# The Informative Vector Machine: A Practical Probabilistic Alternative to the Support Vector Machine

Neil D. Lawrence neil@dcs.shef.ac.uk
Department of Computer Science,
University of Sheffield, Sheffield S1 4DP, United Kingdom
Matthias Seeger seeger@tuebingen.mpg.de
Max Planck Institute for Biological Cybernetics,
Spemannstrasse 38, 72076 Tübingen, Germany
Ralf Herbrich rherb@microsoft.com
Microsoft Research Ltd.,
7 J J Thomson Avenue, Cambridge CB3 0FB, United Kingdom

December 7, 2005

**Abstract**

We present a practical probabilistic alternative to the popular support vector machine (SVM). The algorithm is an approximation to a Gaussian process, and is probabilistic in the sense that it maintains the process variance that is implied by the use of a kernel function, which the SVM discards. We show that these variances may be tracked and made use of selection of an active set which gives a sparse representation for the model. For an active set size of $d$ our algorithm exhibits $O(d^2N)$ computational complexity and $O(dN)$ storage requirements. It has already been shown that the approach is comptetive with the SVM in terms of performance and running time, here we give more details of the approach and demonstrate that kernel parameters may also be learned in a practical and effective manner.

## 1 Introduction

Since the parallel introduction of Gaussian processes (Williams and Rasmussen, 1996) and support vector machines (Cortes and Vapnik, 1995) to the machine learning community the growth in the use of kernel methods has been dramatic. Traditionally the community had prefered non-linear parameteric models such as neural networks trained by backpropagation (Rumelhart et al., 1986). Some of the reasons for the rapid acceptance of the non-parametric methods include the reduced number of parameters in the models and concavity of the optimisation problems. Practitioners of Gaussian processes and support vector machines normally take two differing, but complementary, views of the kernel matrix. In support vector machines, and many other kernel algorithms, the kernel matrix is simply considered to be a dot product matrix where the vectors exist in some, perhaps infinite dimensional, space. Algorithms which take this view typically prove that some function of interest can be written in terms of a kernel expansion:

$$\mu\left(\mathbf{x}\right) = \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{k}\left(\mathbf{x}\right) + b\,, \tag{1}$$

where $b$ is a scalar offset, $\boldsymbol{\alpha}$ is a vector containing $N$ elements, $\mathbf{k}(\mathbf{x}) := [k(\mathbf{x}, \mathbf{x}_1) \dots k(\mathbf{x}, \mathbf{x}_N)]^{\mathrm{T}}$ is a vector evaluating the kernel function between a data-point $\mathbf{x}$ and all data-points in the training data $\mathbf{X} := [\mathbf{x}_1 \dots \mathbf{x}_N]^{\mathrm{T}}$, and $N$ is the number of data-points. The function itself can be non-linear in the input space, but it is linear in the feature space that is implicitely defined by the kernel.

In Gaussian processes (O'Hagan, 1992) the kernel function is a treated as a covariance function, where the covariance is from a Gaussian prior over latent functions $f$. For particular training data $\mathbf{X}$, the prior is often assumed to be zero mean,

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

where $\mathbf{f} := [f(\mathbf{x}_1) \dots f(\mathbf{x}_N)]^{\mathrm{T}}$ contains the $N$ values of the function $f$ at the training points $\mathbf{x}_i$ and $K_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$. Typically a posterior distribution is sought which, while not always Gaussian, will be approximated by a Gaussian process of the form

$$p(\mathbf{f}|\mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\mathbf{y} := [y_1 \dots y_N]^{\mathrm{T}}$ is a vector of $N$ data labels, $\boldsymbol{\mu} := [\mu_1 \dots \mu_N]^{\mathrm{T}}$ is a vector of $N$ posterior mean values and $\boldsymbol{\Sigma}$ is the posterior covariance kernel. Note that the mean values of this distribution will be computed by (1) applied to the training data-points $\mathbf{x}_i$. It is therefore natural to talk about a *mean function* and we now have a new *covariance function* associated with the posterior. The equivalence between the functional form of the mean function and (1) shows the strong relationship between the two perspectives. However when the dot product perspective is taken the posterior covariance function does not arise.

In practice the former viewpoint is often taken because of the deceptive simplicity of the notion of fitting linear algorithms in high dimensional feature spaces. Furthermore, in the Gaussian process approach the posterior covariance function is accounted for and must be propagated through the analysis. However, this propagation can only rarely be done exactly so ignoring the existence of the posterior covariance typically leads to simpler algorithms. We consider these algorithms to be non-probabilistic, because the kernel function is considered to be a deterministic function rather than a probabilistic process.

The aim of this paper is to emphasise the value associated with tracking this covariance, thereby maintaining the probabilistic interpretation of the algorithm. We aim is to show that in the classical machine learning domains of classification and regression these variances can be handled, through approximations, without a significant increase in algorithmic complexity. Indeed, we believe that the implementation of our approach is as simple as that of the popular SVM. The advantages of tracking these variances are numerous, but perhaps foremost amoungst them is the ability to learn the parameters of the kernel matrix through principled techniques such as maximum likelihood. The cornerstone of our approach, approximations inspired by the technique of assumed density filtering, have been developed for Gaussian processes (Csató and Opper, 2001; Csató, 2002; Minka, 2001). We combine these approximations with a powerful, heuristic, active set selection scheme to improve the speed of the algorithms. The resulting approach, which we refer to as the *informative vector machine* (IVM), is competetive with the support vector machine in performance and brings added value in terms of its ability to automatically determine the kernel parameters.

This paper is laid out as follows: In the next section we briefly review Gaussian processes and the approximation known as assumed density filtering (ADF). We illustrate how ADF is carried out for two very common noise models: the Gaussian distribution for regression and a probit based classification model. In Section 3 we address the computational problems that arise when implementing ADF in practice,

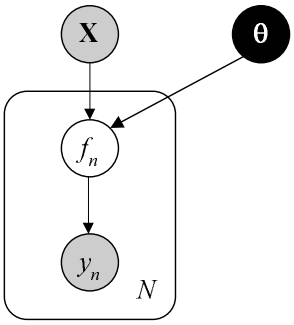


Figure 1: The Gaussian process model drawn graphically. We have made use of 'plate' notation to indicate the independence relationship between $\mathbf{f}$ and $\mathbf{y}$.

introducing the IVM as a solution. This section is followed by results on real world and toy data-sets and a brief discussion.

We use $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to denote a Gaussian density at $\mathbf{x}$ with a mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. When dealing one dimensional Gaussians the vectors and matrices are replaced by scalars. If $p$ is a density over $\mathbf{x}$, we will write $\langle g(\mathbf{x}) \rangle_{p(\mathbf{x})}$ as a shorthand notation for the expectation of $g$ over $\mathbf{x}$, $\int g(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x}$. If $\mathbf{A}$ is a matrix and $I$ and $J$ are index sets, then $\mathbf{A}_{I,J}$ denotes the submatrix obtained by jointly selecting all rows in $I$ and columns in $J$, that is, $\mathbf{A}_{I,J} \in \mathbb{R}^{|I| \times |J|}$. In this context, we use : to denote the set of all row/column indicies. For example, $\mathbf{A}_{:,j}$ is the $j$th column of $\mathbf{A}$.

## 2 Gaussian Processes

To start we will briefly review Gaussian process models; more detailed coverage is given in O'Hagan (1992) and Williams (1998). We will follow up by considering the approximating scheme mentioned in the previous section. Consider a simple latent variable model for data where the observations, $\mathbf{y} := [y_1 \dots y_N]^{\mathrm{T}}$, are independent from input data, $\mathbf{X} := [\mathbf{x}_1 \dots \mathbf{x}_N]^{\mathrm{T}}$, given a set of latent variables, $\mathbf{f} := [f(\mathbf{x}_1) \dots f(\mathbf{x}_N)]^{\mathrm{T}}$. The prior distribution for these latent variables is given by a Gaussian process,

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

with covariance function, or 'kernel', $\mathbf{K}$ which is parameterised by the vector $\boldsymbol{\theta}$ and evaluated at the points given in $\mathbf{X}$, that is, $K_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$. The joint likelihood of the data can be written as

$$p(\mathbf{y}, \mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) = p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) p(\mathbf{y}|\mathbf{f}) = p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) \prod_{n=1}^{N} p(y_n|f_n) \, , \tag{3}$$

where $p(y_n|f_n)$ gives the relationship between the latent variable and our observations and is sometimes referred to as the *noise model*. This relationship is shown graphically in Figure 1. For the moment, let us assume that the noise model takes the form of a Gaussian, $p(y_n|f_n) = \mathcal{N}(y_n; f_n, \beta_n^{-1})$, that is,

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}\left(\mathbf{y}; \mathbf{f}, \mathbf{B}^{-1}\right) \, , \tag{4}$$

where $\mathbf{B}$ is a diagonal matrix whose $n$th diagonal element is given by the precision $\beta_n$. These are the only assumptions made in Gaussian processes. Using Bayes' rule, we can now solve several problems.

**Learning the Kernel Parameters** In order to learn the parameter $\boldsymbol{\theta}$ of the kernel function we consider the marginalised likelihood of the data, $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$. This quantity can be obtained by integrating over $\mathbf{f}$ in (3). A straightforward calculdation shows that in the Gaussian case,

$$p\left(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}\right) = \int \mathcal{N}\left(\mathbf{f}; \mathbf{0}, \mathbf{K}\right) \mathcal{N}\left(\mathbf{y}; \mathbf{f}, \mathbf{B}^{-1}\right) d\mathbf{f} = \mathcal{N}\left(\mathbf{y}; \mathbf{0}, \mathbf{K} + \mathbf{B}^{-1}\right) . \tag{5}$$

**Learning to Predict** Given a fixed set of kernel parameters, $\boldsymbol{\theta}$, we are interested in the distribution of latent function values $f(\mathbf{x})$ at a new data-point $\mathbf{x}$. In order to compute this quantity, we start by considering the posterior distribution $p(\mathbf{f}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta})$ which is the fraction of (3) and (5),

$$p\left(\mathbf{f}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}\right) = \frac{p\left(\mathbf{y}, \mathbf{f}|\mathbf{X}, \boldsymbol{\theta}\right)}{p\left(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}\right)} = \mathcal{N}\left(\mathbf{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) , \quad \boldsymbol{\mu} = \boldsymbol{\Sigma}\mathbf{B}\mathbf{y} , \quad \boldsymbol{\Sigma} = \left(\mathbf{B} + \mathbf{K}^{-1}\right)^{-1} . \tag{6}$$

Moreover, by the prior assumption of (2) applied to $[\ \mathbf{f} \quad f(\mathbf{x})\ ]$ and standard properties of multivariate Gaussians we have that

$$p\left(f\left(\mathbf{x}\right)|\mathbf{f}, \mathbf{X}, \mathbf{x}, \boldsymbol{\theta}\right) = \mathcal{N}\left(f\left(\mathbf{x}\right); \mathbf{k}\left(\mathbf{x}\right)^{\mathrm{T}} \mathbf{K}^{-1}\mathbf{f}, k\left(\mathbf{x}, \mathbf{x}\right) - \mathbf{k}\left(\mathbf{x}\right)^{\mathrm{T}} \mathbf{K}^{-1}\mathbf{k}\left(\mathbf{x}\right)\right) ,$$

where we used the definitions of $\mathbf{K}$ and $\mathbf{k}(\mathbf{x})$ as defined in the previous section. Henceforth, the distribution of latent function value $f(\mathbf{x})$ at a new data-point $\mathbf{x}$ is obtained by integrating over the latent outputs at the training data, $\mathbf{f}$, and is Gaussian,

$$p\left(f\left(\mathbf{x}\right)|\mathbf{X}, \mathbf{y}, \mathbf{x}, \boldsymbol{\theta}\right) = \int p\left(f\left(\mathbf{x}\right)|\mathbf{f}, \mathbf{X}, \mathbf{x}, \boldsymbol{\theta}\right) p\left(\mathbf{f}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}\right) \, d\mathbf{f} = \mathcal{N}\left(f\left(\mathbf{x}\right); \mu\left(\mathbf{x}\right), \sigma^2\left(\mathbf{x}\right)\right) ,$$

where the *posterior mean function* $\mu(\mathbf{x})$ and the *posterior variance function* $\sigma^2(\mathbf{x})$ are given by

$$\mu\left(\mathbf{x}\right) = \mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}\boldsymbol{\Sigma}\mathbf{B}\mathbf{y} , \qquad \sigma^2\left(\mathbf{x}\right) = k\left(\mathbf{x}, \mathbf{x}\right) + \mathbf{k}^{\mathrm{T}}\mathbf{K}^{-1}\left(\boldsymbol{\Sigma} - \mathbf{K}\right)\mathbf{K}^{-1}\mathbf{k} .$$

Unfortunately, when the noise model is non-Gaussian, all these marginalisation are not possible. In the next section we shall review the assumed density filtering approximation which gives an approximation scheme that can deal with this problem.

## 2.1 The ADF Approximation

If for each data-point $(\mathbf{x}_n, y_n)$, the non-Gaussian noise, $p(y_n|f_n)$, as a function of $f_n$, can be sensibly approximated by a Gaussian distribution, $\mathcal{N}(m_n; f_n, \beta_n^{-1})$, by selection of appropriate values for $m_n$ and $\beta_n$, then we can make progress through an approach known as assumed-density filtering (ADF). ADF has its origins in online learning; it proceeds by incorporating one data-point at a time, computing the modified posterior (or an approximation to it) each time a point is incorporated. A thorough treatment of this approach is given by Minka (2001) and Csató (2002). Here we review the main points of relevance for our algorithm.

Given a *noise model*, $p(y_n|f_n)$, we note that the joint distribution $p(\mathbf{y}, \mathbf{f}|\mathbf{X}, \boldsymbol{\theta})$ factorises as shown in (3). If we view this as a function of $\mathbf{f}$ only (see also (6)), we may write

$$p\left(\mathbf{y}, \mathbf{f}|\mathbf{X}, \boldsymbol{\theta}\right) = \prod_{n=0}^{N} t_n\left(\mathbf{f}\right) ,$$

where we have defined $t_n(\mathbf{f}) := p(y_n|f_n)$ and in a slight abuse of notation we have taken $t_0(\mathbf{f}) := \mathcal{N}(\mathbf{f}; \mathbf{0}, \mathbf{K})$. Assumed density filtering takes advantage of this factorised form to build up an approximation, $q(\mathbf{f})$, to the true process posterior, $p(\mathbf{f}|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta})$. The factorisation of the joint posterior is exploited by building up this approximation in a sequential way so that after $i$ points are included we have an approximation $q_i(\mathbf{f})$. The starting point is to match the approximation to the prior process, *i.e.* $q_0(\mathbf{f}) := t_0(\mathbf{f}) = \mathcal{N}(\mathbf{f}; \mathbf{0}, \mathbf{K})$. The approximation is then constrained to always have this *functional* form.

As the data-point inclusion process is sequential we will maintain two index sets. The first, $I$, will be refered to as the active set and will represent the indices of those data-points that have been included in our approximation. The second, $J$, will be referred to as the inactive set and represents the indices of those data-points that have not yet been incorporated in our approximation. Initially $I$ is empty and $J = \{1 \dots N\}$. Thus, more formally the approximation $q_i$ is defined as

$$q_i(\mathbf{f}) \propto \mathcal{N}(\mathbf{f}; \mathbf{0}, \mathbf{K}) \mathcal{N}\left(\mathbf{m}_I; \mathbf{f}_I, \mathbf{B}_{I,I}^{-1}\right) \approx p(\mathbf{f}|\mathbf{X}_{I,:}, \mathbf{y}_I, \boldsymbol{\theta}) \ ,$$

where the $|I|$ many parameters $m_{n_i}$ and $\beta_{n_i}$ are sequentially chosen such that the approximation is as good as possible.

The approximation to the true posterior is built up by selecting a data-point index, $n_i$, from $J$. This point is included in the active set, $I$, leading to an updated posterior distribution of the form,

$$\hat{p}_i(\mathbf{f}) \propto q_{i-1}(\mathbf{f}) t_{n_i}(\mathbf{f}) \ . \tag{7}$$

Our new approximation, $q_i(\mathbf{f})$, is the found by minimising the Kullback-Leibler divergence between the two distributions,

$$\mathrm{KL}(\hat{p}_i||q_i) := \left\langle \log \frac{\hat{p}_i(\mathbf{f})}{q_i(\mathbf{f})} \right\rangle_{\hat{p}_i(\mathbf{f})} \ . \tag{8}$$

For the class of Gaussian distributions this minimsation leads to 'moment matching' equations (see Appendix A):

$$\boldsymbol{\mu}_i = \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})} , \tag{9}$$

$$\boldsymbol{\Sigma}_i = \left\langle \mathbf{f}\mathbf{f}^{\mathrm{T}} \right\rangle_{\hat{p}_i(\mathbf{f})} - \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})} \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})}^{\mathrm{T}} \ , \tag{10}$$

where $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are respectively the mean and covariance of the approximating distribution after the $i$th inclusion,

$$q_i(\mathbf{f}) := \mathcal{N}(\mathbf{f}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \ . \tag{11}$$

It turns out that our ability to compute (9) and (10) depends only on the tractability of the normalisation constant in (7),

$$Z_i := Z_i\left(\boldsymbol{\mu}_{i-1}, \boldsymbol{\Sigma}_{i-1}\right) := \int t_{n_i}(\mathbf{f}) q_{i-1}(\mathbf{f}) \, d\mathbf{f} \, .$$

As shown in Appendix A.2, the required updates for the mean and covariance in a form that applies regardless of our noise model (Csató, 2002; Minka, 2001; Seeger, 2004) are given by

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + \boldsymbol{\Sigma}_{i-1}\mathbf{g}_i \, , \tag{12}$$

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_{i-1} - \boldsymbol{\Sigma}_{i-1}\left(\mathbf{g}_i\mathbf{g}_i^{\mathrm{T}} - 2\mathbf{G}_i\right)\boldsymbol{\Sigma}_{i-1} \, , \tag{13}$$

where we have defined $\mathbf{g}_i := \nabla_{\boldsymbol{\mu}_{i-1}} \log(Z_i(\boldsymbol{\mu}_{i-1}, \boldsymbol{\Sigma}_{i-1}))$ and $\mathbf{G}_i := \nabla_{\boldsymbol{\Sigma}_{i-1}} \log(Z_i(\boldsymbol{\mu}_{i-1}, \boldsymbol{\Sigma}_{i-1}))$.

It is important to observe that each factor $t_n$ is acting only one a one-dimensional projection of $\mathbf{f}$, that is, $t_n(\mathbf{f}) = t_n(\mathbf{e}_n^{\mathrm{T}}\mathbf{f})$ where $\mathbf{e}_n$ is the $n$th unit vector. By the chain rule of differentiation the ADF updates on $\boldsymbol{\mu}_{i-1}$ and $\boldsymbol{\Sigma}_{i-1}$ can be written in terms of one-dimensional functions $Z_{n_i}$, $g_{n_i}$ and $G_{n_i}$,

$$
\begin{aligned}
Z_{n_i}\left(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) &= \int t_{n_i}(f)\, \mathcal{N}\left(f; \mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right), && (14)\\
\boldsymbol{\mu}_i &= \boldsymbol{\mu}_{i-1} + g_{n_i} \cdot \boldsymbol{\Sigma}_{i-1}\mathbf{e}_{n_i}, && (15)\\
\boldsymbol{\Sigma}_i &= \boldsymbol{\Sigma}_{i-1} - \underbrace{\left(g_{n_i}^2 - 2G_{n_i}\right)}_{\nu_{n_i}} \cdot \boldsymbol{\Sigma}_{i-1}\mathbf{e}_{n_i}\mathbf{e}_{n_i}^{\mathrm{T}}\boldsymbol{\Sigma}_{i-1}, && (16)
\end{aligned}
$$

where $\mu_{i-1,n_i}$ is the $n_i$th element of the vector $\boldsymbol{\mu}_{i-1}$ and $\varsigma_{i-1,n_i}$ is the $n_i$th diagonal element of $\boldsymbol{\Sigma}_{i-1}$, $g_{n_i} := \mathrm{d}\log(Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i})/\mathrm{d}\ \mu_{i-1,n_i}$ and $G_{n_i} := \mathrm{d}\log(Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i})/\mathrm{d}\ \varsigma_{i-1,n_i}$.

We can now consider a range of different noise models. Here we will review two of the most important, the probit noise model for classification, which we deal with in Section 2.3, and the standard Gaussian noise model for regression problems.

## 2.2 Gaussian Noise Model

One of the most widely used model fitting techniques is that of least squares. In the context of regression this is equivalent to fitting a regression model with Gaussian noise, the more general weighted least squares is equivalent to a Gaussian noise model where each data-point has a different associated inverse variance, $\beta_{n_i}$,

$$
p\left(y_{n_i}|f_{n_i}\right) = \mathcal{N}\left(y_{n_i}; f_{n_i}, \beta_{n_i}^{-1}\right).
$$

We need the normalisation constant (14). Since the noise model is Gaussian, this expectation may be evaluated as

$$
Z_{n_i}\left(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) = \mathcal{N}\left(y_{n_i}; \mu_{i-1,n_i}, \varsigma_{i-1,n_i} + \beta_{n_i}^{-1}\right)
$$

and the log normaliastion constant $\log(Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}))$ may be differentiated w.r.t. both parameters to find

$$
g_{n_i} = \frac{y_{n_i} - \mu_{i-1,n_i}}{\beta_{n_i}^{-1} + \varsigma_{i-1,n_i}}, \qquad G_{n_i} = -\frac{1}{2\left(\varsigma_{i-1,n_i} + \beta_{n_i}^{-1}\right)} + \frac{1}{2}g_{n_i}^2. \tag{17}
$$

This finally results in

$$
\nu_{n_i} = g_{n_i}^2 - 2G_{n_i} = \frac{1}{\varsigma_{i-1,n_i} + \beta_{n_i}^{-1}}. \tag{18}
$$

These updates involve no approximations as the true posterior process is in fact Gaussian: the updates are a scheme for on-line learning of Gaussian processes. In the next section we will consider the probit classification noise model where the updates lead to an approximation to the true posterior.

## 2.3 Probit Noise Model

Consider the *probit* noise model for modelling binary classification, $y_{n_i} \in \{-1, 1\}$,

$$
p\left(y_{n_i}|f_{n_i}\right) = \Phi\left(\lambda y_{n_i}\left(f_{n_i} + b\right)\right),
$$

where $\Phi(\cdot)$ is the cumulative Gaussian[1] the slope of which is controled by $\lambda$. The normalisation constant is again found by evaluating the expectation (14) :

$$Z_{n_i}\left(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) = \int \Phi\left(\lambda y_{n_i}\left(f_{n_i}+b\right)\right) \mathcal{N}\left(f_{n_i}; \mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) df_{n_i} .$$

Exchanging the order of integration and using standard results for the convolution of two normal densities we get the following normalisation constant $Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i})$,

$$\begin{aligned} Z_{n_i}\left(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) &= \int \mathbb{I}_{t\leq 0}\left[\int \mathcal{N}\left(t; -\lambda y_{n_i}\left(f_{n_i}+b\right), 1\right) \mathcal{N}\left(f_{n_i}; \mu_{i-1,n_i}, \varsigma_{i-1,n_i}\right) df_{n_i}\right] dt \\ &= \int \mathbb{I}_{t\leq 0}\, \mathcal{N}\left(t; -\lambda y_{n_i}\left(\mu_{i-1,n_i}+b\right), 1+\lambda^2 \varsigma_{i-1,n_i}\right) dt \\ &= \Phi\left(u_{i-1,n_i}\right) , \end{aligned}$$

where we have defined $u_{i-1,n_i}$ by

$$u_{i-1,n_i} := c_{i-1,n_i}\left(\mu_{i-1,n_i}+b\right) , \qquad c_{i-1,n_i} := \frac{y_{n_i}}{\sqrt{\lambda^{-2}+\varsigma_{i-1,n_i}}} . \tag{19}$$

Performing the derivatives of the log normalisation constant, $\log(Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i}))$ we obtain for $g_{n_i}$ and $G_{n_i}$ [2]

$$\begin{aligned} g_{n_i} &= \frac{c_{i-1,n_i}\mathcal{N}\left(u_{i-1,n_i}; 0, 1\right)}{\Phi\left(u_{i-1,n_i}\right)} , \qquad (20) \\ G_{n_i} &= -\frac{1}{2} g_{n_i} u_{i-1,n_i} c_{i-1,n_i} . \end{aligned}$$

This finally results in

$$\nu_{n_i} = g_{n_i}\left(g_{n_i} + u_{i-1,n_i} c_{i-1,n_i}\right) . \tag{21}$$

In practice we wish to summarise our Gausssian approximation $\mathcal{N}(m_{n_i}; f_{n_i}, \beta_{n_i}^{-1})$ to the likelihood in terms of parameters $m_{n_i}$ and $\beta_{n_i}$. This can be done by equating the values of $g_{n_i}$ and $\nu_{n_i}$ in (20) and (21) with those derived for the Gaussian above. Substitiuting the Gaussian target $y_{n_i}$ in (17) with the parameter $m_{n_i}$ and re-arranging (17) and (18) we obtain

$$\begin{aligned} m_{n_i} &= \frac{g_{n_i}}{\nu_{i,n_i}} + \mu_{i-1,n_i} , \qquad (22) \\ \beta_{n_i} &= \frac{\nu_{i,n_i}}{1-\nu_{i,n_i}\varsigma_{i-1,n_i}} . \qquad (23) \end{aligned}$$

Updates for $\boldsymbol{\mu}_{i-1} \rightarrow \boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_{i-1} \rightarrow \boldsymbol{\Sigma}_i$, the parameters of $q_i(\mathbf{f}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, are identical to those given in (15) and (16). Note from (19) that we can consider the slope, $\lambda$, of the noise model to be $\infty$ and account for this noise by adding a matrix $\lambda^{-2}\mathbf{I}$ to the kernel matrix which will in turn cause $\varsigma_{i-1,n_i}$ to increase by $\lambda^{-2}$ so the two approaches are equivalent.

## 2.4 Ordinal Categorical Model

It may be the case that we are presented with data which can be viewed as a discrete representation of a continuous space. We will consider the case where

[1] This function is defined by $\Phi(z) := \int \mathbb{I}_{t\leq z}\, \mathcal{N}(t; 0, 1)\, dt = \int \mathbb{I}_{t\leq 0}\, \mathcal{N}(t; -z, 1)\, dt$.

[2] In implementation care must be taken in computing $g_{n_i}$: when $u_{i-1,n_i}$ has large magnitude both $\Phi(u_{i-1,n_i})$ and $N(u_{i-1,n_i}; 0, 1)$ become very small and numerical precision issues arise.

a 1-D continuous space has been split into $C$ discrete regions, the probability of belonging to each region given by

$$p\left(y_{n_i}|f_{n_i}\right) = \begin{cases} \Phi\left(\lambda\left(b+\sum_{i=1}^{y_{n_i}} w_i - f_{n_i}\right)\right) - \Phi\left(\lambda\left(b+\sum_{i=1}^{y_{n_i}-1} w_i - f_{n_i}\right)\right) & \text{for } 0 < y_{n_i} < C-1 \\ \Phi\left(\lambda\left(f_{n_i} - b - \sum_{i=1}^{C-2} w_i\right)\right) & \text{for } y_{n_i} = C-1 \\ \Phi\left(\lambda\left(b+f_{n_i}\right)\right) & \text{for } y_{n_i} = 0 \end{cases},$$

where $y_{n_i}$ is an integer in the range $[0, C-1]$. The normalisation or partition function $Z_{n_i}(\mu_{i-1,n_i}, \varsigma_{i-1,n_i})$ is then given by

$$Z_i = \begin{cases} \Phi\left(u_{i-1,n_i}\right) - \Phi\left(u'_{i-1,n_i}\right) & \text{for } 0 < y_{n_i} < C-1 \\ \Phi\left(-u'_{i-1,n_i}\right) & \text{for } y_{n_i} = C-1 \\ \Phi\left(u_{i-1,n_i}\right) & \text{for } y_{n_i} = 0 \end{cases}$$

with the variables $u_{i-1,n_i}$, $u'_{i-1,n_i}$ given by

$$\begin{aligned} u_{i-1,n_i} &= c_{i-1,n_i}\left(b + \sum_{i=1}^{y_n} w_i - \mu_{i-1,n_i}\right), \\ u'_{i-1,n_i} &= c_{i-1,n_i}\left(b + \sum_{i=1}^{y_n-1} w_i - \mu_{i-1,n_i}\right), \\ c_{i-1,n_i} &= \frac{1}{\sqrt{\lambda^{-2} + \varsigma_{i-1,n_i}}}. \end{aligned}$$

Performing the necessary derivatives to obtain[3] $g_{n_i}$ and $\nu_{n_i}$ we have

$$g_{n_i} = \begin{cases} c_{i-1,n_i} \cdot \frac{\mathcal{N}\left(u_{i-1,n_i};0,1\right) - \mathcal{N}\left(u'_{i-1,n_i};0,1\right)}{\Phi\left(u_{i-1,n_i};0,1\right) - \Phi\left(u'_{i-1,n_i};0,1\right)} & \text{for } 0 < y_{n_i} < C-1 \\ - \; c_{i-1,n_i} \cdot \frac{\mathcal{N}\left(-u'_{i-1,n_i};0,1\right)}{\Phi\left(-u'_{i-1,n_i};0,1\right)} & \text{for } y_{n_i} = C-1 \\ c_{i-1,n_i} \cdot \frac{\mathcal{N}\left(u_{i-1,n_i};0,1\right)}{\Phi\left(u_{i-1,n_i};0,1\right)} & \text{for } y_{n_i} = 0 \end{cases}, \quad (24)$$

and

$$\nu_{n_i} = \begin{cases} g_{n_i}^2 + c_{i-1,n_i}^2 \cdot \frac{u_{i-1,n_i}\mathcal{N}\left(u_{i-1,n_i};0,1\right) - u'_{i-1,n_i}\mathcal{N}\left(u'_{i-1,n_i};0,1\right)}{\Phi\left(u_{i-1,n_i}\right) - \Phi\left(u'_{i-1,n_i}\right)} & \text{for } 0 < y_{n_i} < C-1 \\ g_{n_i}\left(g_{n_i} + c_{i-1,n_i} u_{i-1,n_i}\right) & \text{otherwise} \end{cases}. \quad (25)$$

As before we can summarise our approximation to the likelihood in terms of site means and precisions as in (5).

# 3 The Informative Vector Machine

The assumed-density filtering approach outlined in the last section assumes that all data-points will be made use of in determining the model. One problem with this is that including all $N$ data-points gives the algorithm $O(N^3)$ complexity. Even more of a concern is that if we wish to find the parameters, $\boldsymbol{\theta}$, of the kernel by gradient based optimisation of the approximation to the marginalised likelihood given by (5), each gradient evaluation will be $O(N^3)$. It is important to find a method for reducing this complexity. One way forward is to seek a sparse representation of the

[3] Care must be taken in computing $g_{n_i}$ when $u_{i-1,n_i}$ has large magnitude as both $\Phi(\cdot)$and $\mathcal{N}(\cdot)$ become small.

data-set. In Csató and Opper (2001) and Csató (2002) this is achieved by minimising a KL divergence between a sparse representation and the true process posterior. In this paper we propose an alternative approach known as the *informative vector machine* (see also Lawrence et al., 2003; Seeger, 2004). By only estimating a maximimum of $d$ site parameters we force the vectors $\mathbf{m}$ and $\boldsymbol{\beta}$ to be sparse. As a result we can reduce the computational requirements of the algorithm from $O(N^3)$ to $O(d^2N)$ and the storage requirements form $O(N^2)$ to $O(dN)$. In the informative vector machine (IVM) this 'sparsification' of the original Gaussian process is imposed by carefully selecting a sub-set of the data. The ADF algorithm allows us to select this sub-set in an online way; all we need is a data-point selection criterion. The IVM uses an information theoretic selection heuristic to pick the next data-point to include. In this way we greedily minimise the entropy of the posterior process.

## 3.1 Data-point Selection

In a nutshell, the IVM selects the next data-point according to the change in entropy, $H(q_i) := -\langle \log(q_i(\mathbf{f})) \rangle_{q_i(\mathbf{f})}$, of the posterior process after including this data-point. This can be seen as a measure of reduction in the level of uncertainty. Note that this can only be evaluated because we are propagating the posterior covariance function; a similar mearsure cannot be evaluated for the SVM. Recalling that the entropy of a Gaussian is $H(\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})) = \frac{N}{2}(1 + \log 2\pi) + \frac{1}{2}\log|\boldsymbol{\Sigma}|$ and using the general update equation (13) together with the sparse ADF update equation specified in (16), the entropy change is given by

$$\begin{aligned} \Delta H_{i,n_i} &= -\frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \frac{1}{2}\log|\boldsymbol{\Sigma}_{i-1}| = -\frac{1}{2}\log\left|\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_{i-1}^{-1}\right| \\ &= -\frac{1}{2}\log\left|\left(\mathbf{I} - \nu_{n_i}\cdot\boldsymbol{\Sigma}_{i-1}\mathbf{e}_{n_i}\mathbf{e}_{n_i}^{\mathrm{T}}\right)\right| \\ &= -\frac{1}{2}\log\left(1 - \nu_{i,n_i}\varsigma_{i-1,n_i}\right) . \end{aligned} \tag{26}$$

Other criteria (such as information gain) are also straightforward to compute. Such greedy selection criteria are inspired by information theory and have also been applied in the context of active learning (see also MacKay, 1991; Seung et al., 1992), however in active learning the label of the data-point is assumed to be unknown before selection.

## 3.2 Efficient Representation: The Informative Vector Machine Algorithm

We will now disucss the issue of efficient representation of the distributions $q_i$. To this end, we will need to consider the column based representation of the covariance, that is

$$\boldsymbol{\Sigma}_i = \left[\begin{array}{ccc} \mathbf{s}_{i,1} & \cdots & \mathbf{s}_{i,N} \end{array}\right] , \qquad \mathbf{s}_{i,k} := \boldsymbol{\Sigma}_i\mathbf{e}_k .$$

Looking at the data-point selection rule (26) and the general update equations (15) and (16) we note that need we only need to maintain the following structures:

**Posterior covariance (diagonal)** The diagonal $\boldsymbol{\varsigma}_i := \mathrm{diag}(\boldsymbol{\Sigma}_i)$ of the posterior covariance matrix $\boldsymbol{\Sigma}_i$ is necessary in order to be able to compute the score $\Delta H_{i,n_i}$ as well as the terms $g_{n_i}$ and $G_{n_i}$ which, in turn, are required to compute the update factors $\nu_{n_i}$. Applying the diag operator to (16) we have

$$\boldsymbol{\varsigma}_i = \boldsymbol{\varsigma}_{i-1} - \nu_{i,n_i}\mathrm{diag}\left(\mathbf{s}_{i-1,n_i}\mathbf{s}_{i-1,n_i}^{\mathrm{T}}\right) . \tag{27}$$

**Posterior mean** The vector $\boldsymbol{\mu}_i \in \mathbb{R}^N$ of the posterior mean is necessary to compute the terms $g_{n_i}$ and $G_{n_1}$ necessary to compute the update factors $\nu_{n_i}$ which are used both in the score (26) and update equations (15) and (16) . In fact, re-stating (15) we see that the vector $\boldsymbol{\mu}_i$ can be computed simply by

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + g_{n_i}\mathbf{s}_{i-1,n_i} \,. \tag{28}$$

**Posterior covariance (low rank)** In both of the above update equations (27) and (28) we needed the $n_i$th column of the covariance $\boldsymbol{\Sigma}_{i-1}$. However, by virtue of the update equation (16) this matrix has a particular structure, where successive outer products are added to the original prior covariance $\boldsymbol{\Sigma}_0 := \mathbf{K}$ to form the current covariance. This can be re-written as

$$\boldsymbol{\Sigma}_i = \mathbf{K} - \mathbf{M}_i^{\mathrm{T}}\mathbf{M}_i \,, \tag{29}$$

where the $k$th row of $\mathbf{M}_i \in \mathbb{R}^{i\times N}$ is given by $\sqrt{\nu_{k,n_k}}\mathbf{s}_{k-1,n_k}$ and $n_k$ represents $k$th included data-point. Recalling that $\mathbf{s}_{i-1,n_i}$ is the $n_i$th column of $\boldsymbol{\Sigma}_{i-1}$ we note that , if we are not storing $\boldsymbol{\Sigma}_{i-1}$ explicitely, we will not be able to represent $\mathbf{s}_{i-1,n_i}$ directly. However, by virtue of (29) this column can be recomputed from $\mathbf{M}_{i-1}$ and $\mathbf{K}$,

$$\mathbf{s}_{i-1,n_i} = \boldsymbol{\Sigma}_{i-1}\mathbf{e}_{n_i} = \mathbf{K}_{:,n_i} - \mathbf{M}_{i-1}^{\mathrm{T}}\mathbf{m}_{i-1,n_i} \,, \tag{30}$$

where, in this context, $\mathbf{m}_{i-1,n_i}$ is the $n_i$th column of $\mathbf{M}_{i-1}$.

- Computing $\mathbf{K}_{:,n_i}$ requires $N$ kernel computations between the data point $\mathbf{x}_{n_i}$ and all other data points.
- Computing $\mathbf{m}_{i-1,n_i} \in \mathbb{R}^{(i-1)\times 1}$ will require $O((i-1)N)$ operations as $\mathbf{M}_{i-1} \in \mathbb{R}^{(i-1)\times N}$.
- Computing $\mathbf{M}_i^{\mathrm{T}}\mathbf{m}_{i-1,n_i}$ will also require only $O((i-1)N)$ operations as $\mathbf{M}_{i-1} \in \mathbb{R}^{(i-1)\times N}$ and $\mathbf{m}_{i-1,n_i} \in \mathbb{R}^{(i-1)\times 1}$.

Note that the memory requirements are dominated by the last item. If we have $d$ data-points included, the memory requirements are $O(dN)$; maintaining the whole of covariance matrix $\boldsymbol{\Sigma}_{i-1}$ would require $O(N^2)$ storage, which is undesirable and unnecessary. An example of how these updates may be combined efficiently in practice is given in Algorithm 1 which also is a barebones description of the software we used in our experiments available from `http://www.dcs.shef.ac.uk/~neil/ivm`.

## 3.3 Kernel Parameter Updates

So far we have discussed how the posterior's mean and covariance functions can be updated in an on-line way given a fixed kernel. We suggested in the introduction that one of the main reasons we may wish to keep track of a representation of the posterior covariance is to so that we may learn the kernel parameters.

Refering to the graphical representation of our model in Figure 1 our objective is to marginalise the latent variable $\mathbf{f}$ and optimise the parameters of the kernel by maximising the resulting likelihood. This approach is sometimes known as type II maximum likelihood to reflect the fact that some marginalisation has been undertaken before we use maximum likelihood. In general it is not tractable to form the marginalised likelihood exactly, but the ADF framework we have described implies that we are approximating the true marginalised likelihood with a Gaussian. This true likelihood is given in (5) for the Gaussian case, but for the more general case

**Algorithm 1** The informative vector machine learning algorithm.

[1]
A number $d$ of active points.
Set $\boldsymbol{\varsigma}_0 = \text{diag}(\mathbf{K})$, $\boldsymbol{\mu}_0 = \mathbf{0}$, $\mathbf{M}_0$ an empty matrix, $J = \{1, \ldots, N\}$ and $I = \emptyset$.
$i \in \{1, \ldots, d\}$
$n \in J$
Compute $g_{n_i}$ (for example, using (17), (20) or (24)).
Compute $\nu_{n_i}$ (for example, using (18), (21) or (25)).
Compute $\Delta H_{i,n}$ according to (26).
*
$n_i = \text{argmax}_{n \in J} \Delta H_{i,n}$.
Update $m_{n_i}$ and $\beta_{n_i}$ (for example, using (22) and (23)) (not necessary for Gaussian noise).
Compute $\mathbf{s}_{i-1,n_i}$ using (30).
Compute $\boldsymbol{\varsigma}_i$ and $\boldsymbol{\mu}_i$ using (27) and (28).
Append the row $\sqrt{\nu_{i,n_i}}\mathbf{s}_{i-1,n_i}^{\mathrm{T}}$ to $\mathbf{M}_{i-1}$ to form $\mathbf{M}_i$.
$I \leftarrow I \cup \{n_i\}$, $J \leftarrow J \setminus \{n_i\}$.
*

where a vector of site means $\mathbf{m} \neq \mathbf{y}$ and site precisions $\boldsymbol{\beta}$ has been obtained using the ADF updates, the implied approximation is

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \approx N\left(\mathbf{m}; \mathbf{0}, \mathbf{K} + \mathbf{B}^{-1}\right), \tag{31}$$

where the dependence of the likelihood on $\mathbf{y}$ is indirect and through the site parameters $\mathbf{m}$ and $\boldsymbol{\beta}$. Computation of the site parameters was only possible because we were keeping track of the process variances at every stage. For a Gaussian noise model tracking of the site parameters is trivial, but for non-Gaussian noise models, such as the probit classification model discussed in Section 2.3, computation of the site parameters for the $i$th is dependent on (9) and (10) which themselves are dependent on the approximation to the posterior covariance after $(i-1)$th point inclusion. In other words our approximation to the marginalised likelihood is only possible because we have kept track of the posterior covariances.

The ADF based IVM algorithm described in Algorithm 1 leads to sparse vectors $\mathbf{m}$ and $\boldsymbol{\beta}$ each with $d$ non-zero elements (see Line 9 in Algorithm 1). Maximising with these sparse vectors of site parameters implies a further approximation to that given in (31) where we only consider the likelihood of points in the active set,

$$p(\mathbf{y}_I|\mathbf{X}, \boldsymbol{\theta}) \approx N\left(\mathbf{m}_I; \mathbf{0}, \mathbf{K}_I + \mathbf{B}_I^{-1}\right), \tag{32}$$

where $\mathbf{y}_I$ is a vector containing only those elements of $\mathbf{y}$ that are in the active set. Note that the dependence of the approximation on the kernel parameters $\boldsymbol{\theta}$ occurs through the matrix $\mathbf{K}_I$.

Given the active set, $I$, and the site parameters, $\mathbf{m}$ and $\boldsymbol{\beta}$, we can optimise our approximation with respect to the kernel parameters by using a non-linear optimiser such as scaled conjugate gradients. Note that in practice the quality of the active set will depend on the kernel parameter selection as will the optimal site parameters. We can certainly imagine more complex schema for optimising the kernel parameters which take account of these dependencies in a better way, some examples of which are given in Seeger (2004), but for our experiments we found this simple approach to be effective.

**Algorithm 2** The IVM algorithm for parameter optimisation.

Require $d$ active points. $T$ iterations.
$i = 1$ to $T$
Select points using Algorithm 1.
Maximise the approximation to the (log) likelihood (32) using a scaled conjugate gradient optimiser (Nabney, 2001).
noise parameter updates are required.
Select points using Algorithm 1.
Maximise the sum of the log partition functions (34) using a scaled conjugate gradient optimiser.
*
*

### 3.4 Noise Parameter Updates

As well as updating the parameters of the kernel, it may be helpful to update the parameters of the noise function, particularly in the case of the ordered categorical model which has a parameter for each inner category indicating the width of the region associated with that category. However, the likelihood approximation (32) is only indirectly dependent on those parameter values. One way forward would be to optimise a variational lower bound,

$$\sum_{n=1}^{N} \int q_d(f_n) \log p(y_n|f_n, \boldsymbol{\theta}) \, p(f_n) \, df_n - \sum_{n=1}^{N} \int q_d(f_n) \log q(f_n) \, df_n,$$

on the likelihood, where $\boldsymbol{\theta}$ are the parameters of the noise model that we wish to optimise. The relevant term in this bound is

$$\sum_{n=1}^{N} \int q_d(f_n) \log p(y_n|f_n, \boldsymbol{\theta}) \,. \tag{33}$$

This lower bound can be upper bounded by

$$\sum_{n=1}^{N} \log \int q_d(f_n) \, p(y_n|f_n, \boldsymbol{\theta}) = \sum_{n=1}^{N} \log Z_n. \tag{34}$$

For many models it will be straightforward to compute (33), however for all noise models it will be possible to compute (34). We found for the ordered categorical noise model (where optimisation of the noise model parameters is more critical) optimisation of (34) was sufficient.

## 4 Experiments

In the next section we turn to experimental evaluation of the algorithms we have described. All the code used for generating these results is available online from `http://www.dcs.shef.ac.uk/~neil/ivm/`. The algorithm for optimising kernel and noise parameters is given in Algorithm 2.

The covariance function can be developed from any positive definite kernel, furthermore a new positive definite kernel can be formed by adding kernels together. In our experiments we make use of three different kernel functions.

- The linear kernel is simply the matrix of inner products,

$$k_{\text{lin}}(\mathbf{x}_i, \mathbf{x}_j) := \theta_{\text{lin}} \cdot \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j \,,$$

  where $\theta$ is the process variance and controls the scale of the output functions.

- The popular RBF kernel leads to smooth functions

$$k_{\text{rbf}}(\mathbf{x}_i, \mathbf{x}_j) := \theta_{\text{rbf}} \cdot \exp\left(-\frac{\gamma}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)\right) ,$$

  where $\gamma$ is the inverse width parameter.

- We also considered the MLP kernel (Williams, 1997) which is derived by considering a multi-layer perceptron with infinite hidden units,

$$k_{\text{mlp}}(\mathbf{x}_i, \mathbf{x}_j) := \theta_{\text{mlp}} \cdot \sin^{-1}\left(\frac{w\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + b}{\sqrt{\left(w\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i + b + 1\right)\left(w\mathbf{x}_j^{\mathrm{T}}\mathbf{x}_j + b + 1\right)}}\right) ,$$

  where we call $w$ the weight variance and $b$ the bias variance (they have interpretations as the variances of prior distributions in the neural network model).

- A white noise process has a kernel of the form

$$k_{\text{white}}(\mathbf{x}_i, \mathbf{x}_j) := \theta_{\text{white}} \cdot \delta_{ij} ,$$

  where $\delta_{ij}$ is the Kronecker delta which is zero unless $i = j$ when it takes the value 1.

- It is possible to represent uncertainty in the bias by adding a constant term to the kernel matrix,

$$k_{\text{bias}}(\mathbf{x}_i, \mathbf{x}_j) := \theta_{\text{bias}} ,$$

  where we have denoted the variance, $\theta_{\text{bias}}$.

Since all our kernels have a parameters which provides an overall scaling on these inner products (the process variance, the inverse width and the weight variance for the linear, RBF and MLP kernels respectively) we constrain these input scales to be at most 1. All the parameters that are constrained to be positive are implemented by reparameterising,

$$\theta := \log\left(1 + \exp\left(\theta'\right)\right) .$$

Note that as the transformed parameter tends to negative infinity, $\theta' \to -\infty$, the parameter tends to zero, $\theta \to 0$. Similarly, as $\theta' \to \infty$ we see that $\theta \to \theta'$. Finally we considered automatic relevance determination (ARD) versions of each our covariance functions. These priors are formed by replacing any inner product, $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$, with a matrix inner product, $\mathbf{x}_i^{\mathrm{T}}\mathbf{A}\mathbf{x}_j$. If $\mathbf{A}$ is positive (semi-)definite then each kernel will still be valid. The ARD kernels are the specific case where $\mathbf{A}$ is a diagonal matrix, $\mathbf{A} = \text{diag}(\boldsymbol{\alpha})$ and the $i$th diagonal element, $\alpha_i$, provides a scaling on the $i$th input variable. The positive definite constraint means that they cannot go below 0 so we use the sigmoid funciton for reparameterising,

$$\alpha_i = \frac{1}{1 + \exp\left(-\alpha'\right)}.$$

Finally we used a consitent initialisation of the parameters for all experiments. This was $\theta_{\text{lin}} = 1$, $\theta_{\text{rbf}} = 1$, $\gamma = 1$, $\theta_{\text{mlp}} = 1$, $w = 10$, $b = 10$ and $\boldsymbol{\alpha} = [0.999, \ldots, 0.999]^{\mathrm{T}}$.

## 4.1 Toy Problems

Before undertaking any large scale experiments we first present some illustrative results on simple two dimensional toy problems.

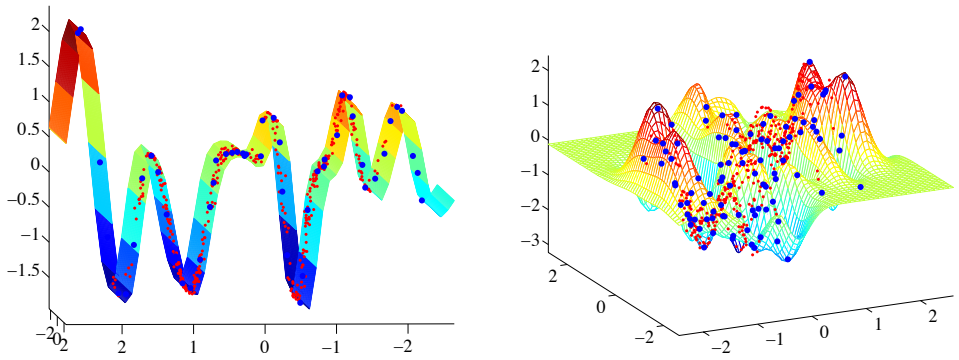

Figure 2: Regression toy problems, input data was sampled from a data-points sampled regression function sampled form a mixture of two Gaussians. The outputs were then generated from *left*: a sample from Gaussian process prior with an RBF ARD kernel. One of the input scale parameters was set to zero. *Right*: a similar kernel but now with input scales $\boldsymbol{\alpha} = [1, 0.2]^{\mathrm{T}}$. In both plots the data-points are red dots, the active set are blue spots and the mean of the process posterior as approximated by the IVM algorithm is a shaded surface.

#### 4.1.1 Regression

For both the following regression examples we sampled input data poisitions from a mixture of two Gaussians with means $[-1, -1]$ and $[1, 1]$ and a shared covariance of $0.25\mathbf{I}$. In the first problem we sampled $y$-values by taking a sample form a GP prior with an RBF ARD kernel function. The process variance for this kernel was $\theta_{\mathrm{rbf}} = 1$, the inverse width $\gamma = 20$, and the input scales , $\boldsymbol{\alpha} = [0, 1]^{\mathrm{T}}$. We then added Gaussian noise with a variance of 0.001. Figure 2, *left*, then shows the regression surface infered by an IVM. The approximating model used a combination of a linear ARD kernel and an RBF ARD kernel, the input scale parameters being shared across the kernels. The IVM was optimised using four iterations of Algorithm 2 and with an active set size of $d = 50$. The infered parameters were, $\theta_{\mathrm{rbf}} = 1.22$, $\gamma = 18.0$, $\theta_{\mathrm{lin}} = 9.63 \times 10^{-9}$, $\boldsymbol{\alpha} = [9.22 \times 10^{-11}, 1.00]^{\mathrm{T}}$. In other words the algorithm was able to infer that there was no linear trend present in the function made good approximations of the true generating functions parameters.

In the second simple regression example (Figure 2, *right*) the data was generated in a similar manner, but the generating input scales were now set to $\boldsymbol{\alpha} = [1, 0.2]^{\mathrm{T}}$. The IVM was trained as before with the exception of the active set size, $d$, which was now set at $d = 100$. The infered parameters were, $\theta_{\mathrm{rbf}} = 1.12$, $\gamma = 22.5$, $\theta_{\mathrm{lin}} = 9.12 \times 10^{-6}$, $\boldsymbol{\alpha} = [1.00, 0.163]^{\mathrm{T}}$. Once again the algoritm seems to have arrived at a good approximation of the generating function's parameters.

#### 4.1.2 Classification

For classification with the probit noise model we considered two data-sets. The first was sampled from a two dimensional mixture of Gaussians (Figure 3, *left*) with three components. The components shared the same covariance matrix but had different means. The means were aligned in such a way that only one input direction was relevant in determining class (the $y$-axis in the figure). An IVM model with a combined MLP ARD and linear ARD kernel was used, and its parameters were inferred by using four iterations of Algorithm 2 . From Figure 3, *left*, it can clearly be seen that the model has learnt that the $x$-axis direction is irrelevant. The linear portion of the kernel was also switched off. This is reflected in the infered

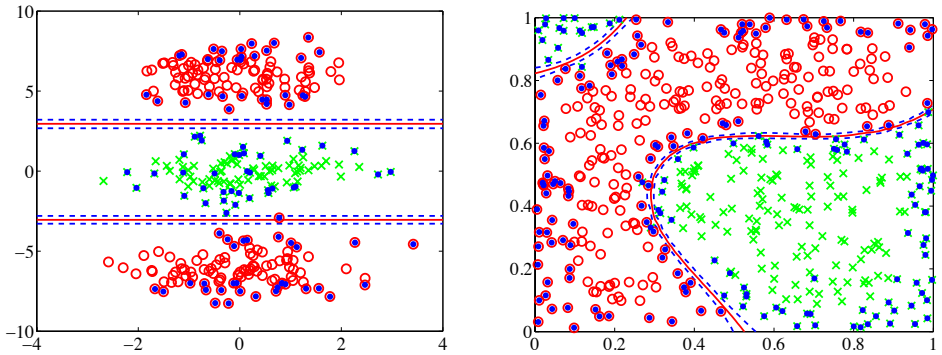


Figure 3: Classification toy problems. In both plots data from the negative class is shown as green crosses and that from the positive class is shown as red circles. The posterior probability of class membership is visualised as a contour (red solid line) at $p(y|\mathbf{x}) = 0.5$ and two contours (blue dashed lines) at $p(y|\mathbf{x}) = 0.25$ and $p(y|\mathbf{x}) = 0.75$. Points used in the active set are marked with blue dots. *Left*: data sampled from from a mixture of Gaussians. The IVM uses an ARD based prior and disregards the $x$-axis direction. *Right*: Data uniformly sampled on the 2–dimensional unit square. Class labels are assigned by sampling from a known Gaussian process prior.

parameters which were $\theta_{\text{mlp}} = 31.8$, $w = 0.473$, $b = 4.95$, $\theta_{\text{lin}} = 1.70 \times 10^{-7}$, $\boldsymbol{\alpha} = [6.12 \times 10^{-8}, 1.00]^{\text{T}}$.

For a second demonstation we sampled 500 data points uniformly from a unit square in two dimensions. A sample was then made from a GP prior of a function at these points. The covariance function (kernel) of the prior was an RBF with process variance of $\theta_{\text{rbf}} = 100$ and an inverse width of $\gamma = 10$. This function was 'squashed' by a cumulative Gaussian distirbution function and a class was assigned to each point randomly with a probability given by the output of the cumulative Gaussian. An IVM was then trained with an RBF kernel and $d = 200$. The data and resulting decision boundary are shown in Figure 3, *right*. The inverse width learnt by the model was $\gamma = 8.51$, which is close to that from the generating Gaussian process of 10, the process variance was $\theta_{\text{rbf}} = 6.54$, which is somewhat different from that of the generating distribution. This is not suprising as squashing the function with the cumulative Gaussian has the effect of removing a lot of the information in the data about the process variance.

### 4.1.3 Ordered Categories

Finally we show results from two problems on ordered categorical data. In the first example each category was generated from an isotropic white Gaussian distribution. The means of each category were located 3 units apart along one direction (the $y$-axis in Figure 4, *left*). It should be possible to seperate the categories with only one input direction using a linear model. We infered the parameters of an IVM with an RBF ARD kernel and a linear ARD kernel. The parameters were infered using four iterations of Algorithm 2 using the optional noise parameter optimisation and setting $d = 200$. As expected the RBF kernel was switched off ($\theta_{\text{rbf}} = 8.85 \times 10^{-5}$) as was one dimension of the linear kernel ($\boldsymbol{\alpha} = [6.03 \times 10^{-8}, 1.00]^{\text{T}}$).

In our second experiment we sampled ordered categorical data in polar co-ordinates to provide a non-linear decision boundary (Figure 4, *right*). Note in particular that the uncertainty in the decision boundary is reduced in the region

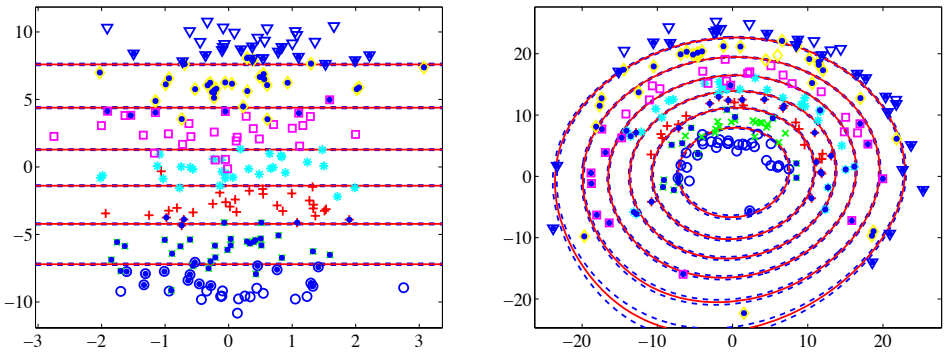


Figure 4: Ordered categories toy problems, the different categories are shown with different coloured symbols. Active data-points are marked with blue spots. *Left*: this problem can be solved with a linear model. *Right*: this categories in this example were sampled in polar co-ordinates. In both examples the decision boundaries between the categories are shown as red solid lines with blue dashed lines representing the expected quartiles.

Table 1: Table of results on the USPS digit data. The figures show the results for the individual binary classification tasks and the overall error computed from the combined classifiers. Each result is sumarised by the % classification error.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Overall |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RBF | 0.648 | 0.698 | 1.40 | 1.05 | 1.49 | 1.25 | 0.747 | 0.598 | 1.20 | 0.747 | 4.58 |
| MLP | 0.548 | 0.698 | 1.49 | 1.20 | 1.64 | 1.25 | 0.797 | 0.598 | 1.20 | 0.747 | 4.78 |
| RBF ARD | 0.548 | 0.598 | 1.49 | 1.10 | 1.79 | 1.20 | 0.797 | 0.498 | 1.20 | 0.847 | 4.68 |

towards the bottom of the plot where there is less data.

## 4.2 USPS digits

For a more realistic challenge we chose the USPS digit data set of $16 \times 16$ greyscale images. The data contains 7291 training images and 2007 test images. We implemented three different kernels with the IVM algorithm. For each data-set we used a 'base kernel' consisting of a linear part, a white noise term and a bias part. Three variations on this base kernel were then used: it was changed by adding first an RBF kernel, then an MLP kernel and finally a variant of the RBF ARD kernel[4]. We ran the experiments for five iterations using Algorithm 2 using the optional noise parameters and setting $d = 500$. The results are summarised in Table 1. A comparison with a summary of results on this data-set (Schölkopf and Smola, 2001, Table 7.4) shows that the IVM is in line with other results on this data. Furthermore these results were achieved with fully automated model selection.

[4]To prevent too many free parameters we placed hard constraints on many of the ARD parameters. We spilt the $16 \times 16$ input data into 16 seperate blocks, each containing 16 pixels. The input scale parameter was then shared within each of these blocks.

# 5 Discussion

In this paper, we have introduced the *informative vector machine*—a sparse and efficient variant of Gaussian proceeses algorithm. At its core, the new algorithm uses an ADF approximation to the Gaussian proccess posterior of the latent function outputs at the sample data points combined with an entropy-reduction based data point selection cirterion. We have demonstrated that the resulting algorithm is fast in runtime and matches state-of-the-art prediction performance in a variety of applications ranging from classification, regression to ordinal regression. There are a number of interesting questions related to learning sparse predictors that we have not addressed in this work:

**Accurate Posterior Approximation** In the current framework, we approximate the posterior $q_d(\mathbf{f})$ sequentially without re-considering early inclusions. The *expectation-propagation* (EP) algorithm addresses this problem by re-iterating over the data points to refine and improve the posterior approximation, $q(\mathbf{f})$ (Minka, 2001). A naive implementation of the EP in the IVM framework would lead to an increase in both the memory and computational complexity as the number of columns of the $\mathbf{M}$ matrix would grow without bound (see Line 12 in Algorithm 1). Our current research is focused on an approximate EP algorithm which uses Cholesky approximations to the posterior covariance matrix $\mathbf{\Sigma}_i$ thus overcoming both the representational and computational issues.

# References


J. A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research*, Cambridge, MA, 1988. MIT Press.

C. Cortes and V. N. Vapnik. Support vector networks. *Machine Learning*, 20: 273–297, 1995.

L. Csató. *Gaussian Processes — Iterative Sparse Approximations*. PhD thesis, Aston University, 2002.

L. Csató and M. Opper. Sparse representation for Gaussian process models. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 444–450, Cambridge, MA, 2001. MIT Press.

N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 625–632, Cambridge, MA, 2003. MIT Press.

D. J. C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991.

T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

I. T. Nabney. *Netlab: Algorithms for Pattern Recognition*. Advances in Pattern Recognition. Springer, Berlin, 2001. Code available from `http://www.ncrg.aston.ac.uk/netlab/`.

A. O'Hagan. Some Bayesian numerical analysis. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 4*, pages 345–363, Valencia, 1992. Oxford University Press.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Programming: Explorations in the Microstructure of Cognition*, volume 1: Foundations, pages 318–362. MIT Press, Cambridge, MA, 1986. Reprinted in (Anderson and Rosenfeld, 1988).

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2001.

M. Seeger. *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, The University of Edinburgh, 2004.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Conference on Computational Learning Theory 10*, pages 287–294. Morgan Kauffman, 1992.

C. K. I. Williams. Computing with infinite networks. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, Cambridge, MA, 1997. MIT Press.

C. K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan, editor, *Learning in Graphical Models*, volume 89 of *Series D: Behavioural and Social Sciences*, Dordrecht, The Netherlands, 1998. Kluwer.

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 514–520, Cambridge, MA, 1996. MIT Press.

# A Matching Expected Natural Statistics

In this appendix we show that minimising the Kullback-Leibler divergence (8) over a family in the class of exponential distributions is achieved by matching the *expected natural statistic*. We will also give an explicit update formula for distributions with only one likelihood term.

A set of distributions over $\mathbb{R}^N$ is in the exponential family if its densities can be written as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})\right) ,$$

where $\boldsymbol{\phi}(\mathbf{x})$ is known as the *natural statistic* of $\mathbf{x}$ and $Z(\boldsymbol{\theta}) := \int \exp(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})) d\mathbf{x}$ ensure normalisation. The exponential family includes many known families of distributions including the Gaussian distribution. For example, in the Gaussian case, the natural statistic $\boldsymbol{\phi}(\mathbf{x})$ is simply the vector of all first and second moments, $\boldsymbol{\phi}(\mathbf{x}) = (x_1, \ldots, x_N, x_1^2, x_1 x_2, \ldots, x_N x_{N-1}, x_N^2)$. Note that the expected natural statistic of $p_{\boldsymbol{\theta}}(\mathbf{x})$ is given in terms of the gradient of $\log(Z(\boldsymbol{\theta}))$ w.r.t. $\boldsymbol{\theta}$, that is,

$$\nabla_{\boldsymbol{\theta}} \log(Z(\boldsymbol{\theta})) = \frac{\int \left[\nabla_{\boldsymbol{\theta}} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})\right)\right] d\mathbf{x}}{Z(\boldsymbol{\theta})} = \langle \boldsymbol{\phi}(\mathbf{x}) \rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} . \tag{35}$$

For any distribution $p$, the distribution $p_{\boldsymbol{\theta}^*}$ which minimises the Kullback-Leibler divergence, $\mathrm{KL}(p||p_{\boldsymbol{\theta}^*})$, over the exponential family with natural statistic $\boldsymbol{\phi}$ is implicitly given by

$$\langle \boldsymbol{\phi}(\mathbf{x}) \rangle_{p_{\boldsymbol{\theta}^*}(\mathbf{x})} = \langle \boldsymbol{\phi}(\mathbf{x}) \rangle_{p(\mathbf{x})} . \tag{36}$$

Recall the Kullback-Leibler divergence from (8) and consider it as a function $f$ of the parameters $\boldsymbol{\theta}$,

$$\begin{aligned} f(\boldsymbol{\theta}) &= \mathrm{KL}(p||p_{\boldsymbol{\theta}}) = \left\langle \log\left(\frac{p(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right)\right\rangle_{p(\mathbf{x})} \\ &= \langle \log(p(\mathbf{x}))\rangle_{p(\mathbf{x})} + \langle \log(Z(\boldsymbol{\theta}))\rangle_{p(\mathbf{x})} - \left\langle \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right\rangle_{p(\mathbf{x})} \\ &= \langle \log(p(\mathbf{x}))\rangle_{p(\mathbf{x})} + \log(Z(\boldsymbol{\theta})) - \boldsymbol{\theta}^{\mathrm{T}}\langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p(\mathbf{x})} . \end{aligned}$$

A necessary condition for the minimum $\boldsymbol{\theta}^*$ is $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^*) = \mathbf{0}$. From (35) we have

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} - \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p(\mathbf{x})} .$$

It remains to show that $\boldsymbol{\theta}^*$ such that $\langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}^*}(\mathbf{x})} = \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p(\mathbf{x})}$ is a minimum. To this end, consider the matrix of second derivatives,

$$\begin{aligned} \left[\nabla\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\right]_{i,j} &= \frac{\partial^2 \log(Z(\boldsymbol{\theta}))}{\partial\theta_i \partial\theta_j} = \frac{\partial}{\partial\theta_j}\frac{\int \phi_i(\mathbf{x}) \exp\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right) d\mathbf{x}}{Z(\boldsymbol{\theta})} \\ &= \langle \phi_i(\mathbf{x})\phi_j(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} - \langle \phi_i(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})}\langle \phi_j(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} . \end{aligned}$$

At the solution $\boldsymbol{\theta}^*$, this is the covariance matrix of the natural statistic $\boldsymbol{\phi}(\mathbf{x})$ over the distribution $p_{\boldsymbol{\theta}^*}$. By definition, this is positive semi-definite matrix (in fact, for every distribution $p_{\boldsymbol{\theta}}$) and thus we have proven the theorem.

In the case of the Gaussian family, $\{\mathcal{N}(\cdot;\boldsymbol{\mu},\boldsymbol{\Sigma})\}$, minimising the KL divergence reduces to matching the mean and covariance (which are related in a one-to-one way to the first and second moments),

$$\boldsymbol{\mu}^* = \langle \mathbf{x}\rangle_{p(\mathbf{x})} , \tag{37}$$

$$\boldsymbol{\Sigma}^* = \left\langle \mathbf{x}\mathbf{x}^{\mathrm{T}}\right\rangle_{p(\mathbf{x})} - \langle \mathbf{x}\rangle_{p(\mathbf{x})}\langle \mathbf{x}\rangle_{p(\mathbf{x})}^{\mathrm{T}} . \tag{38}$$

## A.1 General Update Equations

We will now derive an explicit update formula for matching the expected natural statistic if $p(\mathbf{x})$ has the simple form

$$p(\mathbf{x}) = \frac{1}{\tilde{Z}(\boldsymbol{\theta})} \cdot t(\mathbf{x})\, p_{\boldsymbol{\theta}}(\mathbf{x}) ,$$

where $\tilde{Z}(\boldsymbol{\theta}) := \int t(\mathbf{x}) p_{\boldsymbol{\theta}}(\mathbf{x}) d\mathbf{x}$ ensures normalisation[5]. In fact, similar to (35), the expected natural statistic under $p(\mathbf{x})$ can again be expressed solely in terms of the gradient of $\tilde{Z}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$. In order to see this, note that

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x}) &= \left[\nabla_{\boldsymbol{\theta}}\frac{1}{Z(\boldsymbol{\theta})}\right]\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right) + \frac{1}{Z(\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}}\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right)\right] \\ &= -\frac{[\nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta})]}{Z(\boldsymbol{\theta})} p_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\phi}(\mathbf{x})\, p_{\boldsymbol{\theta}}(\mathbf{x}) \\ &= -\langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} \cdot p_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\phi}(\mathbf{x})\, p_{\boldsymbol{\theta}}(\mathbf{x}) . \end{aligned}$$

Multiplying both sides by $\tilde{Z}^{-1}(\boldsymbol{\theta}) t(\mathbf{x})$, integrating over $\mathbf{x}$ and re-arranging terms we get

$$\begin{aligned} \tilde{Z}^{-1}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\tilde{Z}(\boldsymbol{\theta}) &= -\langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} + \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p(\mathbf{x})} \\ \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p(\mathbf{x})} &= \nabla_{\boldsymbol{\theta}} \log\left(\tilde{Z}(\boldsymbol{\theta})\right) + \langle \boldsymbol{\phi}(\mathbf{x})\rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})} . \end{aligned} \tag{39}$$

[5]Please note that the normalisation constant $\tilde{Z}(\boldsymbol{\theta})$ should not be confused with the normalisation constant $Z(\boldsymbol{\theta})$.

Finally, using the fact that matching the natural statistics minimises the Kullback-Leibler divergence and (35) we obtain

$$\nabla_{\boldsymbol{\theta}} \log\left(Z\left(\boldsymbol{\theta}^*\right)\right) = \nabla_{\boldsymbol{\theta}} \log\left(\tilde{Z}\left(\boldsymbol{\theta}\right)\right) + \nabla_{\boldsymbol{\theta}} \log\left(Z\left(\boldsymbol{\theta}\right)\right) .$$

All that is required to solve the above equation for a given exponential family is to know the analytical solution of the gradient equation of $\log(Z(\boldsymbol{\theta}))$ and $\log(\tilde{Z}(\boldsymbol{\theta}))$. These two equations only depend on the particular natural statistic function $\boldsymbol{\phi}$ and the function $t$. This is applicable, for example, for Gamma densities.

However, some exponential families are usually not parameterised in terms of $\boldsymbol{\theta}$ but rather in terms of $\boldsymbol{\tau}(\boldsymbol{\theta}) := \langle \boldsymbol{\phi}(\mathbf{x}) \rangle_{p_{\boldsymbol{\theta}}(\mathbf{x})}$—a parameterisation also known as the *moment representation*. This representation has particular advantages when minimising the KL divergence as Theorem 1 directly specifies the update equation for the parameters. In this case, (39) can still be used together with the chain rule of differtiation to obtain the update equation for a particular class of exponential densities if the mapping to $\boldsymbol{\tau} \mapsto \boldsymbol{\theta}$ is easy to differentiate. We can also follow the above argument simply in the new parameterisation. In the next section we give a detailed derivation for the Gaussian family (which is represented in terms of its moments).

## A.2 Update Equations for the Gaussian Family

We consider a family of Gaussians parameterised in terms of its mean, $\boldsymbol{\mu}$, and covariance, $\boldsymbol{\Sigma}$,

$$q\left(\mathbf{x}\right) := q\left(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) := \mathcal{N}\left(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) .$$

Our ability to compute (37) and (38) when $p(\mathbf{x}) \propto t(\mathbf{x}) q\left(\mathbf{x}\right)$ depends only on the tractability of the normalisation constant,

$$\tilde{Z} := \tilde{Z}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) := \int t\left(\mathbf{x}\right) q\left(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) d\mathbf{x} .$$

**Matching the Mean** We will consider the mean of $\mathbf{x}$ under $t(\mathbf{x})q(\mathbf{x})$. First note that

$$\nabla_{\boldsymbol{\mu}} q\left(\mathbf{x}\right) = \boldsymbol{\Sigma}^{-1}\left(\mathbf{x} - \boldsymbol{\mu}\right) q\left(\mathbf{x}\right) ,$$

which can be re-expressed in terms of $\mathbf{x}q\left(\mathbf{x}\right)$,

$$\mathbf{x} q\left(\mathbf{x}\right) = \boldsymbol{\mu} q\left(\mathbf{x}\right) + \boldsymbol{\Sigma} \nabla_{\boldsymbol{\mu}} q\left(\mathbf{x}\right) .$$

Now multiplying both sides by $\tilde{Z}^{-1} t(\mathbf{x})$, integrating over $\mathbf{x}$, and exploiting the linearity of the gradient operator gives

$$\begin{aligned}
\langle \mathbf{x} \rangle_{p(\mathbf{x})} &= \boldsymbol{\mu} + \tilde{Z}^{-1} \cdot \boldsymbol{\Sigma} \left[ \nabla_{\boldsymbol{\mu}} \int t\left(\mathbf{x}\right) q\left(\mathbf{x}\right) d\mathbf{x} \right] \\
&= \boldsymbol{\mu} + \tilde{Z}^{-1}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \cdot \boldsymbol{\Sigma} \nabla_{\boldsymbol{\mu}} \tilde{Z}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \\
&= \boldsymbol{\mu} + \boldsymbol{\Sigma} \nabla_{\boldsymbol{\mu}} \log\left(\tilde{Z}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)\right) \\
&= \boldsymbol{\mu} + \boldsymbol{\Sigma} \mathbf{g} , \qquad (40)
\end{aligned}$$

where we have defined $\mathbf{g} := \nabla_{\boldsymbol{\mu}} \log(\tilde{Z}(\boldsymbol{\mu}, \boldsymbol{\Sigma}))$.

**The Second Moment Matrix** Once again we take gradients[6] of $q(\mathbf{x})$, but this time with respect to the covariance matrix $\boldsymbol{\Sigma}$,

$$\nabla_{\boldsymbol{\Sigma}} q(\mathbf{x}) = \frac{1}{2}\left(-\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\right) q(\mathbf{x}) \,,$$

which can be re-arranged, as we did before, in order to obtain

$$\mathbf{x}\mathbf{x}^{\mathrm{T}} q(\mathbf{x}) = 2\boldsymbol{\Sigma}\left[\nabla_{\boldsymbol{\Sigma}} q(\mathbf{x})\right]\boldsymbol{\Sigma} + \left(\boldsymbol{\Sigma} + \mathbf{x}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{\mu}\mathbf{x}^{\mathrm{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}}\right) q(\mathbf{x}) \,.$$

Multiplying both sides by $\tilde{Z}^{-1}t(\mathbf{x})$, integrating over $\mathbf{x}$ and exploiting the linearity of the gradient operator gives

$$\begin{aligned}
\left\langle \mathbf{x}\mathbf{x}^{\mathrm{T}} \right\rangle_{p(\mathbf{x})} &= \boldsymbol{\Sigma} + 2\boldsymbol{\Sigma}\left(\tilde{Z}^{-1}(\boldsymbol{\mu},\boldsymbol{\Sigma})\,\nabla_{\boldsymbol{\Sigma}}\tilde{Z}(\boldsymbol{\mu},\boldsymbol{\Sigma})\right)\boldsymbol{\Sigma} + \langle \mathbf{x}\rangle_{p(\mathbf{x})}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{\mu}\langle \mathbf{x}\rangle_{p(\mathbf{x})}^{\mathrm{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}} \\
&= \boldsymbol{\Sigma} + 2\boldsymbol{\Sigma}\left(\nabla_{\boldsymbol{\Sigma}}\log\left(\tilde{Z}(\boldsymbol{\mu},\boldsymbol{\Sigma})\right)\right)\boldsymbol{\Sigma} + \langle \mathbf{x}\rangle_{p(\mathbf{x})}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{\mu}\langle \mathbf{x}\rangle_{p(\mathbf{x})}^{\mathrm{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}} \\
&= \boldsymbol{\Sigma} + 2\boldsymbol{\Sigma}\mathbf{G}\boldsymbol{\Sigma} + \langle \mathbf{x}\rangle_{p(\mathbf{x})}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{\mu}\langle \mathbf{x}\rangle_{p(\mathbf{x})}^{\mathrm{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}} \,,
\end{aligned}$$

where we have defined $\mathbf{G} := \nabla_{\boldsymbol{\Sigma}}\log(\tilde{Z}(\boldsymbol{\mu},\boldsymbol{\Sigma}))$.

**Matching the Covariance** The update (38) for the covariance requires to compute

$$\left\langle \mathbf{x}\mathbf{x}^{\mathrm{T}} \right\rangle_{p(\mathbf{x})} - \langle \mathbf{x}\rangle_{p(\mathbf{x})}\langle \mathbf{x}\rangle_{p(\mathbf{x})}^{\mathrm{T}} = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\left(\mathbf{g}\mathbf{g}^{\mathrm{T}} - 2\mathbf{G}\right)\boldsymbol{\Sigma} \,, \tag{41}$$

where we used (40). Substituting (40) and (41) into (37) and (38) we obtain the required updates for the mean and covariance in a form that applies regardless of our noise model (Csató, 2002; Minka, 2001; Seeger, 2004):

$$\begin{aligned}
\boldsymbol{\mu}^{*} &= \boldsymbol{\mu} + \boldsymbol{\Sigma}\mathbf{g} \,, \\
\boldsymbol{\Sigma}^{*} &= \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\left(\mathbf{g}\mathbf{g}^{\mathrm{T}} - 2\mathbf{G}\right)\boldsymbol{\Sigma} \,.
\end{aligned}$$

[6] It helps to remember that $\nabla_{\boldsymbol{\Sigma}}\log(q(\mathbf{x})) = (q(\mathbf{x}))^{-\mathbf{1}} \cdot \nabla_{\boldsymbol{\Sigma}} q(\mathbf{x}))$.