# Extensions of the Informative Vector Machine

Neil D. Lawrence[1], John C. Platt[2] and Michael I. Jordan[3]

[1] Department of Computer Science, University of Sheffield, Regent Court, 211
Portobello Street, Sheffield S1 4DP, U.K. neil@dcs.shef.ac.uk
[2] Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA
98052, U.S.A. jplatt@microsoft.com
[3] Computer Science and Statistics, University of California, Berkeley, CA 94720,
U.S.A. jordan@cs.berkeley.edu

**Abstract** The informative vector machine (IVM) is a practical method for Gaussian process regression and classification. The IVM produces a sparse approximation to a Gaussian process by combining assumed density filtering with a heuristic for choosing points based on minimizing posterior entropy. This paper extends IVM in several ways. First, we propose a novel noise model that allows the IVM to be applied to a mixture of labeled and unlabeled data. Second, we use IVM on a block-diagonal covariance matrix, for "learning to learn" from related tasks. Third, we modify the IVM to incorporate prior knowledge from known invariances. All of these extensions are tested on artificial and real data.

## 1 Introduction

Kernel-based methods have become increasingly popular in the machine learning field. Their close relationships to convex optimization and numerical linear algebra—and their corresponding amenability to theoretical analysis—have helped to fuel their fast growth relative to older non-convex regression and classification methods such as neural networks. Kernel-based methods reduce the data to a "kernel matrix," the entries of which are evaluations of a "kernel function" or "covariance function." Computationally efficient algorithms are available to optimize various statistical functionals of interest for given values of this matrix.

As with most statistical procedures, one can take either a Bayesian or a frequentist perspective on kernel-based methods. The Bayesian point of view involves using a Gaussian process framework to place prior distributions on families of regression or discriminant functions [28]. A Gaussian process is characterized by a mean function and a covariance function, and it is this latter function that yields the "kernel matrix" (when evaluated at the observed data). The frequentist point of view focuses on the optimization of loss functions defined on an inner product space; the choice of inner product defines the kernel function and thus the kernel matrix [19].

By focusing on point estimates, the frequentist approach can yield a stripped-down computational problem that has the appeal of tractability in the context of large-scale problems. For example, the support vector machine reduces to a relatively simple quadratic program [6]. This appeal, however, is somewhat diminished in practice by two factors: (1) many kernel-based procedures involve hyperparameters, and setting hyperparameters generally involves a computationally-intensive outer loops involving cross-validation; and (2) fuller frequentist inference requires calculating error bars for point estimates. This too often requires computationally-intensive extensions of the basic parameter-fitting algorithm (e.g., the bootstrap).

The Bayesian approach to kernel-based methods, on the other hand, focuses on the computation or approximation of posterior probability distributions under the Gaussian process prior. This is generally a more ambitious goal than computing a point estimate, but it also goes a significant distance towards solving the other practical problems associated with using kernel-based methods. Indeed, (1) standard procedures involving the marginal likelihood can be used to set hyperparameters; and (2) the posterior distribution naturally provides an assessment of uncertainty.

Another virtue of the Bayesian approach is the relative ease with which one can consider extensions of a basic model, and it is this virtue that is our focus in the current paper. In particular, we consider two elaborations of the basic classification setup: *semi-supervised learning* and *multi-task learning*. While the frequentist paradigm both permits and requires substantial creativity in deriving methods to attack problems such as these, the Bayesian approach provides standard tools with which to proceed. In particular, in this paper we show how standard Bayesian modeling tools—latent variable modeling and hierarchical modeling—can be used to address nonparametric semi-supervised and multi-task learning within the Gaussian process framework.

Kernel matrices are $N \times N$ matrices (where $N$ is the number of data points), and in both the Bayesian approach and the frequentist approach to kernel-based methods it is important to develop procedures that take advantage of putative low-rank structure in these matrices. One generally useful idea involves *sparsity*—one seeks functions that have expansions in which most of the coefficients are zero. Sparsity can be imposed in the loss function and/or imposed as part of a computational approximation. For example, in the support vector machine the use of a hinge loss implies that only a subset of the training data have non-zero coefficients—these are the *support vectors*. Support vectors embody computational efficiencies and also permit the design of useful extensions that are themselves computationally efficient. For example, *virtual support vectors* allow invariances to be incorporated into support vector machine training [18]. In the current paper, we show how a similar notion can be developed within the Bayesian paradigm—a notion that we refer to as *virtual informative vectors*.

In summary, we view sparse Gaussian process methods as providing a flexible, computationally-efficient approach to nonparametric regression and classification; one that is as viable in practice as its frequentist cousins, and one that is particularly easy to extend. In this paper we focus on a specific sparse Gaussian process methodology — the *informative vector machine* (IVM). We show how the basic IVM can be extended to accommodate our proposals for semi-supervised learning and multi-task learning. These extensions are not restricted to the IVM methodology: our approach to multi-task learning falls within the broader class of hierarchical Bayesian methods and our noise model for semi-supervised learning can be used in combination with a wide range of models. However the approach we take to incorporating invariances exploits a particular characteristic of the IVM. It relies on the fact that the IVM can be viewed as a *compression scheme*. By this we mean that the decision boundary can be inferred by considering only the 'active set'. This is a characteristic that the IVM shares with the support vector machine. Previously proposed sparse Gaussian process methods seek sparse representations for mean and covariance functions but rely on the entire data set to formulate them.

The paper is organized as follows. After an overview of the general Gaussian process methodology in Section 2, we review the the informative vector machine in Section 3. In Section 4, we present a latent variable approach to semi-supervised learning within the IVM framework. Section 5 develops the IVM-based approach to multi-task learning. Finally, in Section 6 we show how the notion of sparsity provided by the IVM may be exploited in the incorporation of prior invariances in the model.

## 2 Gaussian Processes

Consider a simple latent variable model in which the output observations, $\mathbf{y} = [y_1 \ldots y_N]^{\mathrm{T}}$, are assumed independent from input data, $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_N]^{\mathrm{T}}$, given a set of latent variables, $\mathbf{f} = [f_1 \ldots f_N]^{\mathrm{T}}$. The prior distribution for these latent variables is given by a Gaussian process [4],

$$p\left(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}\right) = N\left(\mathbf{f}|\mathbf{0}, \mathbf{K}\right),$$

with a covariance function, or 'kernel', $\mathbf{K}$, that is parameterised by the vector $\boldsymbol{\theta}$ and evaluated at the points given in $\mathbf{X}$. This relationship is shown graphically in Figure 1.

The joint likelihood can be written as

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

where $p\left(y_n|f_n\right)$ gives the relationship between the latent variable and our observations and is sometimes referred to as the *noise model*. For the regression case the noise model can also take the form of a Gaussian.

$$p\left(y_n|f_n\right) = N\left(y_n|f_n, \sigma^2\right). \tag{2}$$

[4] We use $N\left(\mathbf{x}|\boldsymbol{\mu}, \Sigma\right)$ to denote a Gaussian distribution over $\mathbf{x}$ with a mean vector $\boldsymbol{\mu}$ and covariance matrix $\Sigma$. When dealing with one dimensional Gaussians the vectors and matrices are replaced by scalars.

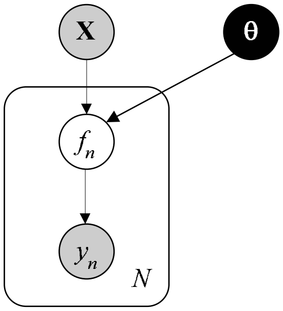


**Figure 1.** The Gaussian Process model drawn graphically. We have made use of 'plate' notation to indicate the independence relationship between **f** and **y**.

Then the marginalised likelihood can be obtained by integrating over **f** in (1). This marginalisation is straightforward—it can be seen to be a special case of the more general result,

$$\begin{aligned} p(\mathbf{y}) &= \int N(\mathbf{f}|\mathbf{0},\mathbf{K}) \prod_{n=1}^{N} N\left(y_n|f_n,\beta_n^{-1}\right) d\mathbf{f} \\ &= N\left(\mathbf{y}|\mathbf{0},\mathbf{K}+\mathbf{B}^{-1}\right), \end{aligned} \tag{3}$$

where **B** is a diagonal matrix whose $n$th diagonal element is given by $\beta_n$. To recover the special case associated with the spherical Gaussian noise model from (2) we simply substitute each $\beta_n$ with $\frac{1}{\sigma^2}$ and each $m_n$ with $y_n$.

### 2.1 Optimising Kernel Parameters

A key advantage of the Gaussian process framework is that once the marginal likelihood is computed then it can be maximised with respect to parameters of the kernel, $\boldsymbol{\theta}$, yielding a so-called *empirical Bayes* method for fitting the model. We have seen that for Gaussian noise models the computation of this marginal likelihood is straightforward. Unfortunately for non-Gaussian noise models the required marginalisation is intractable and we must turn to approximations. One such approximation is known as *assumed density filtering* (ADF). As we shall see, the ADF approximation proceeds by incorporating the data a single point at a time and using Gaussian approximations to the non-Gaussian posterior distributions that arise to maintain the tractability of the model.

### 2.2 The ADF Approximation

Assumed density filtering has its origins in on-line learning: it proceeds by absorbing one data point at a time, computing the modified posterior and replacing it with an approximation that maintains the tractability of the algorithm. A thorough treatment of this approach is given by [16,7]. Here we review the main points of relevance for the IVM algorithm.

Given a noise model, $p\left(y_n|f_n\right)$, we note that the joint distribution over the data and the latent variables factorises as follows

$$p\left(\mathbf{y},\mathbf{f}\right) = N\left(\mathbf{f}|\mathbf{0},\mathbf{K}\right)\prod_{n=1}^{N} p\left(y_n|f_n\right)$$

which may be written

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

where we have defined $t_n\left(\mathbf{f}\right) = p\left(y_n|f_n\right)$ and in a slight abuse of notation we have taken $t_0\left(\mathbf{f}\right) = N\left(\mathbf{f}|\mathbf{0},\mathbf{K}\right)$. ADF takes advantage of this factorised form to build up an approximation, $q\left(\mathbf{f}\right)$, to the true process posterior, $p\left(\mathbf{f}|\mathbf{y}\right)$. 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 $q_i\left(\mathbf{f}\right)$. The starting point is to match the approximation to the prior process, *i.e.* $q_0\left(\mathbf{f}\right) = t_0\left(\mathbf{f}\right) = N\left(\mathbf{f}|\mathbf{0},\mathbf{K}\right)$. The approximation is then constrained to always have this functional form. As the data point inclusion process is sequential two index sets can be maintained. The first, $I$, is referred to as the *active set* and represents those data points that have been included in our approximation. The second, $J$, is referred to as the *inactive set* and represents those data points that have not yet been incorporated in our approximation. Initially $I$ is empty and $J = \{1 \ldots N\}$. The approximation to the true posterior is built up by selecting a point, $n_1$, from $J$. This point is included in the active set leading to an updated posterior distribution of the form,

$$\hat{p}_1\left(\mathbf{f}\right) \propto q_0\left(\mathbf{f}\right) t_{n_1}\left(\mathbf{f}\right),$$

our new approximation, $q_1\left(\mathbf{f}\right)$, is then found by minimising the Kullback Leibler (KL) divergence between the two distributions.

$$\mathrm{KL}\left(\hat{p}_1||q_1\right) = -\int \hat{p}_1\left(\mathbf{f}\right)\log\frac{q_1\left(\mathbf{f}\right)}{\hat{p}_1\left(\mathbf{f}\right)}d\mathbf{f}.$$

More generally, for the inclusion of the $i$th data point, we can write

$$\hat{p}_i\left(\mathbf{f}\right) = \frac{q_{i-1}\left(\mathbf{f}\right) t_{n_i}\left(\mathbf{f}\right)}{Z_i} \tag{4}$$

where the normalization constant is

$$Z_i = \int t_{n_i}\left(\mathbf{f}\right) q_{i-1}\left(\mathbf{f}\right) d\mathbf{f}. \tag{5}$$

ADF minimises $\mathrm{KL}\left(\hat{p}_i||q_i\right)$ to obtain the new approximation. This KL divergence can be minimised by 'moment matching' equations of the form

$$q_i\left(\mathbf{f}\right) = N\left(\mathbf{f}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i\right).$$

where

$$\boldsymbol{\mu}_i = \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})} \tag{6}$$

$$\boldsymbol{\Sigma}_i = \left\langle \mathbf{f}\mathbf{f}^{\mathrm{T}} \right\rangle_{\hat{p}_i(\mathbf{f})} - \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})} \langle \mathbf{f} \rangle_{\hat{p}_i(\mathbf{f})}{}^{\mathrm{T}} \tag{7}$$

and where $\langle \cdot \rangle_{p(\cdot)}$ denotes an expectation under the distribution $p(\cdot)$. It turns out that our ability to compute (6) and (7) depends on the tractability of the normalisation constant in (4). An update equation for the posterior mean (see Appendix A) is given by

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

where the dependence on $Z_i$ is through

$$\mathbf{g}_i = \frac{\partial \log Z_i}{\partial \boldsymbol{\mu}_i}.$$

The corresponding update for $\mathit{\Sigma}_i$ is given by

$$\mathit{\Sigma}_i = \mathit{\Sigma}_{i-1} - \mathit{\Sigma}_{i-1}\left(\mathbf{g}_i\mathbf{g}_i^{\mathrm{T}} - 2\mathit{\Gamma}_i\right)\mathit{\Sigma}_{i-1}$$

where

$$\mathit{\Gamma}_i = \frac{\partial \log Z_i}{\partial \mathit{\Sigma}_{i-1}}.$$

The ADF approximation can therefore be easily be applied for any noise model for which the expectation of the noise model in (5) is tractable. In the next sections we shall review two of the most commonly used noise models within the context of the ADF algorithm. The Gaussian noise model is commonly used in regression applications and the probit noise model is applicable for classification problems. Later, in Section 4, we describe a noise model that is suitable for semi-supervised learning.

### 2.3 Gaussian Noise Model

One of the most widely used approaches for interpolation between data points is model fitting through least squares. From the probabilistic point of view this is equivalent to a Gaussian noise model. If each data point, $y_n$, has a different associated variance, $\beta_n^{-1}$, then the noise model is given by

$$p(y_n|f_n) = \sqrt{\frac{\beta_n}{2\pi}} \exp\left(-\frac{\beta_n (y_n - f_n)^2}{2}\right).$$

As we described in the previous section, the updates for the mean and covariance functions depend on $Z_i$, the expectation of the noise model under the current estimate of the posterior process, $q_{i-1}(\mathbf{f})$. Since the noise model only depends on $f_n$, the $n$th element of $\mathbf{f}$, we may rewrite this expectation in the form

$$Z_i = \int p(y_n|f_n)\, q(f_n)\, df_n, \tag{8}$$

where we have exploited the fact that the marginal, $q(f_n)$, of the full multivariate Gaussian, $q(\mathbf{f})$, is given by

$$q(f_n) = N(f_n|\mu_{i-1,n}, \varsigma_{i-1,n})$$

where $\mu_{i-1,n}$ is the $n$th element of $\boldsymbol{\mu}_{i-1}$ and $\varsigma_{i-1,n}$ is the $n$th element from the diagonal of $\Sigma_{i-1}$. As a result the expectation in (8) is easily computed as

$$Z_i = N\left(y_n|\mu_{i-1,n}, \left(\varsigma_{i-1,n} + \beta_n^{-1}\right)\right)$$

The log partition function,

$$\log Z_i = -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\left(\varsigma_{i-1,n} + \beta_n^{-1}\right) - \frac{\left(y_n - \mu_{i-1,n}\right)^2}{2\left(\beta_n^{-1} + \varsigma_{i-1,n}\right)},$$

can then be differentiated with respect to $\boldsymbol{\mu}_{i-1}$ to give

$$g_{in} = \frac{y_n - \mu_{i-1,n}}{\beta_n^{-1} + \varsigma_{i-1,n}}, \tag{9}$$

where $g_{in}$ is the $n$th element of $\mathbf{g}_i$ and all other elements are zero. Similarly we can differentiate the log partition with respect to $\Sigma_{i-1}$ to find the $n$th diagonal element of $\Gamma_i$

$$\gamma_{in} = -\frac{1}{2\left(\varsigma_{i-1} + \beta_n^{-1}\right)} + \frac{1}{2}g_{in}^2$$

all other elements of $\Gamma_i$ are zero.

For the update of the covariance we need to consider the matrix $\mathbf{g}_i\mathbf{g}_i^{\mathrm{T}} - 2\Gamma_i$. However, for the Gaussian noise model this matrix can be written in diagonal form, $\mathrm{diag}(\boldsymbol{\nu}_i)$, where the $n$th element of $\boldsymbol{\nu}_i$ is

$$\begin{aligned}\nu_{in} &= -2\gamma_{in} + g_{in}^2 \\ &= \frac{1}{\left(\varsigma_{i-1} + \beta_n^{-1}\right)}\end{aligned} \tag{10}$$

and all other elements are zero. This leads to a set of simplified update equations of the form

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + g_{in}\mathbf{s}_{i-1,n}, \tag{11}$$

$$\Sigma_i = \Sigma_{i-1} - \nu_{in}\mathbf{s}_{i-1,n}\mathbf{s}_{i-1,n}^{\mathrm{T}}, \tag{12}$$

where $\mathbf{s}_{i-1,n}$ is the $n$th column from $\Sigma_{i-1}$.

Implicit in these update equations is the minimisation of a KL divergence between the true posterior and the approximating Gaussian process posterior, $q(\mathbf{f})$. Of course, for the Gaussian noise model, the true posterior process is Gaussian so the approximation is exact. In other words, for the special case of the Gaussian noise, these updates are simply a scheme for on-line learning of Gaussian processes without any approximations involved.

In the next section, we consider the probit classification noise model. This noise model is an example from the more general case where the true posterior distribution is non-Gaussian and every update of the mean and posterior implicitly involves an approximation to this true posterior through moment matching.

### 2.4 Probit Noise Model

We consider binary classification where $y_n \in \{-1, 1\}$. A convenient representation for the probability of class membership given the latent variable $f_n$ is given by the *probit* noise model for classification

$$p\left(y_n|f_n\right) = \phi\left(y_n \lambda\left(f_n + b\right)\right)$$

where $\phi\left(\cdot\right)$ is the cumulative Gaussian given by

$$\phi\left(z\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{t^2}{2}\right) dt,$$

the slope of which is controlled by $\lambda$. The use of the probit, rather than the more commonly encounter logit, is convenient as it leads to a tractable integral for the partition function:

$$Z_i = \frac{1}{\left(2\pi\right)^{\frac{N}{2}} \varsigma_{i-1,n}^{\frac{1}{2}}} \int \phi\left(y_n \lambda\left(f_n + b\right)\right) \exp\left(-\frac{\left(f_n - \boldsymbol{\mu}_{i-1}\right)^2}{2\varsigma_{i-1,n}}\right) df_n$$

which may be integrated to obtain

$$Z_i = \phi\left(u_{i-1,n}\right).$$

where

$$\begin{aligned} u_{i-1,n} &= c_{i-1,n}\left(\mu_{i-1,n} + b\right) \\ c_{i-1,n} &= \frac{y_n}{\sqrt{\lambda^{-2} + \varsigma_{i-1,n}}}. \end{aligned} \tag{13}$$

Once again, the partition function is only dependent on one element of $\boldsymbol{\mu}_{i-1}$ and one element of $\Sigma_{i-1}$. Performing the necessary derivatives to obtain $g_{in}$ and $\nu_{in}$ we have[5]

$$g_{in} = c_{i-1,n} \mathcal{N}\left(u_{i-1,n}\right)\left[\phi\left(u_{i-1,n}\right)\right]^{-1}, \tag{14}$$

where

$$\mathcal{N}\left(u_{i-1,n}\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u_{i-1,n}^2}{2}\right)$$

and

$$\gamma_{in} = -\frac{1}{2} g_{in} u_{i-1} c_{i-1,n}$$

which implies

$$\nu_{in} = g_{in}\left(g_{in} + u_{i-1} c_{i-1,n}\right). \tag{15}$$

[5] In implementation, care must be taken in computing $g_{in}$: when $u_{i-1,n}$ has large magnitude both $\phi\left(u_{i-1,n}\right)$and $N\left(u_{i-1,n}\right)$ become very small and numerical precision issues arise. This can be avoided by performing the computations should be done in the log domain.

Updates for $\boldsymbol{\mu}_{i-1} \rightarrow \boldsymbol{\mu}_i$ and $\Sigma_{i-1} \rightarrow \Sigma_i$ , the parameters of $q(\mathbf{f})$, are then identical to those given in (11) and (12). Note from (13) that we can consider the slope, $\lambda$, of the noise model to be infinite and develop an equivalent representation by adding a matrix $\lambda^{-2}\mathbf{I}$ to the kernel matrix thereby causing $\varsigma_{i,n}$ to increase by $\lambda^{-2}$. In our experiments, this approach is preferred because the noise model slope can then be optimised in tandem with the kernel parameters.

### 2.5 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 so that we may learn the kernel parameters via an empirical Bayes approach.

With reference 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 marginalisation can only be achieved exactly if the noise model is Gaussian. When dealing with non-Gaussian noise models we must again consider approximations.

One perspective on the ADF approximation is that we are taking non-Gaussian noise models and approximating them with Gaussian noise models. While in practise we are approximating a non-Gaussian posterior distribution with a Gaussian distribution, $q(\mathbf{f})$, this approximation can also be viewed as replacing the true noise model with a 'apparent Gaussian noise model' that also induces the posterior distribution $q(\mathbf{f})$. Note that this point of view is only reasonable if the noise model is log concave, otherwise the implied Gaussian distribution can have a negative variance.

If we accept this point of view then we consider

$$p(\mathbf{y}) \approx N\left(\mathbf{m}|\mathbf{0}, \mathbf{K} + \mathbf{B}^{-1}\right), \tag{16}$$

where $\mathbf{m}$ and $\boldsymbol{\beta}$ are the means and precisions associated with the implied Gaussians, to be a valid approximation to the marginal likelihood. The individual site mean, $m_n$, and precision, $\beta_n$, associated with each 'apparent Gaussian noise model' can be computed given that noise model's values for $g_{in}$ and $\nu_{in}$. By rearranging (9) and (10) and replacing $y_n$ with $m_n$ we have

$$g_i = \frac{m_n - \mu_{i-1,n}}{\beta_n^{-1} + \varsigma_{i-1,n}}$$

$$g_i = \frac{m_n - \mu_{i-1,n}}{\nu_{in}}$$

$$m_n = \frac{g_{in}}{\nu_{in}} + \mu_{i-1,n} \tag{17}$$

and

$$\begin{aligned}\nu_{in} &= \frac{1}{\varsigma_{i-1,n} + \beta_n^{-1}} \\ \beta_n^{-1} &= \frac{1}{\nu_{in}} - \varsigma_{i-1,n} \\ \beta_n &= \frac{\nu_{in}}{1 - \nu_{in}\varsigma_{i-1,n}}.\end{aligned} \tag{18}$$

which give the site parameters for a given noise model. Note that if $\nu_{in}\varsigma_{i-1,n} > 1$ the site precision, $\beta_n$, becomes negative. This can only occur if the noise model is not log concave—we shall encounter such a noise model in Section 4.

## 3 The Informative Vector Machine

One advantage of the Gaussian process perspective referred to in the introduction is automatic selection of the kernel parameters through likelihood optimisation. In practise though gradients with respect to the parameters must be computed. Computation of these gradients involves an inverse of the kernel matrix, $\mathbf{K}$, at each step. This matrix inverse gives each gradient step $O\left(N^3\right)$ complexity. Even if the kernel parameters are given including $N$ data points through ADF still leads to an $O\left(N^3\right)$ complexity. The IVM algorithm seeks to resolve these problems by seeking a sparse representation for the data set. The inspiration for this approach is the support vector machine, a kernel algorithm for which the solution is typically naturally sparse. This natural sparsity can arise because the SVM algorithm ignores the process variances, however these process variances must be accounted for if optimisation of kernel parameters via empirical Bayes is undertaken. Csató and Opper suggest obtaining a sparse representation through minimisation of KL divergences between the approximate posterior and a sparse representation [8,7]. The IVM takes a simpler approach of using a selection heuristic to incorporate only the most informative points and stopping after $d$ inclusions [14,21]. By making only $d$ inclusions the resulting model is forced to be sparse. As a result we can reduce the computational requirements of the algorithm from $O\left(N^3\right)$ to $O\left(d^2N\right)$ and the storage requirements form $O\left(N^2\right)$ to $O\left(dN\right)$. In the IVM this 'sparsification' of the original Gaussian process is imposed by only selecting a subset of the data. The ADF algorithm allows us to select this subset in an on-line way, given a data point selection criterion. The IVM uses an information theoretic selection heuristic to select which data point to include. Specifically, the point that gives the largest reduction in posterior process entropy is included. In this way the entropy of the posterior process is greedily minimised.

### 3.1 Data Point Selection

The simple greedy selection criterion on which the IVM is based is inspired by information theory. The change in entropy of the posterior process after including a point can be seen as a measure of reduction in the level of uncertainty. This entropy reduction can only be evaluated because we are propagating the posterior covariance function. Loosely speaking, the entropy of the posterior process is analogous to the version space in the SVM. Greedily minimising the entropy can be seen as slicing the largest possible section away from version space and tracking the posterior covariance can be seen as maintaining a representation of the size and shape of version space while the model is being constructed.

The entropy change associated with including the $n$th point at the $i$th inclusion is given by

$$\begin{aligned} \Delta H_{in} &= -\frac{1}{2}\log\left|\Sigma_{i,n}\right| + \frac{1}{2}\log\left|\Sigma_{i-1}\right| \\ &= -\frac{1}{2}\log\left|\mathbf{I} - \Sigma_{i-1}\mathrm{diag}\left(\boldsymbol{\nu}_i\right)\right| \\ &= -\frac{1}{2}\log\left(1 - \nu_{in}\varsigma_{i-1,n}\right). \end{aligned} \tag{19}$$

This entropy change can be evaluated for every point in the inactive set, $J$, and the point associated with the largest entropy decrease can then be included. This process is repeated until $d$ inclusions have been made. 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 [15,22], however in active learning the label of the data point, $y_n$, is assumed to be unknown before selection: here the label is known and can strongly influence whether a particular point is selected.

We note from (19) that in order to score each point we need to keep track of the diagonal terms from $\Sigma_{i-1}$ and the values $\nu_{in}$. If these terms can be stored and updated efficiently then the computational complexity of the algorithm can be controlled.

Maintaining the whole of $\Sigma_{i-1}$ in memory would require $O\left(N^2\right)$ storage, which is undesirable, so our first task is to seek an efficient representation of the posterior covariance.

From (12) it is clear the posterior covariance $\Sigma_i$ has a particular structure that arises from the addition of successive outer products to the original prior covariance matrix $\Sigma_0 = \mathbf{K}$. This can be re-written as

$$\Sigma_i = \mathbf{K} - \mathbf{M}_i^{\mathrm{T}}\mathbf{M}_i \tag{20}$$

where the $k$th row of $\mathbf{M}_i \in \Re^{i\times N}$ 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 $\Sigma_{i-1}$ we note that, if we are not storing $\Sigma_{i-1}$, we are not able to represent (for instance) $\mathbf{s}_{i-1,n_i}$ directly. However, this column can be recomputed from $\mathbf{M}_{i-1}$ and $\mathbf{K}$ by

$$\mathbf{s}_{i-1,n_i} = \mathbf{k}_{n_i} - \mathbf{a}_{i-1,n_i}^{\mathrm{T}}\mathbf{M}_{i-1} \tag{21}$$

where $\mathbf{k}_{n_i} = \mathbf{s}_{0,n_i}$ is the $n_i$th column of $\mathbf{K} = \Sigma_0$ and $\mathbf{a}_{i-1,n_i}$ is the $n_i$th column of $\mathbf{M}_{i-1}$. This recomputation requires $O\left(\left(i-1\right)N\right)$ operations.

### 3.2 The Matrix $\mathbf{M}_i$

Let us consider more closely what the matrix $\mathbf{M}_i$ represents. It is straightforward to show that a Gaussian process which has included an active set $I$ with $i$ elements has a posterior covariance of the form

$$\Sigma_i = \mathbf{K} - \mathbf{K}_{:,I}\left(\mathbf{B}_I^{-1} + \mathbf{K}_I\right)^{-1}\mathbf{K}_{I,:} \tag{22}$$

where $\mathbf{K}_{:,I}$ is a matrix containing the columns of $\mathbf{K}$ that are in the active set, $\mathbf{K}_I$ is an $i \times i$ symmetric matrix containing the rows and columns from $\mathbf{K}$ that are in $I$ and $\mathbf{B}_I$ is a diagonal matrix of the 'site precisions' for the active set. Equating with (20) we note that $\mathbf{M}_i^{\mathrm{T}}\mathbf{M}_i = \mathbf{K}_{:,I}\left(\mathbf{B}_I^{-1} + \mathbf{K}_I\right)^{-1}\mathbf{K}_{I,:}$ or

$$\mathbf{L}_i^{\mathrm{T}}\mathbf{M}_i = \mathbf{K}_{I,:}$$

where $\mathbf{L}_i\mathbf{L}_i^{\mathrm{T}} = \left(\mathbf{B}_I^{-1} + \mathbf{K}_I\right)^{-1}$.

From the way that the updates in (12) accumulate it can be shown that a valid factorisation of the matrix $\mathbf{L}_i\mathbf{L}_i^{\mathrm{T}}$ is given by the lower triangular matrix

$$\mathbf{L}_i = \begin{bmatrix} \mathbf{L}_{i-1} & \mathbf{0} \\ \mathbf{a}_{i-1,n_i}^{\mathrm{T}} & \nu_{i,n_i}^{-\frac{1}{2}} \end{bmatrix}, \tag{23}$$

where $\mathbf{L}_1 = \nu_{i,n_1}^{-\frac{1}{2}}$. This lower triangular matrix is recognised as a Cholesky factor of $\left(\mathbf{B}_I^{-1} + \mathbf{K}_I\right)^{-1}$. We may gain some reassurance from the implicit presence of a Cholesky decomposition within the algorithm as they are typically considered to be a numerically stable. Note however that in [21,14] a slightly different representation is suggested. Instead of keeping track of the Cholesky decomposition of $\left(\mathbf{B}_I^{-1} + \mathbf{K}_I\right)$, the decomposition of $\left(\mathbf{I} + \mathbf{B}_I^{-\frac{1}{2}}\mathbf{K}_I\mathbf{B}_I^{-\frac{1}{2}}\right)$ is used to give greater numerical stability when $\mathbf{K}_I$ is not full rank and when some values of $\mathbf{B}_I^{-1}$ are close to zero. Here we prefer our representation as it arises naturally and keeps the update equations clear.

Note that the matrix $\mathbf{L_i}$ is only a valid lower Cholesky factor if $\nu_{i,n_i}$ is non-negative. This can be shown to hold true if the noise model is log-concave. Complications arise if the noise model is not log-concave; a simple strategy for avoiding this problem is suggested in Section 4.

We have already stated that storing the entire posterior covariance matrix is impractical because of memory requirements. However, maintaining an estimate of the process variance associated with each data point is of interest. These variances are the diagonal elements from the posterior covariance matrix, $\boldsymbol{\varsigma}_i = \operatorname{diag}\left(\Sigma_i\right)$. A vector storing these variances can be updated easily using (11), (12) and (21) to give,

$$\boldsymbol{\varsigma}_i = \boldsymbol{\varsigma}_{i-1} - \nu_{i,n_i}\operatorname{diag}\left(\mathbf{s}_{i-1,n_i}\mathbf{s}_{i-1,n_i}^{\mathrm{T}}\right) \tag{24}$$

which is computed in $O\left(N\right)$ operations. The posterior mean output vector should also be stored and updated using

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + g_{in_i}\mathbf{s}_{i-1,n_j} \tag{25}$$

**Algorithm 1** The IVM point selection algorithm.

```
Require: Require $d$ = number of active points. Start with $\mathbf{m} = \mathbf{0}$ and $\boldsymbol{\beta} = \mathbf{0}$ for classification. For regression substitute appropriate target values. Take $\varsigma_0 = \mathrm{diag}\left(\mathbf{K}\right)$, $\boldsymbol{\mu} = \mathbf{0}$, $J = \{1, \ldots, N\}$, $I = \{\cdot\}$, $\mathbf{S}_0$ is an empty matrix.
  for $k = 1\,\mathrm{to}\,d$ do
    for all $n \in J$ do
      compute $g_{kn}$ according to (14) (not required for Gaussian).
      compute $\nu_{kn}$ according to (15) or (10).
      compute $\Delta H_{kn}$ according to (19).
    end for
    $n_k = \mathrm{argmax}_{n \in J} \Delta H_{kn}$.
    Update $m_{n_k}$ and $\beta_{n_k}$ using (17) and (18).
    Compute $\varsigma_k$ and $\boldsymbol{\mu}_k$ using (21), (24) and (25).
    Append $\sqrt{\nu_{kn_k}}\mathbf{s}_{k-1,n_k}^{\mathrm{T}}$ to $\mathbf{M}_{k-1}$ using (21) to form $\mathbf{M}_k$.
    Update $\mathbf{L}_k$ from $\mathbf{L}_{k-1}$ using (23).
    Add $n_k$ to $I$ and remove $n_k$ from $J$.
  end for
```

which also requires $O\left(N\right)$ operations.

An example of how these updates may be combined efficiently in practise is given in Algorithm 1.

### 3.3 IVM Kernel Parameter Updates

The ADF-based IVM algorithm described in Algorithm 1 leads to sparse vectors $\mathbf{m}$ and $\boldsymbol{\beta}$ each with $d$ non-zero elements. In Section 2.5, we described how kernel parameters could be updated given non-sparse vectors of these site parameters. To maximise kernel parameters for the IVM we need to express the likelihood in terms of sparse versions of these vectors. In [21] this is achieved by expressing the likelihood in terms of both the active and inactive sets. Here we take a much simpler approach of maximising the likelihood of the active points only,

$$p\left(\mathbf{y}_I|\boldsymbol{\theta}\right) \approx N\left(\mathbf{m}_I|\mathbf{0}, \mathbf{K}_I + \mathbf{B}_I^{-1}\right), \tag{26}$$

where $\mathbf{y}_I$ is a vector containing only those elements of $\mathbf{y}$ that are in the active set. The dependence of the likelihood on the parameters, $\boldsymbol{\theta}$, is through the active portion of the kernel 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 practise, the quality of the active set depends on the kernel parameter selection as does the optimal site parameters. We can certainly imagine more complex schema for optimising the kernel parameters that take account of these dependencies in a better way, some examples of which are given in [21], but for our experiments we simply iterated between active set selection and kernel parameter optimisation.

### 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. However, the likelihood approximation (26) is only indirectly dependent on those parameter values so cannot be used as an objective function. One way forward would be to optimise a variational lower bound,

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

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

$$\sum_{n=1}^{N} \int q_d\left(f_n\right) \log p\left(y_n | f_n, \boldsymbol{\theta}\right). \tag{27}$$

This lower bound can be upper bounded by

$$\begin{aligned} \mathcal{L}\left(\theta\right) &= \sum_{n=1}^{N} \log \int q_d\left(f_n\right) p\left(y_n | f_n, \boldsymbol{\theta}\right) \\ &= \sum_{n=1}^{N} \log Z_n. \end{aligned} \tag{28}$$

For many models it is straightforward to compute (27), however for all noise models it is possible to compute (28). We found that, for an ordered categorical noise model (where there are an atypically large number of noise model parameters), optimisation of (28) was sufficient.

### 3.5 Point Removal and Expectation Propagation

The sequential nature of the ADF algorithm leads to a weakness in the approximation for which a fairly straightforward fix has been suggested [16]. When the Gaussian approximation to the likelihood after $i$ inclusions is computed, its associated site parameters, $\beta_{n_i}$ and $m_{n_i}$ are based on only the $i-1$ points that are already in the active set. However as more points are included and the posterior approximation evolves it is likely that the quality of this approximation becomes worse. The approach suggested by [16] is to revisit the approximation and improve it. This refinement of the ADF algorithm is known as *expectation propagation* (EP). Conceptually one can view these later updates as removing a point from the active set (by resetting the associated site parameters to zero) and then computing a new Gaussian approximation in the light of the current (and usually more accurate) representation of the posterior covariance. The key issue for the algorithm is, therefore, to be able remove a point in an efficient manner.

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

```
Require: Require $d$ active points. $T$ iterations.
  for $i = 1 \,\mathrm{to}\, T$ do
    Select points using Algorithm 1.
    Maximise the approximation to the likelihood (26) using a scaled conjugate gradient optimiser [17].
    if noise parameter updates are required. then
      Select points using Algorithm 1.
      Maximise the sum of the log partition functions (28) using a scaled conjugate gradient optimiser.
    end if
  end for
```

Even if the expectation propagation algorithm is not being used, it may be desirable to remove points within the IVM algorithm as it is likely to be the case that points that were included in the early stages of the algorithm, when the estimate of the posterior's covariance and mean functions were poor, are less useful than they originally appeared. Inclusion of such points is a natural consequence of greedily selecting points to reduce entropy. Some approaches to the implementation of point removal and expectation propagation with the IVM are further discussed in [21].

### 3.6 IVM Implementation

In the experimental sections that follow we make use of the IVM algorithm for learning. For all these experiments we used the kernel parameter initialisations and the types of kernels detailed in Appendix B. The algorithm we used for point selection is specified in Algorithm 1 and when optimisation was required we typically made use of 8 iterations of Algorithm 2.

## 4 Semi-supervised Learning

In a traditional classification problem, data are presented in the form of a set of input vectors $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_N]^{\mathrm{T}}$ and associated labels $\mathbf{y} = [y_1 \ldots y_N]^{\mathrm{T}}$, $y_n \in \{-1, 1\}$. When performing classification the IVM algorithm seeks a process mapping between the inputs and the labels that yields high predictive accuracy. This is known as *supervised learning*, as each input data point, $\mathbf{x}_n$, has an associated label, $y_n$. In practice, labelling a data set can be a time consuming process; very often it is straightforward to obtain input data points, $\mathbf{x}_n$, but expensive to obtain an associated label $y_n$. It is natural then to consider whether unlabelled data points can be used to improve predictive performance. Fitting models using such partially labelled data sets is known as 'semi-supervised learning'.

Most probabilistic classification algorithms can be categorised as either discriminative or generative methods. Discriminative methods, such as the IVM, estimate the probability of class membership, $p\left(y_n|\mathbf{x}_n\right)$ directly. Generative methods typically model the class-conditional densities, $p\left(\mathbf{x}_n|y_n\right)$ and use them in combination with an estimate of the class priors to infer $p(y_n|\mathbf{x}_n)$ through Bayes' theorem. In the former case, if we fail to make any assumptions about the underlying distribution of input data, the unlabelled data does not affect our predictions. Thus, most attempts to make use of unlabelled data within a probabilistic framework focus on incorporating a model of $p(\mathbf{x}_n)$; for example, by treating it as a mixture, $\sum_{y_n} p(\mathbf{x}_n|y_n)\, p(y_n)$, and inferring $p(y_n|\mathbf{x}_n)$ (e.g., [13]), or by building kernels based on $p(\mathbf{x}_n)$ (e.g., [20]). These approaches can be unwieldy, however, in that the complexities of the input distribution are typically of little interest when performing classification, so that much of the effort spent modelling $p(\mathbf{x}_n)$ may be wasted.

An alternative is to make weaker assumptions regarding $p(\mathbf{x}_n)$ that are of particular relevance to classification. In particular, the *cluster assumption* asserts that the data density should be reduced in the vicinity of a decision boundary (e.g., [5]). Such a qualitative assumption is readily implemented within the context of non-probabilistic kernel-based classifiers. Here we discuss how the same assumption can be incorporated within the IVM algorithm [11].

Our approach involves a notion of a 'null category region', a region that acts to exclude unlabelled data points. Such a region is analogous to the traditional notion of a 'margin' and indeed our approach is similar in spirit to the transductive SVM [26], which seeks to maximise the margin by allocating labels to the unlabelled data. A major difference, however, is that through our use of the IVM algorithm our approach maintains and updates the process variance. As we will see, this variance turns out to interact in a significant way with the null category concept.

### 4.1 Null Category Noise Model

Before considering the null category noise model (NCNM) we first briefly review ordered categorical models that underpin the NCNM. In the specific context of binary classification, we consider an ordered categorical model containing three categories[6].

$$p(y_n|f_n) = \begin{cases} \phi\left(-\left(f_n + \frac{w}{2}\right)\right) & \text{for } y_n = -1 \\ \phi\left(f_n + \frac{w}{2}\right) - \phi\left(f_n - \frac{w}{2}\right) & \text{for } y_n = 0 \\ \phi\left(f_n - \frac{w}{2}\right) & \text{for } y_n = 1 \end{cases},$$

where $\phi(x) = \int_{-\infty}^{x} N(z|0,1)\, dz$ is the cumulative Gaussian distribution function and $w$ is a parameter giving the width of category $y_n = 0$ (see Figure 2).

We can also express this model in an equivalent and simpler form by replacing the cumulative Gaussian distribution by a Heaviside step function $H(\cdot)$ and adding independent Gaussian noise to the process model:

$$p(y_n|f_n) = \begin{cases} H\left(-\left(f_n + \frac{1}{2}\right)\right) & \text{for } y_n = -1 \\ H\left(f_n + \frac{1}{2}\right) - H\left(f_n - \frac{1}{2}\right) & \text{for } y_n = 0 \\ H\left(f_n - \frac{1}{2}\right) & \text{for } y_n = 1 \end{cases},$$

[6] See also [23] that makes use of a similar noise model in a discussion of Bayesian interpretations of the SVM.

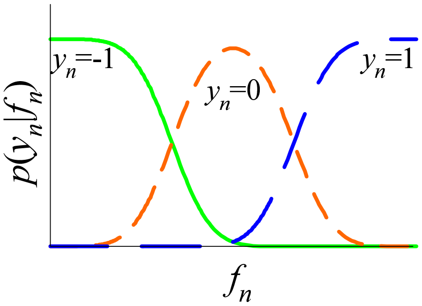


**Figure 2.** The ordered categorical noise model. The plot shows $p\left(y_n|f_n\right)$ for different values of $y_n$. Here we have assumed three categories.

where we have standardised the width parameter to 1, by assuming that the overall scale is also handled by the process model.

To use this model in an unlabelled setting, we introduce a further variable, $z_n$, that is one if a data point is unlabelled and zero otherwise. We first impose

$$p\left(z_n = 1|y_n = 0\right) = 0; \tag{29}$$

in other words, a data point can not be from the category $y_n = 0$ and be unlabelled. We assign probabilities of missing labels to the other classes

$$\begin{aligned} p\left(z_n = 1|y_n = 1\right) = \gamma_+ \quad \text{and} \\ p\left(z_n = 1|y_n = -1\right) = \gamma_- . \end{aligned}$$

We see from the graphical representation in Figure 3 that $z_n$ is $d$-separated from $\mathbf{x}_n$. Thus when $y_n$ is observed, the posterior process is updated by using $p\left(y_n|f_n\right)$. On the other hand, when the data point is unlabelled the posterior process must be updated by $p\left(z_n|f_n\right)$ which is easily computed as:

$$p\left(z_n = 1|f_n\right) = \sum_{y_n} p\left(y_n|f_n\right) p\left(z_n = 1|y_n\right).$$

The "effective likelihood function" for a single data point, $L\left(f_n\right)$, therefore takes one of three forms:

$$L\left(f_n\right) = \left\{ \begin{array}{cl} H\left(-\left(f_n + \frac{1}{2}\right)\right) & \text{for } y_n = -1,\; z_n = 0 \\ \gamma_- H\left(-\left(f_n + \frac{1}{2}\right)\right) + \gamma_+ H\left(f_n - \frac{1}{2}\right) & \text{for } \qquad z_n = 1 \\ H\left(f_n - \frac{1}{2}\right) & \text{for } \quad y_n = 1\; z_n = 0 \end{array} \right. .$$

The constraint imposed by (29) implies that an unlabelled data point never comes from the class $y_n = 0$. Since $y_n = 0$ lies between the labelled classes this is equivalent to a hard assumption that no data comes from the region around

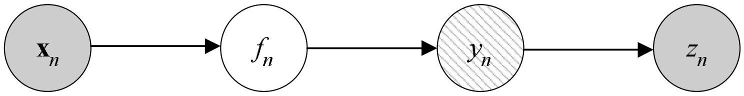


**Figure 3.** Graphical representation of the null category model. The fully-shaded nodes are always observed, whereas the lightly-shaded node is observed when $z_n = 0$.

the decision boundary. We can also soften this hard assumption if so desired by injection of noise into the process model. If we also assume that our labelled data only comes from the classes $y_n = 1$ and $y_n = -1$ we never obtain any evidence for data with $y_n = 0$; for this reason we refer to this category as the *null category* and the overall model as a *null category noise model* (NCNM).

### 4.2 Process Model and Effect of the Null Category

To work within the Gaussian process framework we take

$$p\left(f_n|\mathbf{x}_n\right) = N\left(f_n|\mu\left(\mathbf{x}_n\right), \varsigma\left(\mathbf{x}_n\right)\right),$$

where the mean $\mu\left(\mathbf{x}_n\right)$ and the variance $\varsigma\left(\mathbf{x}_n\right)$ are functions of the input vector. A natural consideration in this setting is the effect of our likelihood function on the distribution over $f_n$ from incorporating a new data point. First we note that if $y_n \in \{-1, 1\}$ the effect of the likelihood is similar to that incurred in binary classification, in that the posterior is a convolution of the step function and a Gaussian distribution. This is comforting as when a data point is labelled the model acts in a similar manner to a standard binary classification model. Consider now the case when the data point is unlabelled. The effect of the data point depends on the mean and variance of $p\left(f_n|\mathbf{x}_n\right)$. If this Gaussian has little mass in the null category region, the posterior is similar to the prior. However, if the Gaussian has significant mass in the null category region, the outcome may be loosely described in two ways:

1. If $p\left(f_n|\mathbf{x}_n\right)$ 'spans the likelihood', Figure 4 (Left), then the mass of the posterior can be apportioned to either side of the null category region, leading to a bimodal posterior. The variance of the posterior is greater than the variance of the prior, a consequence of the fact that the effective likelihood function is not log-concave (as can be easily verified).
2. If $p\left(f_n|\mathbf{x}_n\right)$ is 'rectified by the likelihood', Figure 4 (Right), then the mass of the posterior is pushed in to one side of the null category and the variance of the posterior is smaller than the variance of the prior.

Note that for all situations when a portion of the mass of the prior distribution falls within the null category region it is pushed out to one side or both sides. The intuition behind the two situations is that in case 1, it is not clear what label the data point has, however it is clear that it shouldn't be where it currently is (in the null category). The result is that the process variance increases. In case 2, the data point is being assigned a label and the decision boundary is pushed to one side of the point so that it is classified according to the assigned label.

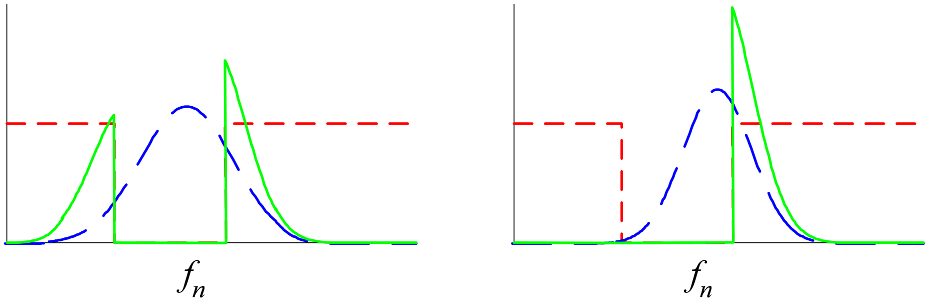


**Figure 4.** Two situations of interest. Diagrams show the prior distribution over $f_n$ (long dashes) the effective likelihood function from the noise model when $z_n = 1$ (short dashes) and a schematic of the resulting posterior over $f_n$ (solid line). *Left*: The posterior is bimodal and has a larger variance than the prior. *Right*: The posterior has one dominant mode and a lower variance than the prior. In both cases the process is pushed away from the null category.

### 4.3 Posterior Inference and Prediction

Recall from Section 4.2 that the noise model is not log-concave. When the variance of the process increases the best Gaussian approximation to our noise model can have negative variance. This has an important implication: the site precision can be computed as a negative value (see Section 2.5). This situation is discussed in [16], where various suggestions are given to cope with the issue. For the IVM we followed the simplest suggestion: we set a negative variance to zero. As we have discussed, one important advantage of the Gaussian process framework is that hyperparameters in the covariance function (i.e., the kernel function), can be fit by optimising the marginal likelihood. In practise, however, if the process variance is maximised in an unconstrained manner the effective width of the null category can be driven to zero, yielding a model that is equivalent to a standard binary classification noise model[7]. To prevent this from happening we regularise with an $L_1$ penalty on the process variances (this is equivalent to placing an exponential prior on those parameters).

### 4.4 Semi-supervised Learning: Toy Problem

In all our experiments we used an RBF kernel with a white noise term (see Appendix B).

[7] Recall, as discussed in Section 2, that we fix the width of the null category to unity: changes in the scale of the process model are equivalent to changing this width.

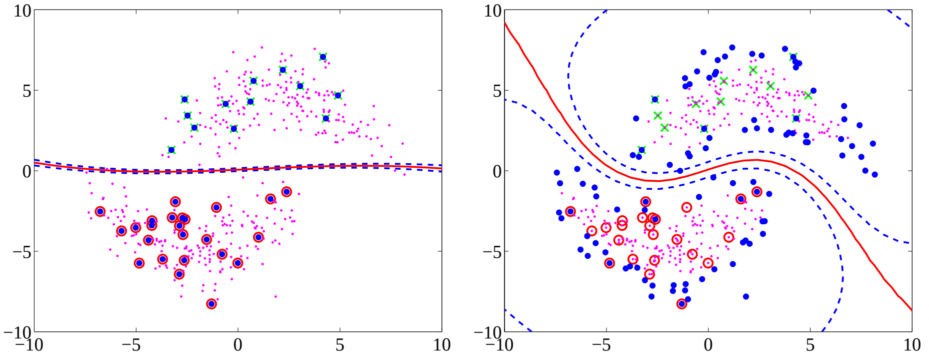


**Figure 5.** Results from the toy problem. There are 400 points that are labelled with probability 0.1. Labelled data points are shown as circles and crosses. data points in the active set are shown as large dots. All other data points are shown as small dots. *Left*: Learning on the labelled data only with the IVM algorithm. All labelled points are used in the active set. *Right*: Learning on the labelled and unlabelled data with the NCNM. There are 100 points in the active set. In both plots, decision boundaries are shown as a solid line; dotted lines represent contours within 0.5 of the decision boundary (for the NCNM this is the edge of the null category).

We made use of Algorithm 2 without the optional noise parameter update. To ensure that the width of the null category region wasn't collapsing with repeated iteration of the algorithm we used 15 iterations (typically the algorithm found a good solution after only 4). The parameters of the noise model, $\{\gamma_+, \gamma_-\}$ could also be optimised, but note that if we constrain $\gamma_+ = \gamma_- = \gamma$ then the likelihood is maximised by setting $\gamma$ to the proportion of the training set that is unlabelled.

We first considered an illustrative toy problem to demonstrate the capabilities of our model. We generated two-dimensional data in which two class-conditional densities interlock. There were 400 points in the original data set. Each point was labelled with probability 0.1, leading to 37 labelled points. First a standard IVM classifier was trained on the labelled data only (Figure 5, Left). We then used the null category approach to train a classifier that incorporates the unlabelled data. As shown in Figure 5 (Right), the resulting decision boundary finds a region of low data density and more accurately reflects the underlying data distribution.

### 4.5 Semi-supervised Learning: USPS Digits

We next considered the null category noise model for learning of USPS handwritten digit data set. This data set is fully labelled, but we can ignore a proportion of the labels and treat the data set as a semi-supervised task. In the experiments that followed we used an RBF kernel with a linear component. We ran each experiment ten times, randomly selecting the data points that were labelled. The fraction of labelled points, $r$, was varied between 0.01 and 0.25. Each digit was treated as a separate 'one against the others' binary classification class. We also summarised these binary classification tasks in the standard way with an overall error rate. In the first of our experiments, we attempted to learn the parameters of the kernel using 8 iterations of Algorithm 2 to learn these parameters. The results are summarised in Table 1.

| $r$ | **0** | **1** | **2** | **3** | **4** | |
|---|---|---|---|---|---|---|
| 0.010 | 17.9 ± 0.00 | 7.99 ± 6.48 | 9.87 ± 0.00 | 8.27 ± 0.00 | 9.97 ± 0.00 | |
| 0.025 | 11.4 ± 8.81 | 0.977 ± 0.10 | 9.87 ± 0.00 | 6.51 ± 2.43 | 9.97 ± 0.00 | |
| 0.050 | 1.72 ± 0.21 | 1.01 ± 0.10 | 3.66 ± 0.40 | 5.35 ± 2.67 | 7.41 ± 3.50 | |
| 0.10 | 1.73 ± 0.14 | 0.947 ± 0.14 | 3.17 ± 0.23 | 3.24 ± 0.30 | 3.33 ± 0.30 | |
| 0.25 | 1.63 ± 0.16 | 0.967 ± 0.09 | 2.50 ± 0.18 | 2.90 ± 0.20 | 2.77 ± 0.08 | |
| $r$ | **5** | **6** | **7** | **8** | **9** | **Overall** |
| 0.010 | 7.97 ± 0.00 | 8.47 ± 0.00 | 7.32 ± 0.00 | 8.27 ± 0.00 | 8.82 ± 0.00 | 83.3 ± 7.30 |
| 0.025 | 7.97 ± 0.00 | 8.47 ± 0.00 | 7.32 ± 0.00 | 8.27 ± 0.00 | 8.82 ± 0.00 | 64.2 ± 5.08 |
| 0.05 | 7.11 ± 1.94 | 1.69 ± 0.15 | 7.32 ± 0.00 | 7.42 ± 1.89 | 7.60 ± 2.72 | 33.3 ± 7.15 |
| 0.1 | 3.02 ± 0.27 | 1.48 ± 0.05 | 1.32 ± 0.08 | 3.40 ± 0.15 | 1.95 ± 0.25 | 7.67 ± 0.19 |
| 0.25 | 2.38 ± 0.19 | 1.25 ± 0.17 | 1.19 ± 0.06 | 2.61 ± 0.25 | 1.59 ± 0.15 | 6.44 ± 0.22 |

**Table 1.** Table of results for semi-supervised learning on the USPS digit data. For these results the model learned the kernel parameters. We give the results for the individual binary classification tasks and the overall error computed from the combined classifiers. Each result is summarised by the mean and standard deviation of the percent classification error across ten runs with different random seeds.

It is immediately apparent that for values of $r$ below 0.1 a sensible decision boundary is not found for many of the binary classification tasks. At first sight, this reflects badly on the approach: many semi-supervised learning algorithms give excellent performance even when the proportion of unlabelled data is so low. However here it must be bourne in mind that the algorithm has the additional burden of learning the kernel parameters. Most other approaches do not have this capability and therefore results are typically reported for a given set of kernel parameters. For this reason we also undertook experiments using kernel parameters that were precomputed by an IVM trained on the fully labelled data set. These results are summarised in Table 2. As expected these results are much better in the range where $r < 0.1$. With the exception of the digit 2 at $r = 0.01$ a sensible decision boundary was learned for at least one of the runs even when $r = 0.01$.

We would certainly expect the results to be better in this second experiment as we are providing more information (in terms of precomputed kernel parameters) to the model. However it is very encouraging that for $r > 0.1$ the results of both experiments are similar.

| $r$ | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **Overall** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.010 | $3.17 \pm 5.17$ | $13.3 \pm 14.18$ | $9.87 \pm 0.00$ | $3.09 \pm 0.22$ | $8.32 \pm 2.63$ | $7.50 \pm 0.99$ | $7.70 \pm 8.48$ | $12.3 \pm 16.91$ | $7.48 \pm 1.23$ | $35.2 \pm 22.90$ | $42.3 \pm 10.05$ |
| 0.025 | $1.50 \pm 0.18$ | $1.52 \pm 0.87$ | $5.18 \pm 1.95$ | $2.90 \pm 0.19$ | $4.36 \pm 2.08$ | $5.03 \pm 1.30$ | $1.60 \pm 0.15$ | $1.93 \pm 1.90$ | $4.32 \pm 0.45$ | $9.93 \pm 8.51$ | $140 \pm 6.06$ |
| 0.050 | $1.50 \pm 0.15$ | $1.24 \pm 0.22$ | $3.37 \pm 0.35$ | $2.85 \pm 0.14$ | $3.27 \pm 0.19$ | $3.63 \pm 0.56$ | $1.49 \pm 0.11$ | $1.28 \pm 0.09$ | $4.07 \pm 0.43$ | $2.59 \pm 1.33$ | $8.43 \pm 0.66$ |
| 0.10 | $1.46 \pm 0.09$ | $1.24 \pm 0.13$ | $2.82 \pm 0.16$ | $2.75 \pm 0.19$ | $3.07 \pm 0.22$ | $2.75 \pm 0.22$ | $1.30 \pm 0.10$ | $1.30 \pm 0.14$ | $3.48 \pm 0.26$ | $1.97 \pm 0.24$ | $7.24 \pm 0.51$ |
| 0.25 | $1.40 \pm 0.15$ | $1.31 \pm 0.16$ | $2.44 \pm 0.21$ | $2.61 \pm 0.19$ | $2.80 \pm 0.16$ | $2.32 \pm 0.19$ | $1.15 \pm 0.10$ | $1.15 \pm 0.05$ | $2.74 \pm 0.19$ | $1.62 \pm 0.16$ | $6.10 \pm 0.41$ |

**Table 2.** Table of results for semi-supervised learning on the USPS digit data. For these results the model was given the kernel parameters learnt by the IVM on the standard fully labelled data. We give the results for the individual binary classification tasks and the overall error computed from the combined classifiers.

## 5 Multi-task Learning

In this section, we show how the IVM approach can be extended to handle the situation where we have multiple independent tasks (see also [12]).

The idea behind multi-task learning is that the tasks are related and that an algorithm that makes use of these relationships may allow us to adapt to an additional task with very little training data [1,24,4].

From a Bayesian perspective the multi-task learning problem can be approached as an instance of the general hierarchical Bayesian approach to sharing strength among related statistical models [9,10]. The component models ("tasks") are assumed to be independent given an underlying latent variable; marginalising across this variable couples these components. Inferentially, learning about one task updates the posterior distribution for the latent variable, which provides a sharpened prior distribution for learning a subsequent task.

In our setting, one natural hierarchical model to consider is a model in which there are $M$ Gaussian process models, $p(\mathbf{y}_m|\mathbf{X}_m, \boldsymbol{\theta})$, for $m = 1, \ldots, M$, and in which the kernel hyperparameter $\boldsymbol{\theta}$ is shared among the component models. Conditional on $\boldsymbol{\theta}$ the component models are assumed independent. While in a full Bayesian approach we would place a prior distribution on $\boldsymbol{\theta}$ and integrate over its posterior, in the current section we again make the empirical Bayesian approximation and find a single best-fitting value of $\boldsymbol{\theta}$ by maximising the marginal likelihood.

This setup is depicted as a graphical model in Figure 6. We model the output data for each task, $\mathbf{y}_m$, as a GP so that the probability distribution for the matrix $\mathbf{Y}$, whose columns are $\mathbf{y}_m$, is

$$p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{m=1}^{M} p(\mathbf{y}_m|\mathbf{X}_m, \boldsymbol{\theta})$$

where each $p\left(\mathbf{y}_m|\mathbf{X}_m,\boldsymbol{\theta}\right)$ is a Gaussian process.

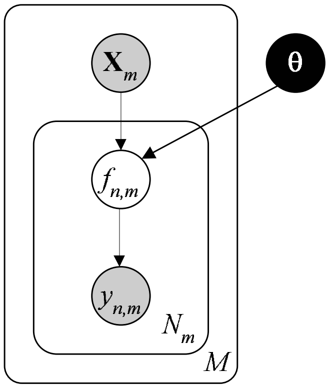


**Figure 6.** A graphical model that represents a multi-task GP, compare with the single task GP in Figure 1.

The overall likelihood can be considered to be a Gaussian process over a vector $\mathbf{y}$ that is formed by stacking columns of $\mathbf{Y}$, $\mathbf{y}=\left[\mathbf{y}_1^{\mathrm{T}}\ldots\mathbf{y}_M^{\mathrm{T}}\right]^{\mathrm{T}}$. The covariance matrix is then

$$\mathbf{K}=\begin{bmatrix}\mathbf{K}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{K}_2 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{K}_M\end{bmatrix}$$

and we write

$$p\left(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}\right)=N\left(\mathbf{0},\mathbf{K}\right). \tag{30}$$

This establishes the equivalence of the multi-task GP to a standard GP. Once again we obtain an estimate, $\hat{\boldsymbol{\theta}}$, of the parameters by optimising the log-likelihood with respect to the kernel parameters. These gradients require the inverse of $\mathbf{K}$ and, while we can take advantage of its block diagonal structure to compute this inverse, we are still faced with inverting $N_m \times N_m$ matrices, where $N_m$ is the number of data points associated with task $m$: however we can look to the IVM algorithm to sparsify the GP specified by (30).

It is straightforward to show that the new posterior covariance structure after $k$ inclusions, $q_k\left(\mathbf{f}\right)$, also is a Gaussian with a block-diagonal covariance matrix,

$$q_k\left(\mathbf{f}\right)=\prod_{m=1}^{M}N\left(\mathbf{f}_m|\boldsymbol{\mu}_i^{(m)},\varSigma_i^{(m)}\right). \tag{31}$$

Note that $k$ inclusions in total does not mean $k$ inclusions for each task. Each block of the posterior covariance is

$$\Sigma_i^{(m)} = \mathbf{K}_m - \mathbf{M}_i^{(m)\,\mathrm{T}} \mathbf{M}_i^{(m)},$$

where the rows of $\mathbf{M}_i^{(m)}$ are given by $\sqrt{\nu_{in_i}^{(m)}}\mathbf{s}_{i-1,n_i}^{(m)}$. The means associated with each task are given by

$$\boldsymbol{\mu}_i^{(m)} = \boldsymbol{\mu}_{i-1}^{(m)} + g_{in_i}^{(m)} \mathbf{s}_{i-1,n_i}^{(m)} \tag{32}$$

and updates of $\boldsymbol{\varsigma}_i^{(m)}$ can still be achieved through

$$\boldsymbol{\varsigma}_i^{(m)} = \boldsymbol{\varsigma}_{i-1}^{(m)} - \nu_{i,n_i}^{(m)} \mathrm{diag}\left(\mathbf{s}_{i-1,n_i}^{(m)}\ \mathbf{s}_{i-1,n_i}^{(m)}{}^{\mathrm{T}}\right). \tag{33}$$

From (32) and (33) it is obvious that the updates of $q_i^{(m)}(\mathbf{f}_m)$ are performed independently for each of the $M$ models. Point selection, however, should be performed across models, allowing the algorithm to select the most informative point both within and across the different tasks.

$$\Delta H_{in}^{(m)} = -\frac{1}{2}\log\left(1 - \nu_{in}^{(m)} \varsigma_{i-1,n}^{(m)}\right).$$

We also need to maintain an active, $I^{(m)}$, and an inactive, $J^{(m)}$, set for each task. The details of the algorithm are given in Algorithm 3.

The effect of selecting across tasks, rather than selecting independently within tasks is shown by a simple experiment in Figure 7. Here there are three tasks, each contains 30 data points sampled from sine waves with frequency $\frac{\pi}{5}$ and differing offsets. The tasks used different distributions for the input data: in the first it was sampled from a strongly bimodal distribution; in the second it was sampled from a zero mean Gaussian with standard deviation of 2 and in the third task data was sampled uniformly from the range $[-15, 15]$. An MT-IVM with $d = 15$ and an RBF kernel of width 1 was trained on the data. The data points that the MT-IVM used are circled. Note that all but six of the points came from the third task. The first and second task contain less information because the input data is less widely distributed, thus the MT-IVM algorithm relies most heavily on the third task. This toy example illustrates the importance of selecting the data points from across the different tasks.

To determine the kernel parameters, we again maximise the approximation to the likelihood,

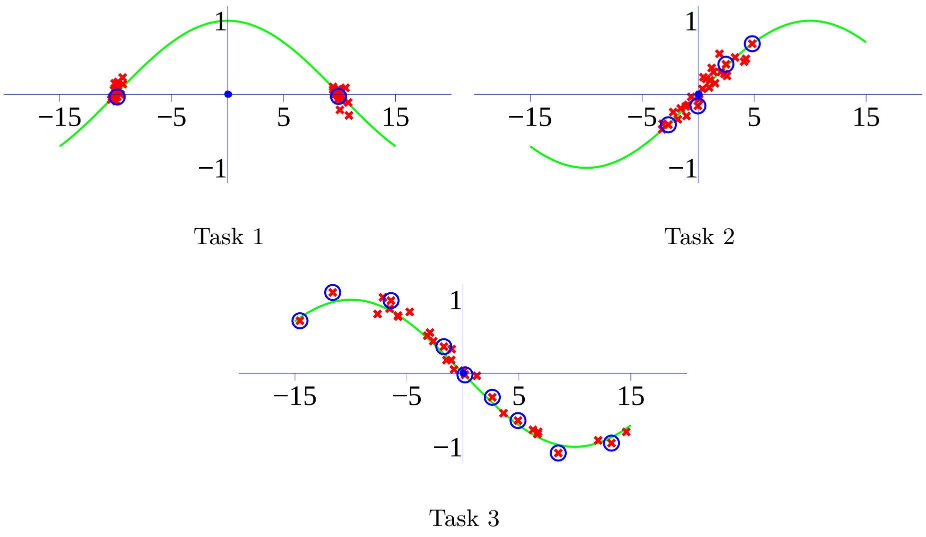
$$p(\mathbf{Y}) \approx \prod_{m=1}^{M} p\left(\mathbf{z}_m | \mathbf{0}, \mathbf{K}_m + \mathbf{B}_m^{-1}\right).$$

### 5.1 Multi-task Learning of Speech

We considered a speech data set from the UCI repository [3]. The data consist of 15 different speakers saying 11 different phonemes 6 times each (giving 66

**Algorithm 3** The multi-task IVM algorithm.

```
Require: $d$ the number of active points.
  for $m = 1 \, \mathrm{to} \, M$ do
    For classification $\mathbf{z}^{(m)} = \mathbf{0}$ and $\boldsymbol{\beta}^{(m)} = \mathbf{0}$. For regression substitute appropriate target values. Take $\boldsymbol{\varsigma}_0^{(m)} = \mathrm{diag}\left(\mathbf{K}_m\right)$, $\boldsymbol{\mu}^{(m)} = \mathbf{0}$, $J^{(m)} = \{1, \dots, N_m\}$, $\mathbf{M}_0^{(m)}$ is an empty matrix.
  end for
  for $k = 1 \, \mathrm{to} \, d$ do
    for $m = 1 \, \mathrm{to} \, M$ do
      for all $n \in J^{(m)}$ do
        compute $g_{kn}^{(m)}$ according to (14) (not required for Gaussian).
        compute $\nu_{kn}^{(m)}$ according to (15) or (10).
        compute $\mathit{\Delta} H_{kn}^{(m)}$ according to (19).
      end for
      {Comment: Select largest reduction in entropy for each task.}
      $\mathit{\Delta} H_k^{(m)} = \max_n \mathit{\Delta} H_{kn}^{(m)}$
      $n_k^{(m)} = \mathrm{argmax}_{n \in J} \mathit{\Delta} H_{kn}$.
    end for
    {Comment: Select the task with the largest entropy reduction.}
    $m_k = \mathrm{argmax}_{m \in J} \mathit{\Delta} H_k^{(m)}$, $n_i = n_k^{(m_k)}$
    Update $m_{n_i}$ and $\beta_{n_i}^{(m_k)}$ using (17) and (18).
    Compute $\boldsymbol{\varsigma}_i^{(m_k)}$ and $\boldsymbol{\mu}_i^{(m_k)}$ using (21), (24) and (25).
    Append $\sqrt{\nu_{in_i}^{(m_k)}} \, \mathbf{s}_{i-1,n_i}^{(m_k)}{}^{\mathrm{T}}$ to $\mathbf{M}_{i-1}^{(m_k)}$ using (21) to form $\mathbf{M}_i$.
    Add $n_i$ to $I^{(m_k)}$ and remove $n_i$ from $J^{(m)}$.
  end for
```



Task 1

Task 2

Task 3

**Figure 7.** Three different learning tasks sampled from sine waves. The input distribution for each task is different. Points used by the MT-IVM are circled. Note that more points are taken from tasks that give more information about the function.

training points for each speaker). By considering the each speaker as a separate task we sought kernel parameters that are appropriate for classifying the different phonemes, *i.e.* we considered that each speaker is independent given the kernel parameters associated with the phoneme. We used 14 of the speakers to learn the kernel parameters for each phoneme giving 14 tasks. Model evaluation was then done by taking one example of each phoneme from the remaining speaker (11 points) and using this data to construct a new Gaussian process model based on those kernel parameters. Then we evaluated this model's performance on the remaining 55 points for that speaker. This mechanism was used for both an MT-IVM model and an ADF trained GP where points were randomly sub-sampled.

To demonstrate the utility of the multi-task framework we also built a IVM based Gaussian process model on the 14 speakers ignoring which speaker was associated with each data point (924 training points). The kernel parameters were optimised for this model and then the model was evaluated as above.

For enough information to be stored by the kernel about the phonemes it needs to be 'parameter rich'. We therefore used used an ARD RBF kernel in combination with an ARD linear kernel, a white noise and constant component. This leads to a total of 15 parameters for the kernel.

The results on the speech data are shown in Figure 8. The convergence of MT-IVM to $\approx 10\%$ error is roughly 10 times faster than the IVM. The MT-IVM takes advantage of the assumed independence to train much faster than the regular IVM. While this data set is relatively small, the structure of this experiment

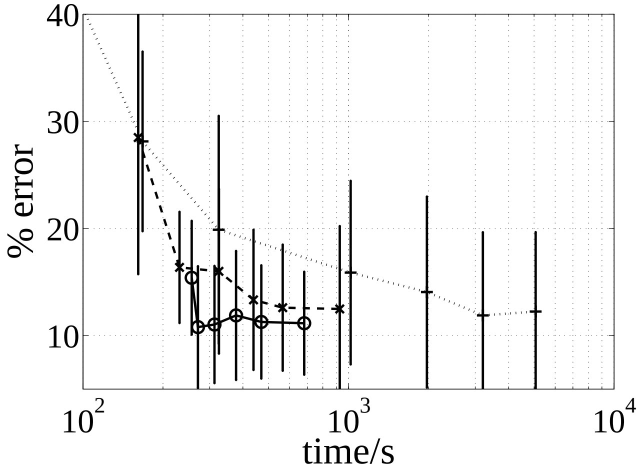


**Figure 8.** Average average time vs error rate for a MT-IVM (solid line with circles) and sub-sampled ADF-GP (dashed line with crosses) whose kernel parameters are optimised by considering each speaker to be an independent task and an IVM optimised by considering all points to belong to the same task (dotted line with pluses).

is important. One reason for the popularity of the HMM/mixture of Gaussians in speech recognition is the ease with which these generative models can be modified to take account of an individual speakers—this is known as speaker-dependent recognition. Up until now it has not been clear how to achieve this with discriminative models. The approach we are suggesting may be applicable to large vocabulary word recognisers and used in speaker-dependent recognition.

## 6 Incorporating Invariances

A learning algorithm can often have its performance improved if invariances present within a data set can be handled. Invariances occur very commonly, for example, in image based data sets. Images can often be rotated or translated without affecting the class of object they contain. The brute force approach to handling such invariances is simply to augment the original data set with data points that have undergone the transformation to which the data is considered invariance. Naturally, applying this technique leads to a rapid expansion in the size of the data set. A solution to this problem that has been suggested in the context of the SVM is to augment only the support vectors (those data points that are included in the expansion of the final solution) [18]. If these augmented points are then included in the final solution they are known as 'virtual support vectors'. The resulting algorithm yields state-of-the-art performance on the USPS data set. Since the IVM also maintains a sparse representation of the data set it seems natural to use the same idea in the IVM context.

### 6.1 USPS with Virtual Informative Vectors

In [18], it was found that the biggest improvement in performance was obtained when using translation invariances. We therefore only considered translation invariances in our experiments. We first applied the standard IVM classification algorithm to the data set using an RBF kernel combined with a linear term. We used 8 iterations of Algorithm 2 for learning the kernel parameters. We then took the resulting active set from these experiments and used it to construct an augmented data set. Each augmented data set had the original active set examples plus four translations of these examples each by one pixel in the up, down, left and right directions as prescribed in [18]. This results in an augmented active set of 2500 points. We then reselected an active set of size $d = 1000$ from this augmented active set using the standard IVM algorithm without further learning of the kernel parameters. The results are shown in Table 3. The resulting performance of the IVM with 'virtual informative vectors' is very similar to that found through the use of 'virtual support vectors'. However with the IVM all the model selection was performed completely automatically.

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **Overall** |
|---|---|---|---|---|---|---|---|---|---|---|
| $0.648 \pm 0.00$ | $0.389 \pm 0.03$ | $0.967 \pm 0.06$ | $0.683 \pm 0.05$ | $1.06 \pm 0.02$ | $0.747 \pm 0.06$ | $0.523 \pm 0.03$ | $0.399 \pm 0.00$ | $0.638 \pm 0.04$ | $0.523 \pm 0.04$ | $3.30 \pm 0.03$ |

**Table 3.** Table of results for when using virtual informative vectors 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. The experiments are summarised by the mean and variance of the % classification error across ten runs with different random seeds.

## 7 Discussion

In this paper, we have reviewed and extended the IVM algorithm from a standard classification and regression technique to an approach that can perform semi-supervised learning; multi-task learning; and incorporate invariances into the model. The IVM algorithm is as computationally efficient as the popular SVM and typically as performant [14]. However the IVM algorithm is an approximation to a Gaussian process and as such it sits within the wider framework of Bayesian models. This firstly allowed us to use the hierarchical Bayesian machinery to extend the IVM to so that it may perform multi-task learning and secondly it allowed us to easily incorporate a simple noise model designed to accommodate the cluster hypothesis for semi-supervised learning.

# Acknowledgements

NL acknowledges the support of the EPSRC grant ‘Learning Classifiers from Sloppily Labelled Data’. MJ acknowledges the support of NSF grant 0412995.

# References

1. J. Baxter. Learning internal representations. In *Proc. COLT*, volume 8, pages 311–320. Morgan Kaufmann Publishers, 1995.
2. S. Becker, S. Thrun, and K. Obermayer, editors. *Advances in Neural Information Processing Systems*, volume 15, Cambridge, MA, 2003. MIT Press.
3. C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.
4. R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.
5. O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In Becker et al. [2].
6. C. Cortes and V. N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
7. L. Csató. *Gaussian Processes — Iterative Sparse Approximations.* PhD thesis, Aston University, 2002.
8. 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.
9. A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis.* Chapman and Hall, 1995.
10. R. E. Kass and D. Steffey. Approximate Bayesian inference in conditionally independent hierarchical models (parametric empirical Bayes models). *Journal of the American Statistical Association*, 84:717–726, 1989.
11. N. D. Lawrence and M. I. Jordan. Semi-supervised learning via Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 17, Cambridge, MA, 2005. MIT Press. To appear.
12. N. D. Lawrence and J. C. Platt. Learning to learn with the informative vector machine. In R. Greiner and D. Schuurmans, editors, *Proceedings of the International Conference in Machine Learning*, volume 21, pages 512–519, San Francisco, CA, 2004. Morgan Kauffman.
13. N. D. Lawrence and B. Schölkopf. Estimating a kernel Fisher discriminant in the presence of label noise. In C. Brodley and A. P. Danyluk, editors, *Proceedings of the International Conference in Machine Learning*, volume 18, San Francisco, CA, 2001. Morgan Kauffman.
14. N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In Becker et al. [2], pages 625–632.
15. D. J. C. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, 1991.
16. T. P. Minka. *A family of algorithms for approximate Bayesian inference.* PhD thesis, Massachusetts Institute of Technology, 2001.
17. 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/`.
18. B. Schölkopf, C. J. C. Burges, and V. N. Vapnik. Incorporating invariances in support vector learning machines. In *Artificial Neural Networks — ICANN'96*, volume 1112, pages 47–52, 1996.

19. B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2001.
20. M. Seeger. Covariance kernels from Bayesian generative models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 905–912, Cambridge, MA, 2002. MIT Press.
21. M. Seeger. *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, The University of Edinburgh, 2004.
22. H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Conference on Computational Learning Theory 10*, pages 287–294. Morgan Kauffman, 1992.
23. P. Sollich. Probabilistic interpretation and Bayesian methods for support vector machines. In *Proceedings 1999 International Conference on Artificial Neural Networks, ICANN'99*, pages 91–96, London, U.K., 1999. The Institution of Electrical Engineers.
24. S. Thrun. Is learning the $n$-th thing any easier than learning the first? In Touretzky et al. [25], pages 640–646.
25. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors. *Advances in Neural Information Processing Systems*, volume 8, Cambridge, MA, 1996. MIT Press.
26. V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.
27. 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.
28. C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In Touretzky et al. [25], pages 514–520.

## A ADF Mean and Covariance Updates

In Section 2.2, we stated that the updates to the mean function and the covariance function for the ADF algorithm could be expressed in terms of the gradients of

$$Z_i = \int t_{n_i}\left(\mathbf{f}\right) q_{i-1}\left(\mathbf{f}\right) d\mathbf{f}.$$

In this appendix, we derive these results, see also [16,7,21]. First we consider that the mean under $q_i\left(\mathbf{f}\right)$ is given by

$$\langle \mathbf{f} \rangle = Z_i^{-1} \int \mathbf{f} t_{n_i}\left(\mathbf{f}\right) q_{i-1}\left(\mathbf{f}\right) d\mathbf{f}.$$

At this point we could substitute in the relevant noise model for $t_{n_i}\left(\mathbf{f}\right)$ and compute the expectation, however it turns out that we can express this expectation, and the second moment, in terms of gradients of the log partition function. This is convenient, as it means we can rapidly compute the update formula for novel noise models. To see how this is done we first note that

$$\nabla_{\boldsymbol{\mu}_{i-1}} q_{i-1}\left(\mathbf{f}\right) = \mathit{\Sigma}_{i-1}^{-1}\left(\mathbf{f} - \boldsymbol{\mu}_{i-1}\right) q_{i-1}\left(\mathbf{f}\right)$$

which can be re-expressed in terms of $\mathbf{f} q_{i-1}\left(\mathbf{f}\right)$,

$$\mathbf{f} q_{i-1}\left(\mathbf{f}\right) = \mathit{\Sigma}_{i-1} \nabla_{\boldsymbol{\mu}_{i-1}} q_{i-1}\left(\mathbf{f}\right) + \boldsymbol{\mu}_{i-1} q_{i-1}\left(\mathbf{f}\right),$$

now multiplying both sides by $Z_i^{-1} t_{n_i}(\mathbf{f})$ and integrating over $\mathbf{f}$ gives

$$\begin{aligned}
\langle \mathbf{f} \rangle_i &= \boldsymbol{\mu}_{i-1} + Z_i^{-1} \Sigma_{i-1} \nabla_{\boldsymbol{\mu}_{i-1}} \int t_{n_i}(\mathbf{f})\, q_{i-1}(\mathbf{f})\, d\mathbf{f} \\
&= \boldsymbol{\mu}_{i-1} + Z_i^{-1} \Sigma_{i-1} \nabla_{\boldsymbol{\mu}_{i-1}} Z_i
\end{aligned}$$

$$\langle \mathbf{f} \rangle_i = \boldsymbol{\mu}_{i-1} + \Sigma_{i-1} \mathbf{g}_i \tag{34}$$

where we have defined $\mathbf{g}_i = \nabla_{\boldsymbol{\mu}_{i-1}} \log Z_i$. The second moment matrix is given by

$$\left\langle \mathbf{f}\mathbf{f}^{\mathrm{T}} \right\rangle_i = Z_i^{-1} \int \mathbf{f}\mathbf{f}^{\mathrm{T}} t_{n_i}(\mathbf{f})\, q_{i-1}(\mathbf{f})\, d\mathbf{f}$$

Once again we take gradients of $q_{i-1}$, but this time with respect to the covariance matrix $\Sigma_{i-1}$.

$$\nabla_{\Sigma_{i-1}} q_{i-1}(\mathbf{f}) = \left( -\frac{1}{2} \Sigma_{i-1}^{-1} + \frac{1}{2} \Sigma_{i-1}^{-1} \left( \mathbf{f} - \boldsymbol{\mu}_{i-1} \right) \left( \mathbf{f} - \boldsymbol{\mu}_{i-1} \right)^{\mathrm{T}} \Sigma_{i-1}^{-1} \right) q_{i-1}(\mathbf{f})$$

which can be re-arranged, as we did for $\langle \mathbf{f} \rangle_i$ to obtain

$$\begin{aligned}
\left\langle \mathbf{f}\mathbf{f}^{\mathrm{T}} \right\rangle_i = \Sigma_{i-1} + 2\Sigma_{i-1} \Gamma^{(i)} \Sigma_{i-1} + \langle \mathbf{f} \rangle_i \boldsymbol{\mu}_{i-1}^{\mathrm{T}} \\
+ \boldsymbol{\mu}_{i-1} \langle \mathbf{f} \rangle_i^{\mathrm{T}} - \boldsymbol{\mu}_{i-1} \boldsymbol{\mu}_{i-1}^{\mathrm{T}}
\end{aligned}$$

where

$$\Gamma^{(i)} = \nabla_{\Sigma_{i-1}} \log Z_i.$$

The update (7) for the covariance requires

$$\left\langle \mathbf{f}\mathbf{f}^{\mathrm{T}} \right\rangle_i - \langle \mathbf{f} \rangle_i \left\langle \mathbf{f}^{\mathrm{T}} \right\rangle_i = \Sigma_{i-1} - \Sigma_{i-1} \left( \mathbf{g}_i \mathbf{g}_i^{\mathrm{T}} - 2\Gamma^{(i)} \right) \Sigma_{i-1}. \tag{35}$$

Substituting (34) and (35) into (6) and (7) we obtain the required updates for the mean and covariance in a form that applies regardless of our noise model.

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + \Sigma_{i-1} \mathbf{g}_i \tag{36}$$

$$\Sigma_i = \Sigma_{i-1} - \Sigma_{i-1} \left( \mathbf{g}_i \mathbf{g}_i^{\mathrm{T}} - 2\Gamma^{(i)} \right) \Sigma_{i-1} \tag{37}$$

## B Kernels Used in Experiments

Throughout the experiments discussed in the main text we construct and learn the parameters of a range of different covariance functions. In this appendix, we give an overview of all the kernels used.

A covariance function can be developed from any positive definite kernel. A new kernel function can also be formed by adding kernels together. In the experiments we present we principally made use of the following three kernel matrices.

The inner product kernel constrains the process prior to be over linear functions,

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

where $\theta$ is the process variance and controls the scale of the output functions. Non linear functions may be obtained through the RBF kernel,

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

where $\gamma$ is the inverse width parameter, which leads to infinitely differentiable functions. Finally we considered the MLP kernel [27] which is derived by considering a multi-layer perceptron in the limit of an infinite number of hidden units,

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

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).

In combination with these kernels we often make use of a white noise process

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

where $\delta_{ij}$ is the Kronecker delta [8]. It is possible to represent uncertainty in the bias by adding a constant term to the kernel matrix,

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

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

All the parameters we have introduced in these kernels need to be constrained to be positive. In our experiments, this constraint was implemented by re-parameterising,

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

Note that as our transformed parameter $\theta' \to -\infty$ the parameter $\theta \to 0$ and as $\theta' \to \infty$ we see that $\theta \to \theta'$.

Finally we can also consider automatic relevance determination (ARD) versions of each of these covariance functions. These process 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 is still valid. The ARD kernels are the specific case where $\mathbf{A}$ is a diagonal matrix, $\mathbf{A} = \text{diag}\left(\boldsymbol{\alpha}\right)$ and the $i$th diagonal element, $\alpha_i$, provides a scaling on the $i$th input variable, these input scales are constrained to be between 0 and 1 by re-parameterising with a sigmoid function,

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

[8] The Kronecker delta $\delta_{ij}$ is zero unless $i = j$ when it takes the value 1.

Unless otherwise stated the kernel parameters were initialised with $\theta_{\mathrm{lin}} = 1$, $\theta_{\mathrm{rbf}} = 1$, $\gamma = 1$, $\theta_{\mathrm{mlp}} = 1$, $w = 10$, $b = 10$ and $\boldsymbol{\alpha} = [0.999, \ldots, 0.999]^{\mathrm{T}}$.