The EM-EP Algorithm for Gaussian Process Classification

Hyun-Chul Kim* and Zoubin Ghahramani

Gatsby Computational Neuroscience Unit, UCL,
London, WC1N 3AR, UK,
{hckim,zoubin}@gatsby.ucl.ac.uk,
http://www.gatsby.ucl.ac.uk

Abstract. Gaussian process classifiers (GPCs) are fully statistical kernel classification models derived from Gaussian processes for regression. In GPCs, the probability of belonging to a certain class at an input location is monotonically related to the value of some latent function at that location. Starting from a prior over this latent function, the data are used to infer both the posterior over the latent function and the values of hyperparameters determining various aspects of the function. GPCs can also be viewed as graphical models with latent variables. Based on the work of [1, 2], we present an approximate EM algorithm, the EM-EP algorithm for learning both the latent function and the hyperparameters of a GPC. The algorithm alternates the following steps until convergence. In the E-step, given the hyperparameters, a density for the latent variables defining the latent function is computed via the Expectation-Propagation (EP) algorithm [1, 2]. In the M-step, given the density for the latent values, the hyperparameters are selected to maximize a variational lower bound on the marginal likelihood (i.e. the model evidence). This algorithm is found to converge in practice and provides an efficient Bayesian framework for learning hyperparameters of the kernel. We examine the role of various different hyperparameters which model labeling errors, the lengthscales (i.e. relevances) of different features, and sharpness of the decision boundary. The added flexibility these provide results in significantly improved performance. Experimental results on synthetic and real data sets show that the EM-EP algorithm works well, with GPCs giving equal or better performance than support vector machines (SVMs) on all data sets tested.

1 Introduction

Kernel classifiers have recently received much attention from the machine learning community. Some popular kernel classifiers are the support vector machine (SVM), Bayes point machine (BPM), and Gaussian process classifier (GPC). The SVM was proposed as a classifier maximizing the margin, which is the smallest distance between data points and the class boundary [3]. SVMs have been a popular tool and have resulted in many successful applications. The BPM is a kernel classifier whose goal is to approximate Bayes-optimal classification by finding the center of the mass of version space, which is the set of hyperplanes in feature space that separate the data [4]. It was

* He was a visting PhD student from POSTECH, South Korea

also shown that SVMs can be viewed as a form of Bayes point machine which tries to find the center of the largest ball to fit in version space. In contrast with the above two classifiers, GPCs are a Bayesian kernel classifier derived from Gaussian process priors over functions which were developed originally for regression [5–7].

Gaussian processes for regression [8, 9, 7] model the density of the target values as a multivariate Gaussian density. Usually, the mean of this Gaussian is assumed to be zero and the covariance between the targets at two different points is a decreasing function of their distance in input space. This decreasing function is controlled by a small set of *hyperparameters* which capture interpretable properties of the function, such as the length scale of autocorrelation, the overall scale of the function, and the amount of noise. The posterior distributions of these hyperparameters given the data can be inferred in a Bayesian way via Markov Chain Monte Carlo (MCMC) methods [8, 7] or they can be selected by maximizing the marginal likelihood (a.k.a. the evidence) [9].

In GPCs, the target values are discrete class labels and so it is not appropriate to model them via a multivariate Gaussian density. However, we can use the Gaussian process as a latent function whose sign determines the class label for binary classification and for multiclass classification one can use multiple GPs or a multivariate GP. GPCs can also be represented as graphical models which have random variables for inputs, latent functions and class labels. Since only the class labels are observed, we need to integrate over both hyperparameters and latent values of these functions at the data points. Williams and Barber (1998) used a Laplace approximation to integrate over the latent values and Hybrid Monte Carlo (HMC) to integrate over the hyperparameters. Neal (1997) used HMC to integrate over both latent values and hyperparameters. Gibbs and Mackay (2000) used a variational approximation to integrate over latent values and determined hyperparameters by maximizing the marginal likelihood. Opper and Winther (2000) used the TAP approach originally proposed in Statistical Physics to integrate over latent values.

It turns out that the TAP approach for GPCs is equivalent to the Expectation Propagation (EP) algorithm for approximate inference in Bayes point kernel machines [1]. In these previous papers, the focus has been on approximate inference rather than determining hyperparameters. Two potential methods for determining the hyperparameters have been proposed in [10]. The first method, which they called mean field method I, is to maximize the variational lower bound of the evidence under the assumption that the densities of latent values are independent and Gaussian. The second method, which they called mean field method II, is to maximize the evidence approximated by Fourier transformation of the likelihood and a saddle point approximation.

In this paper, we propose and investigate a conceptually simple EM-like algorithm to learn the hyperparameters which we call EM-EP. In the E-step, we use EP to estimate the joint density of latent values under the assumption that the joint density is multivariate Gaussian. This multivariate approximation is better than factorized approximations such as the mean field method I. In the M-step, we maximize with respect to the hyperparameters the variational lower bound on the marginal likelihood given by using the density of latent values obtained from the E-step. These two steps are repeated until convergence. The idea of using the variational lower bound for model selection in GPC was suggested in [11]. Here we use a slightly different formulation for GPC and

provide experimental results. Another emphasis of this paper is examining the role of the different hyperparameters and comparing these algorithms with several variants of SVMs. Finally, although improving computational complexity of GPC learning through sparsification methods is an important research problem ([12–14]), we will not address this problem in this paper.

The paper is organized as follows. Section 2 introduces Gaussian process classification. In section 3, we introduce the EP/TAP method for approximate inference. In section 4, we derive the EM-EP algorithm. In section 5, we show experimental results on both synthetic and real data sets, comparing our approach with SVMs. In section 6, we discuss our approach and related work. Software implementing the EM-EP algorithm is available at `http://www.gatsby.ucl.ac.uk/~hckim/software/`.

2 Gaussian Process Classifiers

Let us assume that we have a data set D of data points $\mathbf{x}_i$ with binary class labels $y_i \in \{-1, 1\}$: $D = \{(\mathbf{x}_i, y_i)|i = 1, 2, \ldots, n\}$, $X = \{\mathbf{x}_i|i = 1, 2, \ldots, n\}$, $Y = \{y_i|i = 1, 2, \ldots, n\}$. Given this data set, we wish to find the correct class label for a new data point $\tilde{\mathbf{x}}$. We do this by computing the class probability $p(\tilde{y}|\tilde{\mathbf{x}}, D)$. The graphical model for GPCs is shown in Fig 1.

We assume that the class label is obtained by transforming some real valued latent variable $\tilde{f}$, which is the value of some latent function $f(\cdot)$ evaluated at $\tilde{\mathbf{x}}$. We put a Gaussian process prior on this function, meaning that any number of points evaluated from the function have a multivariate Gaussian density (see Williams and Rasmussen, 1995 for a review of GPs). Assume that this GP prior is parameterized by Θ which we will call the hyperparameters. We can write the probability of interest given Θ as:

$$p(\tilde{y}|\tilde{\mathbf{x}}, D, \Theta) = \int p(\tilde{y}|\tilde{f}, \Theta) p(\tilde{f}|D, \tilde{\mathbf{x}}, \Theta)\, d\tilde{f} \quad (1)$$

The second part of Eq 1 is obtained by further integration over $\mathbf{f} = [f_1 f_2 \cdots f_n]$, the values of the latent function at the data points.

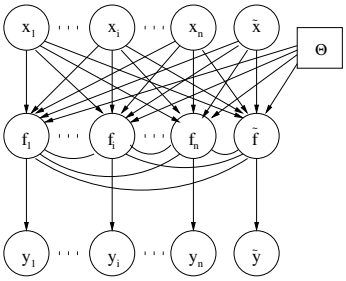

Fig. 1. Graphical model for GPCs with n training data points and one test data point. $\mathbf{x}_i$ and y_i are observed, $\tilde{\mathbf{x}}$ is given, $\tilde{y}$ is what should be predicted, f_i and $\tilde{f}$ are latent and jointly Gaussian, hence have the undirected edges.

$$p(\tilde{f}|D, \tilde{\mathbf{x}}, \Theta) = \int p(\mathbf{f}, \tilde{f}|D, \tilde{\mathbf{x}}, \Theta)\, d\mathbf{f} = \int p(\tilde{f}|\tilde{\mathbf{x}}, \mathbf{f}, \Theta) p(\mathbf{f}|D, \Theta) d\mathbf{f} \quad (2)$$

where $p(\tilde{f}|\tilde{\mathbf{x}}, \mathbf{f}, \Theta) = p(\tilde{f}, \mathbf{f}|\tilde{\mathbf{x}}, \{\mathbf{x}_i\}, \Theta)/p(\mathbf{f}|\{\mathbf{x}_i\}, \Theta)$ and

$$p(\mathbf{f}|D, \Theta) \propto p(Y|\mathbf{f}, X, \Theta) p(\mathbf{f}|X, \Theta) = \{\prod_{i=1}^{n} p(y_i|f_i, \Theta)\} p(\mathbf{f}|X, \Theta). \quad (3)$$

The first term is the likelihood for each observed class given the latent function value, while the second term is the GP prior over functions evaluated at the data and $\tilde{\mathbf{x}}$. One form for the likelihood term $p(y_i|f_i, \Theta)$, which relates $f(x_i)$ monotonically to probability of $y_i = +1$, is

$$p(y_i|f_i, \Theta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y_i f(\mathbf{x}_i)} \exp(-\frac{z^2}{2})\, dz = \mathrm{erf}(y_i f(\mathbf{x}_i)). \tag{4}$$

Writing the dependence of $\mathbf{f}$ on $\mathbf{x}$ implicitly, the GP prior over functions can be written

$$p(\mathbf{f}|\Theta) = \frac{1}{(2\pi)^{N/2}|\mathbf{C}_\Theta|^{1/2}} \exp(-\frac{1}{2}(\mathbf{f} - \boldsymbol{\mu})^\top \mathbf{C}_\Theta^{-1}(\mathbf{f} - \boldsymbol{\mu})), \tag{5}$$

where the mean $\boldsymbol{\mu}$ is usually assumed to be the zero vector $\mathbf{0}$ and each term of a covariance matrix C_{ij} is a function of $\mathbf{x}_i$ and $\mathbf{x}_j$, i.e. $c(\mathbf{x}_i, \mathbf{x}_j)$.

In general, the class probability is obtained by integrating over the hyperparameters $p(\tilde{y}|\tilde{\mathbf{x}}, D) = \int p(\tilde{y}|\tilde{\mathbf{x}}, D, \Theta) p(\Theta|D)\, d\Theta$. This integral can be costly and there are usually many fewer hyperparameters than data points. Therefore, in this paper, rather than integrating over the hyperparameters we fit them by maximizing the marginal likelihood: $\hat{\Theta} = \arg\max_\Theta p(D|\Theta)$. Since $p(\mathbf{f}|D, \Theta)$ in Eq 3 is intractable due to the nonlinearity in Eq 4, we use EP to approximate it.

3 EP for Gaussian Process Classifiers

The Expectation-Propagation (EP) algorithm is an approximate Bayesian inference method [1]. We review EP in its general form before describing its application to GPCs.

Consider a Bayesian inference problem where the posterior over some parameter ϕ is proportional to the prior times likelihood terms for an i.i.d. data set

$$p(\phi|y_1, \ldots, y_n) \propto p(\phi) \prod_{i=1}^{n} p(y_i|\phi) \tag{6}$$

We approximate this by

$$q(\phi) \propto \tilde{t}_0(\phi) \prod_{i=1}^{n} \tilde{t}_i(\phi) \tag{7}$$

where each term (and therefore q) is assumed to be in the exponential family. EP successively solves the following optimization problem

$$\tilde{t}_i^{\mathrm{new}}(\phi) = \arg\min_{\tilde{t}_i(\phi)} \mathrm{KL}\left(\frac{q(\phi)}{\tilde{t}_i^{\mathrm{old}}(\phi)} p(y_i|\phi) \,\middle\|\, \frac{q(\phi)}{\tilde{t}_i^{\mathrm{old}}(\phi)} \tilde{t}_i(\phi) \right) \tag{8}$$

Where KL is the Kullback-Leibler divergence. Since q is in the exponential family, this minimization is solved by matching moments of the approximated distribution. EP iterates over i until convergence. The algorithm is not guaranteed to converge although it did in practice in all our examples. Assumed Density Filtering (ADF) is a special online form of EP where only one pass through the data is performed ($i = 1, \ldots n$).

We describe EP for GPC referring to [1, 2, 14]. The latent function $\mathbf{f}$ plays the role of the parameter ϕ above. The form of the likelihood we use in the GPC is

$$p(y_i|f_i) = \epsilon + (1-2\epsilon)H(y_i f_i), \tag{9}$$

where $H(x) = 1$ if $x > 0$, and otherwise 0. The hyperparameter, ϵ in Eq 9 models labeling error outliers. The EP algorithm approximates the posterior $p(\mathbf{f}|D) = p(\mathbf{f})p(D|\mathbf{f})/p(D)$ as a Gaussian having the form $q(\mathbf{f}) \sim \mathcal{N}(\mathbf{m_f}, \mathbf{V_f})$, where the GP prior $p(\mathbf{f}) \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$ has covariance matrix $\mathbf{C}$ with elements C_{ij} defined by the covariance function

$$C_{ij} = c(\mathbf{x}_i, \mathbf{x}_j) = v_0 \exp\{-\frac{1}{2}\sum_{m=1}^{d} l_m d_m(x_i^m, x_j^m)\} + v_1 + v_2\delta(i,j), \tag{10}$$

where x_i^m is the mth element of $\mathbf{x}_i$, and $d_m(x_i^m, x_j^m) = (x_i^m - x_j^m)^2$ if x^m is continous; otherwise, $1 - \delta(x_i^m, x_j^m)$. The hyperparameter v_0 specifies the overall vertical scale of variation of the latent values, v_1 the overall bias of the latent values from zero mean, v_2 the latent noise variance, and l_m the (inverse) lengthscale for feature dimension m. The erf likelihood term 4 is equivalent to using the threshold function in Eq 9 with $\epsilon = 0$ and non-zero latent noise v_2.

EP tries to approximate $p(\mathbf{f}|D) = p(\mathbf{f})/p(D)\prod_{i=1}^{n} p(y_i|\mathbf{f})$, where $p(\mathbf{f}) \sim \mathcal{N}(0, \mathbf{C})$. $p(y_i|\mathbf{f}) = t_i(\mathbf{f})$ is approximated by $\tilde{t}_i(\mathbf{f}) = s_i \exp(-\frac{1}{2v_i}(f_i - m_i)^2)$. From this initial setting, we can derive EP for GPC by applying the general idea described above. The resulting EP procedure is virtually identical to the one derived for BPMs in [1]. We define the following notation[1]: $\Lambda = \mathrm{diag}(v_1, \ldots, v_n)$; $h_i = E[f_i]$; $h_i^{\backslash i} = E[f_i^{\backslash i}]$, where $h_i^{\backslash i}$ and $f_i^{\backslash i}$ are ones obtained from a whole set except for $\mathbf{x}_i$. The EP algorithm is as follows which we repeat for completeness—please refer to [1] for the details of the derivation. After the initialization $v_i = \infty, m_i = 0, s_i = 1, h_i = 0, \lambda_i = C_{ii}$, the following process is performed until all (m_i, v_i, s_i) converge:

Loop $i = 1, 2, \ldots, n$:

1. Remove the approximate density $\tilde{t}_i$ (for ith data point) from the posterior to get an 'old' posterior: $h_i^{\backslash i} = h_i + \lambda_i v_i^{-1}(h_i - m_i)$
2. Recompute part of the new posterior: $z = \frac{y_i h_i^{\backslash i}}{\sqrt{\lambda_i}}$; $Z_i = \epsilon + (1-2\epsilon)\mathrm{erf}(z)$ $\alpha_i = \frac{1}{\sqrt{\lambda_i}}\frac{(1-2\epsilon)\mathcal{N}(z;0,1)}{\epsilon+(1-2\epsilon)\mathrm{erf}(z)}$; $h_i = h_i^{\backslash i} + \lambda_i\alpha_i$, where $\mathrm{erf}(z)$ is a cumulative normal density function.
3. Get a new $\tilde{t}_i$: $v_i = \lambda_i(\frac{1}{\alpha_i h_i} - 1)$; $m_i = h_i + v_i\alpha_i$; $s_i = Z_i\sqrt{1 + v_i^{-1}\lambda_i}\exp(\frac{\lambda_i\alpha_i}{2h_i})$
4. Now that v_i is updated, finish recomputing the new posterior: $\mathbf{A} = (\mathbf{C}^{-1} + \boldsymbol{\Lambda}^{-1})^{-1}$; For all i, $h_i = \sum_j A_{ij}\frac{m_j}{v_j}$; $\lambda_i = (\frac{1}{A_{ii}} - \frac{1}{v_i})^{-1}$

Our approximated posterior over the latent values is:

$$q(\mathbf{f}) \sim \mathcal{N}(\tilde{\mathbf{C}}\boldsymbol{\alpha}, \mathbf{A}), \tag{11}$$

[1] $\mathrm{diag}(v_1, \ldots, v_n)$ means a diagonal matrix whose diagonal elements are $v_1, \ldots, v_n$. Similarly for $\mathrm{diag}(\mathbf{v})$.

where $\tilde{C}_{ij} = y_j c(\mathbf{x}_i, \mathbf{x}_j)$ (or $\tilde{\mathbf{C}} = \mathbf{C}\mathrm{diag}(\mathbf{y})$) The approximate evidence can obtained in the same way as for BPMs:

$$p(D|\Theta) \approx \frac{|\Lambda|^{1/2}}{|\mathbf{C}+\Lambda|^{1/2}} \exp(B/2) \prod_{i=1}^{n} s_i \tag{12}$$

where $B = \sum_{ij} A_{ij} \frac{m_i m_j}{v_i v_j} - \sum_i \frac{m_i^2}{v_i}$. The approximate evidence in 12 can be used to evaluate the feasibility of kernels or their hyperparameters to the data. But, it is tricky to get a updating rule from Eq 12. In the following section, we derive the algorithm to find the hyperparameters automatically based not on Eq 12 but a variational lower bound of the evidence. Classification of a new data point $\tilde{\mathbf{x}}$ can be done according to $\mathrm{argmax}_{\tilde{y}} p(\tilde{y}|\tilde{\mathbf{x}}) = \mathrm{sgn}(E[\tilde{f}]) = \mathrm{sgn}(\sum_{i=1}^{n} \alpha_i y_i c(\mathbf{x}_i, \tilde{\mathbf{x}}))$.

4 The EM-EP Algorithm

We tackle the problem of learning the classifier hyperparameters as one of optimzing hyperparameters for Gaussian process regression with hidden target values. This idea makes it possible to apply an EM-like algorithm. In the E-step, we infer the approximate (Gaussian) density for latent function values $q(\mathbf{f})$ using EP. In the M-step, using $q(\mathbf{f})$ obtained in the E-step, we maximize the variational lower bound of $p(D|\Theta)$. The E-step and M-step are alternated until convergence.

E-step EP iterations are performed given the hyperparameters. $p(\mathbf{f}|\mathbf{y})$ is approximated as a Gaussian density $q(\mathbf{f})$ given by Eq 11.

M-step Given $q(\mathbf{f})$ obtained from the E-step, find the hyperparameters which maximize the variational lower bound of $p(\mathbf{y}|\Theta)$. We define $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$, $\Theta = \Theta_{\mathrm{cov}} \cup \{\epsilon\}, \Theta_{\mathrm{cov}} = \{v_0, v_1, v_2\} \cup \{l_p | p = 1, 2, \ldots, d\}$. Then we get $p(\mathbf{y}|\Theta) = \int p(\mathbf{y}|\mathbf{f}, \epsilon) p(\mathbf{f}|\Theta_{\mathrm{cov}})\, d\mathbf{f}$. Since the above integral is intractable, we use a variational approximation technique. We take a variational lower bound F as follows.

$$\begin{aligned} \log p(\mathbf{y}|\Theta) &= \log \int p(\mathbf{y}|\mathbf{f}, \epsilon) p(\mathbf{f}|\Theta_{\mathrm{cov}})\, d\mathbf{f} \\ &\geq \int q(\mathbf{f}) \log \frac{p(\mathbf{y}|\mathbf{f}, \epsilon) p(\mathbf{f}|\Theta_{\mathrm{cov}})}{q(\mathbf{f})}\, d\mathbf{f} = F \end{aligned} \tag{13}$$

Using the E-step result Eq 11 and the definition of $\tilde{\mathbf{C}}$, we obtain the following gradient update rule with respect to the covariance hyperparameters

$$\begin{aligned} \frac{\partial F}{\partial \Theta_{\mathrm{cov}}} = \frac{1}{2} \boldsymbol{\alpha}^\top \mathrm{diag}(\mathbf{y}) \frac{\partial \mathbf{C}}{\partial \Theta_{\mathrm{cov}}} \mathrm{diag}(\mathbf{y}) \boldsymbol{\alpha} \\ - \frac{1}{2} tr(\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \Theta_{\mathrm{cov}}}) + \frac{1}{2} tr(\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \Theta_{\mathrm{cov}}} \mathbf{C}^{-1} \mathbf{A}). \end{aligned} \tag{14}$$

We update ϵ^* to maximize F by $\epsilon^* = r/n$, where r is the number of errors on the training data. (See The Appendix for the derivation of the M-step.)

We found that in practice EM-EP always converged and the local maxima were good solutions. EM-EP has a complexity of $O(n^3)$ due to the matrix inversion in EP.

5 Experimental Results

One of the goals of this paper is to demonstrate the usefulness of learning the hyperparameters used to control the kernel and label noise for classification. The EM-EP procedure is a relatively straightforward procedure for doing this in GPCs. Two synthetic data sets are used to show the usefulness of a labeling noise (ϵ) and lengthscale hyperparameters (l_m), respectively. We used real world data sets to compare the proposed algorithm with SVMs and other classification methods. In the M-step, we used the conjugate gradient method with line searches[2]. All covariance function hyperparameters are optimized in log transformed spaces so as to avoid constrained optimization.

5.1 Synthetic Data Sets

The labeling error hyperparameter ϵ helps deal with label outliers in a training set. A labeling error cannot be guaranteed to be located near a decision boundary. So even though a classifier might provide a soft decision boundary, it might not deal with a mislabeled data point properly.

We generated a simple 2-class 2-dimensional data set whose input features x_1 and x_2 are random numbers between -1 and 1. The class $+1$ or -1 is determined by the $x_1^2 + x_2^2 \geq 0.5$ decision rule. We generated 1000 data points and selected 40 data points as the training set and added labeling errors to two randomly selected data points in the training set. Fig 2 (a) and Fig 2 (b) shows the training set and the test set. We compared a version of our algorithm with fixed ($\epsilon = 0$) labeling error hyperparamter to one where ϵ was adpated[3]. All other covariance function hyperparameters were updated.

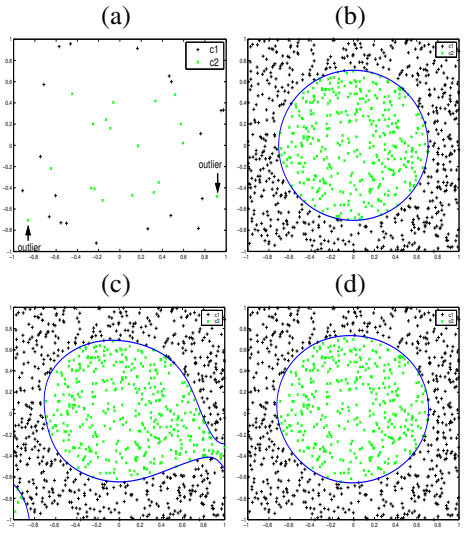

Fig 2: (a) Training set, (b) Test set, (c) Classification result with fixed labeling error hyperparamter ($\epsilon = 0$), (d) Classification result with adapted labeling error hyperparameter ϵ

The classification results for the test set with adapted and fixed labeling error hyperpramter are shown in Fig 2 (c) and Fig 2 (d), respectively. The figures show that

[2] The code is available from `www.gatsby.ucl.ac.uk/~edward/code/minimize/`.

[3] Initial values: $\epsilon^0 = 0.01$ (adpated), $v_0^0 = 1$, $v_1^0 = 10^{-8}$, $v_2^0 = 10^{-6}$, $l_1^0 = l_2^0 = 0.1$.

with fixed labeling error hyperparameter GPC cannot deal with outliers properly even though it has the latent function noise hyperparameter, v_2 which controls the softness of the decision boundary. The outliers affected the hyperparameter learning and made a more complex model. The lengthscale-related parameters are larger in the GPC with fixed labeling error hyperparameter than the one with it. The log approximate evidences $\log p(D|\Theta)$ [4] for the case $\epsilon = 0$ and adapted ϵ are -23.2358 and -22.9487, repectively. The classification error rates for the case $\epsilon = 0$ and adapted ϵ are 0.0740 and 0.0281, respectively. GPC with adapted labeling error hyperparameter clearly outperforms the one with it fixed by both approximate evidence and classification test error rate. The ϵ obtained from EM-EP was the real labeling error 0.05.

Lengthscale hyperparameters help to select relevant features.[5] Irrelevant features can have a devastating effect on classifier performance. We can select irrelevant features before learning a classifier, or try to learn classifiers with many combinations of features. The first approach is not easy because it is hard to find a good measure to evaluate the relevance for general classification problems. The second approach is computationally intractable if the data set has many features. Moreover, it can lead to overfitting in the choice of feature subsets. If we learn lengthscale hyperparameters of the kernel, we can learn the classifier and relevance of each feature simultaneously.

We generated a simple data set with 6 features distributed as follows: $x_1, x_2, x_3 \sim \mathcal{N}(y, 1)$ and $x_4, x_5, x_6 \sim \mathcal{N}(0, 1)$. That is, x_1, x_2, x_3 are relevant features while x_4, x_5, x_6 are irrelevant to the classification problem. We generated 300 data samples for a training set and 100 data samples for a test set. We tried the EM-EP algorithm with a single lengthscale hyperparameter for all dimensions or with multiple lengthscale hyperparameters. In the case of mutiple lengthscale hyperparameters, as would be hoped, we see that the lengthscale parameters for the irrelevant features (x_4,x_5,x_6) went down to near zero. The approximate log evidences for the single and multiple cases are -55.45 and -35.41, respectivley. The classification error rates for the single and mutliple cases are 0.06 and 0.04, respectively. The result show that GPC with multiple lengthscale hyperparameters was significantly better than one with a single lengthscale as measured both by classification error rates as well as log evidence.

5.2 Real World Data Sets

We applied the proposed algorithm to several real world data sets. The real world data sets we used are from the UCI Machine Learning Repository and PRNN sites[6]. The detailed information for the data sets is in Table 1. Thyroid, Heart disease, Ionosphere data sets were obtained from UCI MLR and Crabs and Pima data sets were obtained from the PRNN site.

Table 2 shows classification error rates of various methods. Each data set was divided into 10 folds. Each fold were subsequently used as a test set, while the other 9

[4] The approximate evidence was obtained from Eq 12

[5] In our parameterization, relevant features should have larger values of the hyperparameter, which actually means the underlying function has *shorter* lengthscales.

[6] Available from `http://www.ics.uci.edu/~mlearn/MPRepository.html` and `http://www.stats.ox.ac.uk/pub/PRNN`, respectively

Table 1. Detailed information on the real world data sets

Data set	# of discrete variables	# of continous variables	# of classes	# of data points
Thyroid	0	5	3 (to 2)	215
Heart disease	7	6	2	294
Ionosphere	0	34	2	351
Crabs	2	5	2	200
Pima (Tr)	0	7	2	200

folds were used as a training set. The numbers in Table 2 are the means of those 10 error rates and standard deviations of the mean estimators. We tried three versions of the EM-EP Gaussian Process Classifier: **GPC-EP(s,soft)** used a *single* lengthscale hyperparameter for all feature dimensions, while **GPC-EP(m,soft)** used a different lengthscale hyperparameter for each feature dimension[7]. Finally, **GPC-EP(s,hard)** was a GPC with a single lengthscale hyperparameter where the decision boundary was "hard" in the sense that the latent function noise parameter v_2 was fixed to zero or a very small number. All GPC models adpated ϵ.

We compared our results to several variants of SVMs[8]. In **SVM-EP(s,soft)** the kernel, (i.e. covariance function) had the *same* hyperparameters as the corresponding GPC-EP(s,soft) trained using EM-EP except for the latent noise variance v_2 which was omitted because it caused degradation in the SVM performance. Instead, the penalty parameter C allowing training errors (i.e. penalizing the SVM slack variables) was selected by 5-fold cross-validation.[9]

We also applied both hard and soft-margin SVMs with a Gaussian kernel with a single lengthscale hyperparameter (without v_0, v_1 and v_2) selected by 5-fold cross-validation.[10] For hard-margin SVM, **SVM-CV(hard)**, we only needed to do a 2-level grid search for l. For soft-margin SVMs, **SVM-CV(soft)**, we also had to determine the penalty parameter C, so we did a 2-level grid search over a 2-dimensional parameter space (C, l)[11]. Finally, for comparison to baseline methods we also examined the performance of One Nearest Neighbour (**1-NN**) and linear discriminant analysis(**LDA**).

The experimental results for the three versions of EM-EP applied to GPC models provide interesting insights. GPC with latent function noise (GPC-EP(s,soft)), i.e. which explicitly allows soft boundaries, is better than or as good as the harder version

[7] The initial values of hyperparameters for GPC-EP(s,soft) for the first fold were as follows: $\epsilon^0 = 0, v_0^0 = 1, v_1^0 = 0.0001, v_2^0 = 0.001, l_m^0 = 0.05, \forall m$, and those for subsequent folds are the results for the former fold. For GPC-EP(m,soft), the initial values of the hyperparameters for every fold were the results learned for the same fold in GPC-EP(s,soft).

[8] Using the MATLAB Support Vector Machine Toolbox available from `http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox` with modified kernels.

[9] Firstly, we did a coarse grid search over $\{C|\log_{10} C = 0, 0.5, 1, 1.5, 2, 2.5, 3\}$ to obtain C_1. Then did a finer search over $\{C|\log_{10} C = \log C_1 - 0.4, \log C_1 - 0.3, \ldots, \log C_1 + 0.4\}$.

[10] Similarly to the selection of C, we did a 2-level grid search over $\{l|\log_{10} l = -3, -2.5, -2, \ldots, 0\}$ and $\{l|\log_{10} l = \log_{10} l_1 - 0.4, \log l_1 - 0.3, \ldots, \log l_1 + 0.4\}$.

[11] The same grids as above for parameters C, l were used.

Table 2. Classification error rates of various methods for real world data sets

	Heart disease	Thyroid	Ionosphere	Crabs	Pima
GPC-EP(m,soft)	**0.1559±0.0329**	0.0277±0.0147	0.0514±0.0173	**0.025±0.0118**	0.280±0.0510
GPC-EP(s,hard)	0.2043±0.0193	0.0277±0.0147	0.0513±0.0141	0.065±0.0158	0.325±0.0473
GPC-EP(s,soft)	0.1568±0.0212	0.0277±0.0147	0.0485±0.0156	0.065±0.0158	**0.245±0.0448**
SVM-EP(s,soft)	0.2011±0.0289	**0.0229±0.0148**	**0.0429±0.0163**	0.050±0.0176	0.255±0.0448
SVM-CV(hard)	0.2349±0.0241	0.0420±0.0153	0.0568±0.0167	0.060±0.0205	0.340±0.0483
SVM-CV(soft)	0.1943±0.0212	0.0366±0.0224	0.0598±0.0158	0.055±0.0146	0.275±0.0403
1-NN	0.3724±0.0421	0.0561±0.0177	0.1398±0.0159	0.075±0.0196	0.295±0.0254
LDA	0.1664±0.0186	0.1251±0.0204	0.1256±0.0226	0.055±0.0166	0.255±0.0288

(GPC-EP(s,hard)). This shows that allowing ambiguity at the boundary is important. For these size data sets, the model with multiple lengthscale hyperparameters (GPC-EP(m,soft)) did not always outperform the single lenghscale model (GPC-EP(s,soft)). However, multiple lengthscales did seem to be essential in learning the Crabs data set, where its error rate was less than half the nearest competitor, and the multiple lengthscale model usually performed among the top methods. For higher dimensional data sets, fitting too many lengthscale hyperparameters can clearly lead to overfitting.

The experimental results for the three variants of SVMs are also enlightening. The SVM with the same hyperparameters as GPC trained by EM-EP (SVM-EP(s,soft)) is worse than (Heart disease, Crabs, Pima) or comparable to or slightly better than (Thyroid, Ionosphere) the corresponding GPC (GPC-EP(s,soft)). Hard-margin SVM with cross-validation is worse than GPC with a hard decision boundary on 4 out of 5 data sets. In all data sets, GPC with soft decision boundary (GPC-EP(m,soft) or GPC-EP(s,soft)) is better than Soft-margin SVM (SVM-CV(soft)) with cross-validation. In all cases, the EM-EP procedure seems to perform better than cross-validation, even when it comes to fitting the SVM kernel hyperparameters. Moreover, cross-validation would be computationally prohibitive for models with many hyperparameters, such as the multiple lengthscale models.

GPC and SVM both have complexity of $O(n^3)$, but SVM is usually faster since it uses a sparse scheme. SVM-CV is slow because of cross validation.

6 Conclusion

Based on the work of [1, 2], we presented an approximate EM algorithm, the EM-EP algorithm, for hyperparameter learning in Gaussian process classifiers. Experiments on synthetic real world data sets showed the usefulness of hyperparameters related to a labeling error, lengthscales, and latent noise. GPC with EM-EP showed better performances than SVM with cross-validation in all the data sets used in the experiment.

Viewing GPCs as probablistic graphical models, they can be incorporated into larger graphs that also model the input variables, or a models of the conditional distribution of discrete children given parents [15]. Gaussian Process Classifiers also have some advantages over other kernel methods because they are fully statistical models. We can

use the evidence for model selection and kernel hyperparameter optimization. Also, given new data, we can get a class probability rather than a hard decision. Even though we did not use it in this paper, prior information can be used to inform learning of the hyperparameters (for example, if some input features are thought to be more relevant or the noise is thought to be high). Sparse approaches for GPC to reduce the computation time have recently been developed [12, 14] that could be applied here. The relation between GPCs and SVMs is elaborated in [16] where other sparse versions of GPCs are also derived.

An important area of future work is a muti-class extension of GPC with the EM-EP algorithm. GPC can provide a more fully statistical model for multi-class classification than SVMs by making use of softmax parameterization of the multinomial distribution [5]. Combined with sparse versions of EM-EP and parameterized kernels, this could provide a very powerful general classification system.

Acknowledgement

This work was done while Hyun-Chul was a visiting PhD student at the Gatsby Unit, through BK21 program at POSTECH, funded by the Education Ministry of Korea.

Appendix A

We take a variational lower bound F as follows. Starting from Eq 13,

$$F = \int q(\mathbf{f}) \log p(\mathbf{y}|\mathbf{f}, \epsilon)\, d\mathbf{f} + \int q(\mathbf{f}) \log p(\mathbf{f}|\Theta_{\text{cov}})\, d\mathbf{f} - \int q(\mathbf{f}) \log q(\mathbf{f})\, d\mathbf{f} \quad (15)$$

We use F_ϵ, $F_{\Theta_{\text{cov}}}$, and $H(Q)$, respectively, to denote the three integrals that make up F in Eq 15. Since $H(Q)$ is independent of the hyperparamenter set Θ, and ϵ is independent of Θ_{cov}, we optimize F for Θ, by optimizing $F_{\Theta_{\text{cov}}}$ and F_ϵ for Θ_{cov} and ϵ, respectively.

By expanding $F_{\Theta_{\text{cov}}}$, we get

$$\begin{aligned} F_{\Theta_{\text{cov}}} &= E_Q[\log p(\mathbf{y}|\mathbf{f}, \epsilon)] \\ &= E_Q[-\frac{1}{2}\log|2\pi\mathbf{C}| - \frac{1}{2}\mathbf{f}^\top\mathbf{C}^{-1}\mathbf{f}] \\ &= -\frac{1}{2}\log|2\pi\mathbf{C}| - \frac{1}{2}E_Q[\mathbf{f}^\top\mathbf{C}^{-1}\mathbf{f}] \\ &= -\frac{1}{2}\log|2\pi\mathbf{C}| - \frac{1}{2}E[\mathbf{f}]^\top\mathbf{C}^{-1}E[\mathbf{f}] - \frac{1}{2}tr(\mathbf{C}^{-1}Cov(\mathbf{f})) \end{aligned} \quad (16)$$

Differentiating $F_{\Theta_{\text{cov}}}$ for θ, using the E-step result Eq 11 and the definition of $\tilde{\mathbf{C}}$, we obtain

$$\begin{aligned} \frac{\partial F_{\Theta_{\text{cov}}}}{\partial\theta} &= -\frac{1}{2}tr(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta}) + \frac{1}{2}E[\mathbf{f}]^\top\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta}\mathbf{C}^{-1}E[\mathbf{f}] + \frac{1}{2}tr(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta}\mathbf{C}^{-1}Cov(\mathbf{f})) \\ &= -\frac{1}{2}tr(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta}) + \frac{1}{2}\boldsymbol{\alpha}^\top\text{diag}(\mathbf{y})\frac{\partial\mathbf{C}}{\partial\theta}\text{diag}(\mathbf{y})\boldsymbol{\alpha} + \frac{1}{2}tr(\mathbf{C}^{-1}\frac{\partial\mathbf{C}}{\partial\theta}\mathbf{C}^{-1}\mathbf{A}). \end{aligned} \quad (17)$$

Using Eq 17, we find Θ^* which maximizes $F_{\Theta_{\mathrm{cov}}}$.

For the labeling error hyperparameter ϵ, since the integral in F_ϵ is also intractable, it is simply approximated as

$$F_\epsilon = \int q(\mathbf{f}) \log p(\mathbf{y}|\mathbf{f}, \epsilon)\, df \simeq q(E(\mathbf{f})) \log p(\mathbf{y}|E(\mathbf{f}), \epsilon) \Delta\mathbf{f}, \tag{18}$$

where $\Delta\mathbf{f}$ is the width of the distribution $q(\mathbf{f})$ at $E(\mathbf{f})$ Then, only $\log p(\mathbf{y}|E(\mathbf{f}), \epsilon)$ depends on ϵ, and $p(\mathbf{y}|E(\mathbf{f}), \epsilon)$ is expressed as $\epsilon^r(1-\epsilon)^{(n-r)}$, where r is the number of errors when the sign of $E(\mathbf{f})$ is treated as the predicted class labels. By differentiating $p(\mathbf{y}|\epsilon, E(\mathbf{f}))$ for ϵ and equalizing the derivative to zero, we get $\epsilon^* = r/n$.

References

1. Minka, T.: A family of algorithms for approximate Bayesian inference. PhD thesis, MIT (2001)
2. Opper, M., Winther, O.: Gaussian processes for classification: Mean field algorithms. Neural Computation **12** (2000) 2655–2684
3. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
4. Herbrich, R., Graepel, T., Campbell, C.: Bayes point machines. Journal of Machine Learning Research **1** (2001) 245–279
5. Williams, C.K.I., Barber, D.: Bayesian classification with Gaussian processes. IEEE Transactions on PAMI **20** (1998) 1342–1351
6. Gibbs, M., MacKay, D.J.C.: Variational Gaussian process classifiers. IEEE Transactions on NN **11** (2000) 1458
7. Neal, R.: Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report CRG–TR–97–2, Dept. of Computer Science, University of Toronto (1997)
8. Williams, C.K.I., Rasmussen, C.E.: Gaussian processes for regression. In Touretzky, D.S., Mozer, M.C., Hasselmo, M.E., eds.: NIPS. Volume 8., MIT Press (1995)
9. Gibbs, M., MacKay, D.J.C.: Efficient implementation of Gaussian processes. draft manuscript (http://citeseer.nj.nec.com/6489.html) (1997)
10. Csato, L., Fokoue, E., Opper, M., Schottky, B., Winther, O.: Efficient approaches to Gaussian process classification. In: NIPS. Volume 13. (2000)
11. Seeger, M.: Notes on minka's expectation propagation for Gaussian process classification. Technical Report (2002)
12. Csato, L., Opper, M.: Sparse representation for Gaussian process models. In: NIPS. Volume 13. (2000)
13. Csato, L., Opper, M., Winther, O.: TAP Gibbs free energy, belief propagation and sparsity. In: NIPS. Volume 14. (2001)
14. Seeger, M., Lawrence, N., Herbrich, R.: Sparse representation for Gaussian process models. In: NIPS. Volume 15. (2002)
15. Tresp, V.: Mixtures of Gaussian processes. In: NIPS. Volume 13. (2000)
16. Chu, W.: Bayesian approach to support vector machines. PhD thesis, National University of Singapore (2003)