carlo algorithm, and compared it with the true value, obtained numerically by evaluating the denominator of Eq. (66). Since the log marginal likelihood can take any level, a normalized error

measure is more appropriate. So we used the mean absolute percent error (MAPE) measure. The formula is similar to Eq. (64), but with the appropriate comparison variables replaced. Table 3 shows the obtained MAPE (%) values for a variety of numbers of Monte Carlo samples for the log marginal likelihood estimation problem.

As seen from both tables, the algorithm is able to achieve a low error for both, the probability evaluation and the marginal likelihood. One can also see that increasing the number of Monte Carlo samples M leads to better accuracy. As mentioned in the last experiment, the relation between accuracy and dimension is less straightforward to describe. It is influenced more by the specific covariance matrix and the resulting conditional probabilities P_i. By observing Eqs. (43) and (44), one finds that a small P_i can lead to large error. It is therefore advantageous to have the P_i's closer to the middle (in most cases it is around 0.5). To achieve that, it is important to shuffle the data of both classes, rather than list first the data for Class 1, followed by the data of Class 2. The latter will cause more extreme P_i's and therefore lead to less accuracy. Other than random shuffle, one could interleave the data of class 1 and class 2 in a repetitive way (e.g. class 1 pattern, then class 2, then class 1, then class 2, etc).

6.3 Experiment 3: Comparison between the New Monte Carlo Method and the MCMC Approach

In this and the next experiment we present a comparison of the proposed Monte Carlo algorithm with the Markov Chain Monte Carlo (MCMC) approach, its only peer. The MCMC is the only available method that can accurately compute the exact classification probabilities. All other methods give only approximations. There are several MCMC based models. In this experiment we compare between the proposed algorithm and the Hybrid Monte Carlo (HMC) Neal (1999), and the Elliptical Slice Sampler (ESS) by Murray et al. (2010). For both methods, we use the implementation written by Rasmussen & Nickisch (2010), which includes several enhancements of these two methods. For the proposed algorithm we used the variant with the soft count, described in Subsection 5.7.

We considered Problem 3 ($N = 400$) of Experiment 2 (with a linear kernel). As mentioned, these are the only type of problems where the ground truth is known. For the purpose of comparison the two main aspects of speed and accuracy are important. They are contradictory metrics, for example improving the accuracy (by having a larger Monte Carlo sample) will lead to more lengthy runs, and vice versa too. To be able to visualize simultaneously both of these aspects of the performance we have plotted both the CPU time against the logarithm of the MAE in Figure 1 (for the case of probability estimation) and against the logarithm of the MAPE in Figure 2 (for the case of marginal likelihood estimation). In each of the two figures every point corresponds to the average CPU time/average MAE (or MAPE) over ten runs for a particular Monte Carlo parameter. The Monte Carlo parameter for the proposed algorithm, and for the HMC and ESS algorithms is the number of Monte Carlo samples. For the HMC and ESS algorithms we kept the other parameters at their recommended values, as given in the implementation by Rasmussen & Nickisch (2010) (they are any way much less influential than the number of samples). The parameters are fixed as follows: the number of of skipped samples is 40, the number of burn-in samples is 10, and the number of runs to remove finite temperature bias is 3.

To be able to judge the advantage of one algorithm versus another, one should examine the difference in accuracy for the same run time, or similarly the difference in run time for the same accuracy. One can see from the graphs that the proposed algorithm generally beats the HMC algorithm. The margin of outperformance is considerable, especially for the marginal likelihood case. The ESS ties with the the proposed algorithm for pattern probability estimation for low to moderate accuracy targets, but ESS outperforms for high accuracy computationally expensive runs. On the other hand, the proposed algorithm outperforms ESS considerably for the marginal likelihood case.

Table 2: The Mean Absolute Error (MAE) (averaged over the 20 runs) of the Gaussian Process Classification of Experiment 2 for the Four Different Problems against the Number of Monte Carlo Samples (the Standard Error is in Brackets).

No MC Samples	Prob 1 ($N = 100$)	Prob 2 ($N = 200$)	Prob 3 ($N = 400$)	Prob 4 ($N = 800$)
3,000,000	0.00016 (0.00004)	0.00024 (0.00005)	0.00022 (0.00005)	0.00022 (0.00006)
1,000,000	0.00031 (0.00007)	0.00043 (0.00010)	0.00038 (0.00009)	0.00045 (0.00012)
300,000	0.00056 (0.00013)	0.00081 (0.00018)	0.00062 (0.00014)	0.00075 (0.00019)
100,000	0.00095 (0.00021)	0.00139 (0.00031)	0.00112 (0.00025)	0.00128 (0.00033)
30,000	0.00160 (0.00036)	0.00297 (0.00066)	0.00212 (0.00047)	0.00235 (0.00061)
10,000	0.00308 (0.00069)	0.00463 (0.00103)	0.00391 (0.00088)	0.00443 (0.00114)

Table 3: The Mean Absolute Percent Error MAPE (in %, i.e. the Numbers are Multiplied by 100) of the Log Marginal Likelihood of Experiment 2 for the Four Different Problems against the Number of Monte Carlo Samples. All are Averages over the 20 Runs, and the Standard Error is in Brackets.

| No MC Samples	Prob 1 ($N = 100$)	Prob 2 ($N = 200$)	Prob 3 ($N = 400$)	Prob 4 ($N = 800$)
3,000,000	0.0081 (0.0018)	0.0088 (0.0020)	0.0063 (0.0014)	0.0033 (0.0009)
1,000,000	0.0187 (0.0042)	0.0097 (0.0022)	0.0095 (0.0021)	0.0059 (0.0015)
300,000	0.0364 (0.0081)	0.0255 (0.0057)	0.0130 (0.0029)	0.0077 (0.0020)
100,000	0.0671 (0.0150)	0.0340 (0.0076)	0.0238 (0.0053)	0.0130 (0.0034)
30,000	0.1170 (0.0262)	0.0629 (0.0141)	0.0450 (0.0101)	0.0249 (0.0064)
10,000	0.1522 (0.0340)	0.1334 (0.0298)	0.0900 (0.0201)	0.0622 (0.0161)

As pointed out before, the marginal likelihood is by far the most important of the two aspects. The reason is that it is evaluated numerous times in the process of tuning the hyperparameters of the kernel function, while the pattern probability estimation is performed only once. For example, from the figure the proposed algorithm produces a log(MAPE) of the marginal likelihood of about -4.52 with a CPU time of 525 sec. The ESS algorithm produces about the same log(MAPE) (or just a little better at -4.70) with CPU time of 5350 sec. With about a hundred application of an optimization algorithm (such as Rasmussen and Nickisch's GPML toolbox's `minimize` function Rasmussen & Nickisch (2010)), the new algorithm takes about 15 hours, while ESS takes about 148 hours.

## 6.4 Experiment 4: Comparison with the Other MCMC Methods on Synthetic and Real World Problems

The previous experiment, while providing an accurate benchmark comparison, considers only the linear kernel, which may not be very prevalent in real world applications. In this experiment we consider the more common RBF kernel, and also some synthetic and real world problems, in order to have a test as close as possible to realistic situations. We also add to the comparison the model developed by Titsias et al. (2009), in addition to the ESS and HMC algorithms considered last experiment. Titsias et al's algorithm is another MCMC-based algorithm for the GPC problem. It relies on using dynamically optimized control variables that provide a low dimensional representation of the function. The code is publicly available at Titsias (2010). It applies only to RBF and ARD kernels Titsias (2010), and that is why we did not include it in the comparison of the last experiment.

For the purpose of this comparison, there is however the problem of lack of ground truth,

because for RBF kernels we do not know the true value of the integrals that would yield the Gaussian process class probabilities. Nevertheless, we run the competing models on a number of problems, and check their convergence properties. If they converge to the same class probabilities, then this is a strong indication that these algorithms do indeed converge. Also, note that the considered RBF kernel is a more efficient and more widely used kernel than the linear kernel. So the experiments presented here are more relevant, as they fit more closely to the real experimental situations.
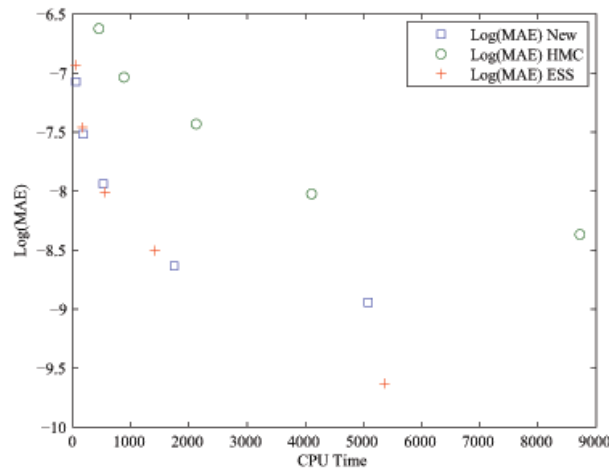
Figure 1: Log of the Mean Absolute Error (MAE) of the Probability Estimates of the New Algorithm, the HMC Algorithm, and the ESS Algorithm against the CPU Time (Seconds) of the Runs
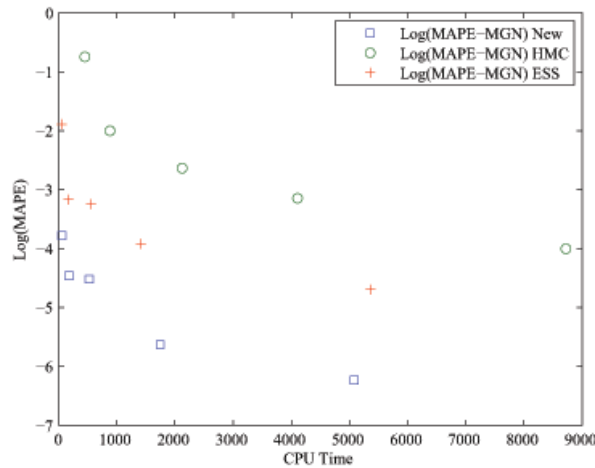
Figure 2: Log of the Mean Absolute Percentage Error (MAPE) of the Log Marginal Likelihood Estimates of the New Algorithm, the HMC Algorithm, and the ESS Algorithm against the CPU Time (Seconds) of the Runs

We considered artificially generated data using Gaussian class-conditional densities, and real data sets. For the artificial problems, using the Bayes classifier's formula, one can compute the true posterior probability $P(y_i = 1|x) \equiv P_i^{true}$ which the Gaussian process classifier attempts to model. However, we must emphasize that these true posterior probabilities need not be the same as those obtained by GPC, as GPC is based on a different formulation. If either proposed

or competing algorithms do a good job converging to the true value of the sought integral, but it turns out to be far from the true posterior, then it is not the fault of the algorithm. It should be attributed to the degree of validity of the Gaussian process formulation or to the finite sampled-ness of the training data. Nevertheless, a comparison with the true posterior provides a useful sanity check. We computed the mean absolute error between each competing method's estimated probabilities and P_i^{true} (let us denote them by MAE-POST(NEW), MAE-POST(TITSIAS), MAE-POST(ESS), and MAE-POST(HMC)). This measure applies only for synthetic problems, as for real problems we do not know the true posteriors. We also computed the following measure. For each pattern, we obtain the median of all four algorithms' estimated probability P_i^{med}. This so-called "consensus" value is compared against each algorithm's estimated probabilities. We get $\text{Mean}_i\left(|P_i - P_i^{med}|\right)$ for each method (denote these by MAE-MED(NEW), MAE-MED(TITSIAS), MAE-MED(ESS), and MAE-MED(HMC)). This measure will expose the aberrant algorithm that fails to converge, and is therefore a useful sanity check. We have computed a similar measure for the log marginal likelihood.

The problems considered are described as follows. Let d be the dimension of the feature vector, and let e_d denote the d-dimensional vector of all ones. Also, let:

$$\Sigma_0 = \left(\begin{array}{cc} 1 & 0,25 \\ 0,25 & 1 \end{array} \right) \tag{68}$$

and Σ_{10} is a 10×10 one-banded matrix with 0.5 on the diagonal and 0.2 on the upper and the lower bands. We considered the following synthetic problems, that provide a variety of different levels of training set sizes, class overlaps, and space dimensions, and also the following real world problems.

- Problem 1: $NTRAIN = 50$, $NTEST = 50$, $d = 2$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, e_2, \Sigma_0)$.

- Problem 2: $NTRAIN = 200$, $NTEST = 200$, $d = 2$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, e_2, \Sigma_0)$.

- Problem 3: $NTRAIN = 50$, $NTEST = 50$, $d = 2$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, 0.5e_2, \Sigma_0)$.

- Problem 4: $NTRAIN = 50$, $NTEST = 50$, $d = 2$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, 2e_2, \Sigma_0)$.

- Problem 5: $NTRAIN = 50$, $NTEST = 50$, $d = 10$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, e_{10}, \Sigma_{10})$.

- Problem 6: $NTRAIN = 1000$, $NTEST = 1000$, $d = 2$, $p(x|C_1) = \mathcal{N}(x, 0, I)$, $p(x|C_2) = \mathcal{N}(x, e_2, \Sigma_0)$.

- Problem 7: Crabs data, $NTRAIN = 100$, $NTEST = 100$, $d = 6$, available at http://www.stats.ox.ac.uk/

- Problem 8: Breast Cancer data, $NTRAIN = 200$, $NTEST = 249$, $d = 9$, available at http://mlearn.ics.uci.edu/databases/breast-cancer-wisconsin/.

- Problem 9: USPS 3 vs 5 data, $NTRAIN = 750$, $NTEST = 790$, $d = 256$, available at http://www.gaussianprocess.org/gpml/data/.

In the synthetic problems (1 to 6) we assume that the a priori probabilities are equal.

We ran six different runs on each of these problems, These six runs consider an RBF covariance function Σ, with the following parameters:

1. $\alpha = 5$, $\beta = 1$

Table 4: The Mean Absolute Error between the Algorithms' Probability Estimates and the Bayesian Posterior Probability (MAE-POST) and the Median of the Algorithms' Probabilities (MAE-MED) over All Hyperparameter Sets. Note that for the Real-World Problems 7, 8, and 9 MAE-POST is not Available

Problem	MAE-POST				MAE-MED			
	NEW	TITSIAS	ESS	HMC	NEW	TITSIAS	ESS	HMC
Problem 1	0.1177	0.1176	0.1177	0.1153	1.4×10^{-3}	1.7×10^{-3}	2.6×10^{-3}	8.1×10^{-3}
Problem 2	0.0668	0.0676	0.06711	0.0673	3.2×10^{-3}	2.6×10^{-3}	3.1×10^{-3}	8.8×10^{-3}
Problem 3	0.0822	0.0830	0.0824	0.0837	2.3×10^{-3}	4.7×10^{-3}	3.1×10^{-3}	1.7×10^{-2}
Problem 4	0.1637	0.1641	0.1632	0.1685	1.6×10^{-3}	1.9×10^{-3}	3.3×10^{-3}	1.8×10^{-2}
Problem 5	0.2884	0.2883	0.2890	0.2872	1.5×10^{-3}	2.0×10^{-3}	3.0×10^{-3}	1.2×10^{-2}
Problem 6	0.0382	0.0384	0.0359	0.0367	7.2×10^{-3}	1.1×10^{-2}	4.1×10^{-3}	8.6×10^{-3}
Problem 7	–	–	–	–	1.8×10^{-3}	2.1×10^{-3}	2.9×10^{-3}	1.3×10^{-2}
Problem 8	–	–	–	–	1.6×10^{-3}	1.9×10^{-3}	2.3×10^{-3}	9.7×10^{-3}
Problem 9	–	–	–	–	2.8×10^{-3}	–	4.1×10^{-3}	2.2×10^{-2}

2. $\alpha = 5, \quad \beta = 5$

3. $\alpha = 3, \quad \beta = 2$

4. $\alpha = 0.5, \quad \beta = 0.5$

5. $\alpha = 3, \quad \beta = 1$

6. $\alpha = 0.5, \quad \beta = 3$

For TITSIAS we considered 50,000 iterations, where we considered 5000 iterations for the burn-in, and a starting number of three control variables. For each problem we first ran the TITSIAS on the first α and β parameter set. We selected the number of Monte Carlo samples of the proposed method, the ESS method, and the HMC method so that it runs in about the same time as the TITSIAS (measured by CPU time). Then, we fixed this number for the other α and β parameter combination runs. Table 4 shows the average MAE-POST and MAE-MED for all nine problem (averaged over the six hyperparameter combintations).

From the runs we note the following observations:

- The runs of all methods lead to close probability estimates for all the test patterns and all the runs (typically an absolute error between the probability estimates and the median of probability estimates is of the order 10^{-3}). The exception is with HMC, which is a further from the other algorithms' estimates.

- All methods lead to very similar mean absolute error with respect to the Bayesian posterior error. This error is also fairly low, especially if the size of the training set is large. This is an indication that the source of the discrepancy is probably the finite-sampled-ness of the training set, rather than an inadequacy of the GPC model.

- The TITSIAS method was considerably slower for hyperparameter sets 4) and 6), and a little slower for hyperparameter sets 3) and 5). Even though we had fixed the number of iterations for all hyperparameter sets (at 50,000), it took about ten times as much to complete the run for sets 4) and 6) (compared to parameter sets 1) and 2) and compared to the other methods). It seems that lower values of α lead to much slower runs for the TITSIAS method. In contrast, the proposed Monte Carlo method yields similar speeds for all parameter sets. Also, the TITSIAS method did not converge for Problem 9 (the USPS problem). Even when reducing the number of samples to 10,000, it did not converge in more than 24 hours of a run.

- For most methods the runs take about a few minutes for a small problem (like $NTRAIN = NTEST = 50$), and about ten minutes for a medium problem (like $NTRAIN = NTEST = 200$). Of course, this is with the exception of the hyperparameter combinations that lead to slow runs for the TITSIAS method. For example for Problem 3 ($NTRAIN = NTEST = 200$) the proposed method with $M = 1,000,000$ took 12 minutes using Matlab on a computer featuring an Intel duo core I3 processor. Again, it is hard to perform an accurate speed comparison between the algorithms, because we do not know the ground truth.

- The marginal likelihood of the proposed Monte Carlo algorithm and the ESS algorithm are very close. However, the HMC algorithm produces a marginal likelihood that is somewhat different (often about 5 % different, and for Problem 9 about 100 % different from that of the other two algorithms). This indicates that it fell short of fully converging.

6.5 Comments on the Results

Working with general Bayesian methods often leads to high dimensional integrals. Evaluating such integrals can sometimes be frustrating because of the high dimensionality, and many Monte Carlo approaches fail (see Magdon-Ismail & Atiya (2002)). The advantage of the proposed appraoch is that it is very reliable. It basically works all the time, as we have not encountered a failing run. It is a fairly short algorithm, and is simple to code, so this will cut down on development time. The fact that it has no tuning parameter (other than the number of Monte Carlo samples) also facilitates applying the method and cuts down on time consuming tuning runs. The proposed algorithm has a main advantage compared to the other MCMC-based algorithms. It is considerably faster for the problem of evaluating the marginal likelihood. This is important because of its repeated evaluation during the hyperparameter tuning step, and this makes it a very time-consuming process. For this step it is sensible to use a smaller number M of Monte Carlo samples (for example 20,000 to 50,000). Exact evaluation of the marginal likelihood will not much impact the optimization outcome. It is evidenced by Table 2 and Table 3 that small size Monte Carlo samples achieve very low error for the log marginal likelihood (significantly lower than the error in the class posterior probabilities). Also, running small samples could possibly yield three or four best hyperparameter sets, for which on a closer look we rerun the method using a larger M to differentiate between them (a classic exploration versus exploitation problem). Once the optimal parameters are obtained, in the classification step we can then use a larger M (for example 200,000 or 500,000), because then, accuracy is important.

An interesting observation is that the impact of the dimension of the integral N on the accuracy of the proposed method is not that large. It seems that other factors weigh in more, such as the structure of the covariance matrix, and the resulting conditional probabilities. It is imperative to shuffle the class 1 and class 2 patterns, or interleave them in a regular way, before constructing the covariance matrix. This will lead to well-behaved conditional probabilities, and therefore better accuracy.

7 Conclusions

In this paper we derived a new formulation that simplifies the multi-integral formula for the Gaussian process classification (GPC) problem. The formulation, given in terms of the ratio of two multivariate Gaussian integrals, gives new insights, and potentially opens the door for better approximations.

We also developed a Monte Carlo method for the evaluation of multivariate Gaussian integrals. This allows us to obtain very close to exact evaluation of the GPC probabilities and the marginal likelihood function. The proposed method is simple, reliable, and fast. As such, it should be considered as a promising candidate for researchers to test, when attempting to obtain exact GPC probabilities.

Appendix

The matrix A_{22} can be written as

$$A_{22} = B + \frac{\mathbf{b}\mathbf{b}^T}{\sigma_*^2} \tag{69}$$

where

$$B = \begin{pmatrix} 1 & 0 \\ 0 & I + \Sigma^{-1} \end{pmatrix} \tag{70}$$

$$b = \begin{pmatrix} 1 \\ -a \end{pmatrix} \tag{71}$$

Using the small rank adjustment matrix inversion lemma Horn & Johnson (1985), we get

$$A_{22}^{-1} = B^{-1} - \frac{B^{-1}\mathbf{b}\mathbf{b}^T B^{-1}}{q} \tag{72}$$

where

$$q = \sigma_*^2 + \mathbf{b}^T B^{-1}\mathbf{b} \tag{73}$$

Substituting (72) into Eq. (24), we get

$$A = \begin{pmatrix} 0 & 0 \\ 0 & I - C'(I + \Sigma^{-1})^{-1}C' \end{pmatrix} + \frac{b'b'^T}{q} \tag{74}$$

where

$$b' = A_{12}B^{-1}b = \begin{pmatrix} -1 \\ +C'(I + \Sigma^{-1})^{-1}a \end{pmatrix} \tag{75}$$

$$= \begin{pmatrix} -1 \\ +C'(I + \Sigma)^{-1}\Sigma_{Xx_*} \end{pmatrix} \tag{76}$$

(The last equality follows from substituting for the variable $\mathbf{a}$ from Eq. (10) (i.e. $\mathbf{a} = \Sigma^{-1}\Sigma_{Xx_*}$), and teleporting the resulting Σ^{-1} into the bracketed expression $(I + \Sigma^{-1})^{-1}$.) The variable q can be simplified as follows:

$$q = \sigma_*^2 + \mathbf{b}^T \begin{pmatrix} 1 & 0 \\ 0 & (I + \Sigma^{-1})^{-1} \end{pmatrix} b \tag{77}$$

$$= \sigma_*^2 + 1 + \mathbf{a}^T(I + \Sigma^{-1})^{-1}\mathbf{a} \tag{78}$$

$$= 1 + \Sigma_{x_*x_*} - \Sigma_{Xx_*}^T(I + \Sigma)^{-1}\Sigma_{Xx_*} \tag{79}$$

The last equation follows from the definition of $\mathbf{a}$ (Eq. 10), and the definition of σ_*^2 Eq. (11), and several steps of simplification.

The first matrix in the RHS of Eq. (74) can be simplified further, by noting that

$$I - C'(I + \Sigma^{-1})^{-1}C' = C'\Big[I - (I + \Sigma^{-1})^{-1}\Big]C' \tag{80}$$

$$= C'(I + \Sigma^{-1})^{-1}\Big[(I + \Sigma^{-1}) - I\Big]C' \tag{81}$$

$$= C'(I + \Sigma)^{-1}C' \tag{82}$$

where we used the fact that $C'^2 = I$ because it is a diagonal matrix of 1's and -1's. We get the final formula for A, as follows:

$$A = \begin{pmatrix} \frac{1}{q} & -\frac{\Sigma_{Xx_*}^T(I+\Sigma)^{-1}C'}{q} \\ -\frac{C'(I+\Sigma)^{-1}\Sigma_{Xx_*}}{q} & C'(I + \Sigma)^{-1} + \frac{C'(I+\Sigma)^{-1}\Sigma_{Xx_*}\Sigma_{Xx_*}^T(I+\Sigma)^{-1}C'}{q} \end{pmatrix} \tag{83}$$

Let us construct the following matrix, which we will subsequently try to invert.

$$R = \begin{pmatrix} 1 + \Sigma_{x_*x_*} & \Sigma_{x_*x_*}^T C' \\ C'\Sigma_{x_*x_*} & C'(I+\Sigma)C' \end{pmatrix} \tag{84}$$

Using the partitioned matrix inverse theory Horn & Johnson (1985), we get

$$A = \begin{pmatrix} \frac{1}{q} & -\frac{\Sigma_{Xx_*}^T(I+\Sigma)^{-1}C'}{q} \\ -\frac{C'(I+\Sigma)^{-1}\Sigma_{Xx_*}}{q} & C'(I+\Sigma)^{-1} + \frac{C'(I+\Sigma)^{-1}\Sigma_{Xx_*}\Sigma_{Xx_*}^T(I+\Sigma)^{-1}C'}{q} \end{pmatrix} \tag{85}$$

which is the same as A in Eq. (83), and that completes the proof.

References

Abdel-Gawad, A.H., Atiya, A.F. (2008). A new accurate approximation for the Gaussian process classification problem. *Proceedings of the International Joint Conference on Neural Networks (IJCNN'08)*, Hong Kong, China.

Atiya, A.F., Metwally, S.A.K. (2005). Efficient estimation of first passage time density function for jump-diffusion processes", *SIAM Journal on Scientific Computing*, 26, 1760-1775.

Barber, D., Williams, C.K.I. (1997). Gaussian processes for Bayesian classification via hybrid Monte Carlo. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems* 9, Cambridge, MA, 1997. The MIT Press.

Batsch, F., Daneshkhah, A., Cheah, M., Kanarachos, S., Baxendale, A. (2019). Performance boundary identification for the evaluation of automated vehicles using Gaussian process classification. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 419-424.

Bazi Y., Melgani, F. (2010). Gaussian Process approach to remote sensing image classification. *IEEE Trans Geoscience and Remote Sensing*, 48(1), 186-197.

Breslaw, J.A. (1994). Evaluation of multivariate normal probability integrals using a low variance simulator. *The Review of Economics and Statistics*, 673-682.

Csató, L., Fokoué, E., Opper, M., Schottky, B., Winther, O. (2000). Efficient approaches to gaussian process classification. In *Advances in neural information processing systems* (pp. 251-257).

Csató, L., Opper, M. (2002). Sparse on-line Gaussian processes. *Neural Computation, 14*(3), 641-668.

Das, S.C. (1956, July). The numerical evaluation of a class of integrals. II. In *Mathematical Proceedings of the Cambridge Philosophical Society* (Vol. 52, No. 3, pp. 442-448). Cambridge University Press.

Deák, I. (1986). Computing probabilities of rectangles in case of multivariate normal distribution. *Journal of Statistics, Comput. Simul.*, 26, 101-114.

Dunnett, C.W., Sobel, M. (1955). Approximations to the probability integral and certain percentage points of a multivariate analogue of Student's t-distribution. B*iometrika, 42*(1/2), 258-260.

Elreedy, D., F Atiya, A., Shaheen, S. (2019). A Novel Active Learning Regression Framework for Balancing the Exploration-Exploitation Trade-Off. *Entropy, 21*(7), 651.

Eriksson, F. (1990). On the measure of solid angles. *Mathematics Magazine, 63*(3), 184-187.

Evans, M., Swartz, T. (2000). *Approximating Integrals via Monte Carlo and Deterministic Methods*, Oxford University Press, New York, USA.

Fayed, H.A., Atiya, A.F., Badawi, A.H. (2015a). Integrals Involving Normal PDF and CDF and Related Series. *Mathematical Sciences Letters, 4*(3), 249.

Fayed, H., Atiya, A., Badawi, A. (2015b). Fitting a rectangular function by Gaussians and application to the multivariate normal integrals. *Appl. Comput. Math, 14*(2), 194-203.

Fayed H., Atiya, A. (2014a). A novel series expansion for the multivariate normal probability integrals based on Fourier serie, *Mathematics of Computation*, 83, 2385-2402.

Fayed H., Atiya, A. (2014b). An evaluation of the integral of the product of the error function and the normal probability density, with application to the bivariate normal integral, *Mathematics of Computation*, 83, 235-250.

Genz, A. (1993), Comparison of methods for the computation of multivariate normal probabilities, *Computing Science and Statistics*, 25, 400-405.

Genz, A., Bretz, F. (2009). *Computation of multivariate normal and t probabilities* (Vol. 195). Springer Science & Business Media.

Gibbs, M.N., MacKay, D.J. (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks, 11*(6), 1458-1464.

Girolami, M., Rogers, S. (2006). Variational Bayesian multinomial probit regression with Gaussian process priors. *Neural Computation, 18*(8), 1790-1817.

Gupta, S. (1963a). Probability integrals of multivariate normal and multivariate t. *The Annals of Mathematical Statistics*, *34*(3), 792-828.

Gupta, S. (1963b). Bibliography of the multivariate normal integrals and related topics. *The Annals of Mathematical Statistics*, 34(3), 829-838.

Hajivassiliou, V., McFadden, D., Ruud, P. (1991). Simulation of multivariate normal orthant probabilities: Methods and programs. In *Yale University CT Working Paper*.

Hernández-Lobato, D., Hernández-Lobato, J.M., Dupont, P. (2011). Robust multi-class Gaussian process classification. In *Advances in neural information processing systems* (pp. 280-288).

Horn, R.A., Johnson, C.R. (1985). *Matrix Analysis*, Cambridge University Press, Cambridge, UK.

Ihm, P. (1959). Numerical evaluation of certain multivariate integrals. *Sankhya*, 21, 363-366.

Jaakkola, T.S., Haussler, D. (2020, August). Probabilistic kernel regression models. In D. Heckerman and J. Whittaker, editors, *Workshop on Artificial Intelligence and Statistics 7*, Morgan Kaufmann, 1999.

Jenssen, R., Erdogmus, D., Principe, J.C., Eltoft, T. (2007). The laplacian classifier. *IEEE Transactions on Signal Processing, 55*(7), 3262-3271.

Johnson, N.L., Kotz, S., Balakrishnan, N. (1994). *Continuous Univariate Distributions, vol. 1*, New York: Wiley.

Kendall, M.G. (1941). Proof of relations connected with the tetrachoric series and its generalization, *Biometrika*, 32, 196-198.

Kim, H.C., Ghahramani, Z. (2006). Bayesian Gaussian process classification with the EM-EP algorithm. *IEEE Trans Pattern Anal Mach Intelligence*, 28(12), 1948-59.

Kotz, S., Balakrishnan, N., Johnson, N.H. (2000). *Continuous Multivariate Distributions, Volume 1, Models and Applications*, John Wiley & Sons, New York.

Kuss, M., Rasmussen, C.E. (2005). Assessing approximate inference for binary Gaussian process classification, *Journal of Machine Learning Research*, 6, 1679-1704.

Magdon-Ismail, M., Atiya, A. (2002). Density estimation and random variate generation using multilayer networks. *IEEE transactions on neural networks, 13*(3), 497-520.

Marsaglia, G. (1963). Expressing the normal distribution with covariance matrix A + B in terms of one with covariance matrix A. *Biometrika*, 50, 535-538.

Minka, T. (2001). *A Family of Algorithms for Approximate Bayesian Inference*, Ph.D. thesis, MIT.

Moran, P.A.P. (1948). Rank correlation and product-moment correlation. *Biometrika, 35*(1/2), 203-206.

Murray, I., Adams R.P., MacKay, D.J.C. (2010). Elliptical slice sampling, *Journal of Machine Learning Research W&CP*, 9, 541-548.

Neal, R. (1998). Annealed importance sampling, Technical Report No. 9805, Department of Statistics, University of Toronto.

Neal, R. (1999). Regression and classification using Gaussian process priors. In J. M. Bernardo, J. O. Berger, A. P. David, and A. F. M. Smith, Eds., *Bayesian Statistics 6*, pp. 475-501, Oxford University Press.

Nickisch, H., Rasmussen, C.E. (2008). Approximations for binary Gaussian process classification. *Journal of Machine Learning Research, 9*(Oct), 2035-2078.

Nickisch, H., Rasmussen. C.E. (2010). Gaussian mixture modeling with Gaussian process latent variable models. *Pattern Recognition*, Springer Berlin Heidelberg, 272-282.

Opper, M., Winther, O. (2000). Gaussian processes for classification: Mean-field algorithms. *Neural computation, 12*(11), 2655-2684.

Pérez-Cruz, F., Vaerenbergh, S., Murillo-Fuentes, J.J., Lázaro-Gredilla, M., Santa Maria, I. (2013). Gaussian processes for nonlinear signal processing, *IEEE Signal Processing Magazine*, 30, 40-50.

Qi, Y., Abdel-Gawad, A.H., Minka, T.P. (2010). Sparse-posterior Gaussian processes for general likelihoods. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 450-457.

Rasmussen, C., Williams, C. (2005). *Gaussian Processes for Machine Learning,* MIT Press.

Rasmussen, C.E., Nickisch, H. (2010). Gaussian Processes for Machine Learning (GPML) Toolbox. *Journal of Machine Learning Research*, 11, 3011-3015.

Ribando, J. (2006). Measuring solid angles beyond dimension three. *Discrete & Computational Geometry*, 36, 479-487.

Rifkin, R., Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5, 101-141.

Schervish, M. (1984). Multivariate normal probabilities with error bound. *Applied Statistics*, 33, 81-87.

Schreiter, J., Nguyen-Tuong, D., Eberts, M., Bischoff, B., Markert, H., Toussaint, M. (2015). Safe exploration for active learning with Gaussian processes. In: Bifet A. et al. (eds) *Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2015*, Lecture Notes in Computer Science, 9286. Springer.

Seeger, M. (2004). Gaussian processes for machine learning. *International Journal of Neural Systems*, 14(2), 69-106.

Seeger M., Jordan, M. (2004). Sparse Gaussian process classification with multiple classes. Technical Report, Department of Statistics TR 661, University of California, Berkeley, CA.

Sundararajan, S., Keerthi, S.S. (2001). Predictive approaches for choosing hyperparameters in Gaussian processes. *Neural computation, 13*(5), 1103-1118.

Titsias, M. (2009, April). Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics* (pp. 567-574).

Titsias, M.K., Honkela, A., Lawrence, N.D., Rattray, M. (2012). Efficient sampling for Gaussian process inference using control variables, *Advances Advances in Neural Processing Systems 12*, MIT Press, 2009.

Titsias, M.K., Lawrence, N.D. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy.

Titsias, M.K. (2010). Gaussian process using control variables code: http://www.well.ox.ac.uk/ mtitsias/software.html.

Urtasun R., Darrell, T. (2007). Discriminative Gaussian process latent variable model for classification. *Proceedings of the 24th International Conference on Machine Learning (ICML'2007)*.

Van Vaerenbergh, S., Lázaro-Gredilla, M., Santamaría, I. (2012). Kernel recursive least-squares tracker for time-varying regression. *IEEE transactions on neural networks and learning systems, 23*(8), 1313-1326.

Vanhatalo, J.,Vehtari, A. (2010). Speeding up the binary Gaussian process classification. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'2010)*, Catalina Island, CA.

Vehtari, A., Särkkä S., Lampinen, J. (2000). On MCMC sampling in Bayesian MLP neural networks. *Proceedings of the 2000 International Joint Conference on Neural Networks*, Shun-Ichi Amari, C. Lee Giles, Marco Gori and Vincenzo Piuri, Eds, Vol. I, 317-322.

Webster, J.T. (1970). On the application of the method of Das in evaluating a multivariate normal integral. *Biometrika*, 57, 657-659.

Williams, C., Barber D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 1342-1351.