Bayesian Learning with Gaussian Processes for Supervised Classification of Hyperspectral Data

Kaiguang Zhao, Sorin Popescu, and Xuesong Zhang

Abstract
Recent advances in kernel machines promote the novel use of Gaussian processes (GPs) for Bayesian learning. Our purpose is to introduce GPs into the remote sensing community for supervised learning as exemplified in this study for classifying hyperspectral images. We first provided the mathematical formulation of GPs concerning both regression and classification; described several GP classifiers (GCPs) and the automatic learning of kernel parameters; and then, examined the effectiveness of GCPs compared with K-nearest neighbor (KNN) and Support Vector Machines (SVM). Experiment results on an Airborne Visible/Infrared Imaging Spectroradiometer image indicate that the GCPs outperform KNN and yield classification accuracies comparable to or even better than SVMs. This study shows that GPs models, though with a larger computation scaling than SVM, bring a competitive tool for remote sensing applications related to classification or possibly regression, particularly with small or moderate sizes of training datasets.

Introduction
Supervised classification remains a crucial task in extracting thematic information from remotely-sensed imagery. Along with the increasing availability of hyperspectral images that have greatly enhanced the discrimination capability of remote sensing for more detailed mapping of land covers (Cochrane, 2000; Chang, 2002; Stippert, 2004), a concomitant need is arising for novel and effective techniques to analyze and classify these high-dimensional data, since most traditional classifiers, though effective for multispectral images, often show unsatisfactory performances for hyperspectral images due to the Hughes phenomenon, i.e., the curse of dimensionality (Landgrebe, 2002; Zhong et al., 2007).

Among traditional image classification algorithms, the maximum likelihood classifier (MLC) is possibly the most notable method, and it assumes the Gaussian distributions for each spectral class (Richards and Jia, 1999); therefore, non-Gaussianity of data could severely compromise the performance of the MLC, and the distribution parameters cannot be reliably estimated without enough training data (Hoffbeck and Landgrebe, 1996). Given adequate training samples, a remedy for non-Gaussianity is to use a mixture of Gaussians or non-parametric estimators (e.g., Parzen window estimator), for inferring class conditional distributions.
in light of training data, as opposed to the resistance of classical SVMs to an appropriate Bayesian framework (Chu et al., 2003).

Successful applications of GPs have recently been reported in various fields both for regression and classification tasks (Bermak and Belhouari, 2006; Ferris et al., 2007). Although the attraction of GPs to the machine learning community is of only recent origin, as initiated by the observation that Bayesian regression using a NN with infinite hidden units may boil down to a GP model (MacKay, 1997), GPs by themselves are far from new as witnessed by the well-known “Kriging” method in geostatistics (Cressie, 1993). GPs are also called the Bayesian version of SVMs because the two approaches bear great resemblance with each other (Chu et al., 2003). Despite all the advantages, GPs do suffer from a major drawback that the training cost is of scale $O(n^2)$ for a training set of $n$ samples, which somewhat precludes its practical use for applications with large-scale training data. In fact, the same computational issue originally existed for training a SVM, but was later alleviated by the proposal of fast training algorithms (Csató and Opper, 2002). To speed-up training GPs, a variety of efficient schemes have been proposed on the basis of various considerations (Rasmussen and Williams, 2006).

The objective of this study is to introduce the Bayesian learning of GP models for remote sensing applications, specifically for classifying hyperspectral data. For this purpose, the rest of the paper is organized broadly into two parts: the first part presents an overview of the general Bayesian GP modeling for supervised learning including both regression and classification, and also deals with several practical aspects on the use of GP models such as the learning of optimal kernel parameters, sparse GP models and the implementation issues; the second part conducts three experiments to evaluate the performances of GP classifiers (GPCLs) as applied on an Airborne Visible Infrared Imaging Spectrometer (AVIRIS) image, and compares the classification results of GPCLs with those of KNN and SVMs in terms of several criteria such as overall accuracies and computational complexities.

### Formulation of Bayesian GP Models for Supervised Learning

Supervised learning tends to uncover a functional relationship $f(x)$ from a training dataset $D = \{(x_i, y_i)\}_{i=1}^n$ of $n$ pairs of observations where $x_i \in \mathbb{R}^d$ is the input vector, and $y_i$ the associated target value. In this study, $\mathbb{R}^d$ is the $d$-dimensional feature space of hyperspectral bands. To make inference on $f(\cdot)$ within a Bayesian framework, we must first elicit prior knowledge about the possible forms of $f(\cdot)$ in terms of a probability distribution over a function space. In Bayesian GP models, such a prior distribution is chosen as a GP,

$$f \sim p(f) = \text{GP}(m(x), k(x, x'))$$

where a GP is fully specified by a mean function $m(x)$ and a covariance function $k(x, x')$, i.e., $E[f(x)] = m(x)$ and $\text{Cov}(f(x), f(x')) = k(x, x')$ (Cressie, 1993). Moreover, a finite collection of function values indexed by any $x$’s also assumes a multivariate Gaussian distribution with its mean and covariance directly calculated from $m(\cdot)$ and $k(\cdot, \cdot)$, respectively. For example, the function values $f(X) = [f(x_1), f(x_2), \ldots, f(x_n)]^T$ at the observed inputs $X = [x_1, x_2, \ldots, x_n]^T$ have a Gaussian distribution $p(f|X)$ that has a mean vector $m(X) = [m(x_1), m(x_2), \ldots, m(x_n)]^T$ and an $n \times n$ covariance matrix $K(X, X)$ with $K_{ij} = k(x_i, x_j)$. Of note is that in the machine learning community, the covariance function $k(\cdot, \cdot)$ is often called a kernel function or a kernel. A kernel is often parameterized by a set of parameters $\theta$, i.e., $k(x, x'; \theta)$. Without the loss of generality, hereafter we assume $m(x) = 0$. Thus, to specify a GP prior of Equation 1, we only need to select a type of kernel function and specify the kernel parameters $\theta$.

Intuitively, a GP prior means that before looking at the training data, we believe that possible forms of $f(x)$ are random realizations drawn from the prior $p(f)$. An example about a 1-D $f(x)$ is illustrated in Figure 1a where the three curves are random draws from a GP prior $p(f)$ that uses the Gaussian kernel of Equation 11 with parameters $(\sigma_0, l)$ being $(0.5, 1.0)$. Before taking into account the training data (the circles), the three curves are equally probable a priori as candidates for $f$.

In light of the training data $D$, the GP prior of $f$ can be updated to a posterior distribution, i.e., $p(f|D)$. However, the observations $y = [y_i]_{i=1}^n$ generally do not equal the true

![Figure 1](image-url)
function values \( f(X) \), either due to measurement errors or the fact that in classification problems \( y \) takes discrete labels while \( f(x) \) takes continuous values; therefore, an observation or “noise” model \( p(y | f) \) needs to be specified for defining the probability of observing \( y \) when the true value is \( f \).

Then, according to the Bayes theorem, \( p(y | f) \) can update the CP prior \( p(f | X) \) to the posterior distribution as,

\[
p(f | \mathcal{D}, \theta) = \frac{p(y | f) p(f | X, \theta)}{p(\mathcal{D} | \theta)}
\]

where \( \theta \), the set of kernel parameters, is made explicit to show the inference’s dependence on it; \( p(y | f) \) is also known as likelihood in that it is a function of unknown \( f \) for a fixed set of observed \( y \), and \( p(\mathcal{D} | \theta) \) is marginal likelihood since it is a function of \( \theta \) given \( D \). In the case that a confident value of \( \theta \) cannot be obtained a priori, a sensible way is to learn an optimal value \( \theta \) in favor of the training data \( D \) by maximizing \( p(\mathcal{D} | \theta) \) with respect to \( \theta \). Be aware, however, that the desideratum of learning an informative value of \( \theta \) rather than an unreliable guess is achieved at the expense of extra computation for the involved optimization.

Once the posterior \( p(f | \mathcal{D}, \theta) \), available prediction at a new input \( x \), can be made by (Rasmussen and Williams, 2006):

\[
p(f | x, \mathcal{D}, \theta) = \int p(f | x, | \mathcal{D}, \theta) df
\]

which, in conjunction with the “noise” model, gives the predictive distribution of \( y \):

\[
p(y | x, \mathcal{D}, \theta) = \int p(y | f, \theta)p(f | x, \mathcal{D}, \theta) df.
\]

The general formulation of Bayesian GP models in Equations 1 through 4 can be used to deal with both classification and regression problems, depending on the choice of the likelihood model \( p(y | f) \). Before proceeding to GP classification, it is beneficial to first consider GP regression (GPR) problems because GPR could either be employed directly as a least squares classifier or be extended for classification by transforming regression outputs to class probabilities through a sigmoid response function as shall be seen soon.

**GP Regression (GPR) and Least Squares Classifiers**

Bayesian GPR assumes the Gaussian (normal) noise for the likelihood model, i.e., \( p(y | f) = N(f; \sigma^2) \) with \( \sigma^2 \) denoting the variance of the independent and identically-distributed noises. This very Gaussianity of \( p(y | f) \) makes the relevant posterior inference analytically tractable. As such, relevant distributions, such as the posterior \( p(f | \mathcal{D}, \theta) \) in Equation 2 and the predictive distributions of \( f \) and \( y \), in Equations 3 and 4, will all boil down to Gaussian ones. Specifically, the posterior mean and variance for \( f \), in Equation 3 can be explicitly expressed in terms of \( \mathcal{D} \) and \( k(\cdot, \cdot) \) as follows,

\[
\mu_f = k(x*, X)[K(X, X) + \sigma^2I]^{-1}y
\]

\[
\text{Var}(f) = k(x*, x*) - k(x*, X)[K(X, X) + \sigma^2I]^{-1}k(X, x*)
\]

where \( I \) is the \( n \times n \) identity matrix, \( k(X, X) \) is a \( 1 \times n \) vector with its \( i^{th} \) element being \( k(x_i, x) \), and we simply have \( k(X, X)^T = k(X, x*) \) (MacKay, 1997). To further illustrate GP posterior inference, Figure 1a shows that although the three curves are equally probable a priori, the dotted-dash curve (prior curve #1) favors the data points more than the other two and thus possesses a higher posterior probability of being the true function. Figure 1b depicts the posterior mean and variance that have been predicted with Equations 5 and 6 from the training data.

GPR models of Equations 5 and 6 can be used directly as least squares classifiers to solve classification problems in the same spirit as the early work of NNs (Richards and Jia, 1999). To illustrate, consider a binary classification problem with two classes \{+, −\}. First, the class labels are transformed to numerical values by assigning two real numbers \( y_+ \) and \( y_- \) to the “+” class and the “−” class, respectively; one such choice is \( y_+ = 1.0 \) and \( y_- = −1.0 \), which was employed for the subsequent experiments in this study.

Then, after fitting a GPR model to the transformed values of \( \pm 1.0 \), the class label of a new input \( x \), \( C(x) \), can be determined by thresholding \( \mu_x \), e.g., \( C(x) = \text{sgn}[\mu_x] \). Of note are that the predicted value \( \mu_x \) may fall outside the interval \( [y_-, y_+] \), and also that such an approach lacks a probabilistic interpretation. A remedy is to employ a sigmoid function to squash the GPR outputs into \([0, 1]\) to get class probabilities, as also used for SVMs (Platt, 2000).

**GP Classification (GPC)**

GPC refers to GP models that directly formulate classification problems, and it often needs a sigmoid function as the likelihood model in Equations 1 through 4 for explicitly modeling the class probabilities \( p(y | f) \). The role of sigmoid functions is to squash continuous values of the latent function \( f(x) \) into the unit interval \([0, 1]\), and the common choices for such sigmoid functions are the logistic and probit functions. The logistic function is written as:

\[
r_{\log}(f) = 1/(1 + \exp(-f)),
\]

and the probit function \( r_{\text{probit}}(f) \) is simply the standard normal cumulative distribution function. For binary classification, the likelihood term becomes \( p(y | f) = r(\tau \cdot f) \) where \( y \in \{+1, −1\} \).

The GPC posterior \( p(f | \mathcal{D}, \theta) \), however, is no longer a Gaussian and, more due to the non-Gaussianity of \( r(\tau \cdot f) \), thus making impossible the exact analytical posterior of GPC. In such a case, the Monte Carlo Markov Chain (MCMC) sampling is a standard procedure for posterior inference, but it is computationally daunting. Instead, we employ two widely used numerical approximation algorithms, i.e., the Laplace’s method (LP) and Expectation-propagation (EP) algorithm, for (Williams and Barber, 1998). The mode can be found by a second-order Taylor expansion to obtain an approximated GP posterior \( p(f | \mathcal{D}, \theta) \) in Equations 3 and 4 for analytical posterior inference. Next, we provide brief descriptions of the two numerical methods.

The LP method expands \( p(f | \mathcal{D}, \theta) \) around its mode \( \tilde{f} \) using a second-order Taylor expansion to obtain an approximated CP \( q(f | \mathcal{D}, \theta) \), i.e.,

\[
p(f | \mathcal{D}, \theta) = q(f | \mathcal{D}, \theta) \propto N(f | \tilde{f}, \mathbf{H}^{-1}) \times \exp(-\frac{1}{2}(f - \tilde{f})^T \mathbf{H}(f - \tilde{f}))
\]

where \( \tilde{f} = \arg \max f \{p(f | \mathcal{D}, \theta) \} \), and \( \mathbf{H} = -\nabla^2 \log p(f | \mathcal{D}, \theta) \) is the Hessian matrix of the negative log posterior at the mode (Williams and Barber, 1998). The mode \( \tilde{f} \) can be found by a gradient-based optimizer. In contrast to the LP method, the EP algorithm is more complicated in that it attempts to iteratively find a GP approximation to the true posterior \( p(f | \mathcal{D}, \theta) \) based on the sequential moment matching between approximated marginal distributions. Interested readers are referred to Rasmussen and Williams (2006) for more details on the implementation of the two methods. In general, both theoretical and experiential evidences have shown that the EP algorithm performs better than the LP method in terms of predictive distributions and marginal likelihood estimates, but
at the expense of more computation (Kuss and Rasmussen, 2005). However, the LP and SP methods are found comparable if only classifying class labels is concerned (Kuss and Rasmussen, 2005).

### Sparse GP Models

A typical GP model, either GPR or GPC, involves all training data for inference as indicated in Equation 4. Inspired by the sparsity of SVM, it is desirable to obtain a sparse GP model as well, which uses only part of the training data, i.e., support vectors (SV), for speeding up model training but without significant losses of inference accuracy. To help illustrate, let us first turn to the regularization-based formulation of kernel machines: Learning a kernel-based model of SVMS or GPs is equivalent to minimizing the functional:

$$f(f) = \frac{C}{2} \| f \|^2_2 + L(y, f)$$

(9)

where $\| f \|^2_2 = F^T K^{-1} F$ is a regularization term, $L(y, f)$ is the loss function, and $C$ is a scalar parameter that controls the trade-off between the two terms (Evgeniou et al., 2000). For the loss term $L(y, f)$, GPs use the negative log-likelihood function $-\log p(y|f)$ while SVMs use the hinge loss function $\Sigma (1 - y \cdot f)$. As depicted in Figure 2, the hinge loss function enjoys the sparsity property in that the training samples with zero loss, i.e., those with $y_i f_i \geq 1$, can be discarded without influencing the training of SVMs; however, this function resists a probabilistic interpretation and thus cannot be used as a likelihood term in GPC. Conversely, the probit and logistic functions, though being likelihood terms, cannot be used as a likelihood term in GPC. Therefore, given a kernel type, the selection of an appropriate GP model then boils down to determining sensible values of $\theta$. GPs offer a natural Bayesian paradigm to learn kernel parameters $\theta$ from the training data $\mathcal{D}$, as opposed to SVMS for which the kernel and tradeoff parameters are usually determined empirically or by a cross-validation approach.

### Model Selection of GPs

As stated above, a GP prior is fully specified by choosing a kernel function and its parameters $\theta$. Therefore, given a kernel type, the selection of an appropriate GP model then boils down to determining sensible values of $\theta$. GPs offer a natural Bayesian paradigm to learn kernel parameters $\theta$ from the training data $\mathcal{D}$, as opposed to SVMS for which the kernel and tradeoff parameters are usually determined empirically or by a cross-validation approach.

### Kernels and the Automatic Relevance Determination (ARD)

Several popular kernels for machine learning are the linear, polynomial, Gaussian and sigmoid kernels. In GPs, a kernel is parameterized by a set of hyperparameters $\theta$ as symbolized in Equation 2; for example, the Gaussian kernel has two hyperparameters $\theta = [\sigma_0, l]$, i.e.,

$$k_{\text{gaussian}}(x, x'; \sigma_0, l) = \sigma_0^2 \exp\left(-\frac{\|x - x'\|^2}{l^2}\right)$$

(11)

which is often called the radial basis function in SVMs. In GPs, a kernel is a covariance function, thus rendering $\theta$ more interpretable. For example, $l$ in Equation 11 is the characteristic length that determines a distance in the input space beyond which function values become less relevant. Moreover, if the characteristic length varies with input dimensions (i.e., input bands), we arrive at an interesting kernel known as the Automatic Relevance Determination (ARD):

$$k_{\text{ARD}}(x, x'; \sigma_0, l_1, l_2, \ldots, l_d) = \sigma_0^2 \exp\left(-\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{l_i^2}\right)$$

(12)

where the magnitude of $l_i$ indicates the inference ability of the $i^{th}$ input dimension, and large values of $l_i$’s can effectively downplay or eliminate the associated irrelevant input dimensions (Neal, 1996). Therefore, the ARD provides a parameterization scheme for automatic feature reduction especially when tackling high-dimensional problems such as classification of hyperspectral data.

Another kernel that gains special interest is the Neural Network (NN) covariance function. Its typical form as derived by Neal (1996) is:

$$k_{\text{NN}}(x, x'; \sigma_0, \tilde{x}) = \sigma_0^2 \sin^{-1}\left(\frac{2\tilde{x}_T \tilde{x}}{(1 + 2\tilde{x}_T \tilde{x})(1 + 2\tilde{x}_T \tilde{x})}\right)$$

(13)

where $\tilde{x} = [x_1, x_2, \ldots, x_d]^T$, and $\Sigma$ denotes a covariance matrix that may take structural parameterization. The importance of the NN kernel lies in that a GP model with this
kernel is equivalent to an infinite Bayesian NN with the probit transfer function.

**Learning Kernel Parameters**

In GP models, the sensible values of kernel parameters or hyperparameters \( \theta \) could be learned using Bayesian approaches. According to the Bayes theory, the posterior of \( \theta \) in light of training data \( D \) is modeled by:

\[
p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}
\]

where the marginal likelihood \( p(D|\theta) \) is given by:

\[
p(D|\theta) = \int p(y|f)p(f|\theta) \, df.
\]

The Maximum a posteriori (MAP) estimate of \( p(\theta|D) \), i.e., \( \hat{\theta} = \arg \max_{\theta} p(\theta|D) \), can be used as the optimal value of \( \theta \).

Because \( p(\theta) \) is often assumed to be flat due to the lack of prior information, the maximization of \( p(\theta|D) \) reduces to that of the marginal likelihood \( p(D|\theta) \). This procedure is called the type II maximum likelihood (ML-II) (Rasmussen et al., 2002). In GP models, \( p(D|\theta) \) is analytically available due to the Gaussianity of \( p(y|f) \), and in CPC models, it is approximated by using the Laplace and EP algorithms. In practice, the maximization of \( p(D|\theta) \) for \( \theta \) can be done with a general gradient-based optimizer. However, such a global optimization process may be trapped into local maxima especially if there are a large number of hyperparameters (e.g., when using the ARD for feature selection over high-dimensional inputs), and a common practice to find a better optimum is to perform the optimization several times with different random initial values and to select the trial with the highest marginal likelihood (Chu et al., 2003).

**Implementation Issues**

The direct implementation of GPs requires \( O(n^3) \) computation for training due to the inversion of the \( n \times n \) Gram matrix \( K \), which makes GPs less suitable for large-scale training data (e.g., \( n > 10,000 \)) (Rasmussen and Williams, 2006). So far, there have been many proposals for fast and efficient GP learning, such as the sparse model with the trigonometric loss function discussed above, sparse online learning, and the informative vector machines, among others (Csató and Opper, 2002; Lawrence et al., 2003; Kuus and Rasmussen, 2005). The typical strategy of fast and efficient GP learning is to select a small representative subset according to certain selection criteria, or in the simplest case, to make a random selection. In this study, except the sparse CPC model with the trigonometric loss function, we made no attempt to evaluate any complicated strategies for efficient GPs.

The major overhead in using a GP or SVM is the computation cost for learning hyperparameters (Chu et al., 2003), as illustrated later by the experiment results. To reduce such computation cost, it is practical and effective to first search for optimal hyperparameters with more iterations using a small randomly selected subset to get good initial values for the next learning phase with less iterations using a larger random subset, and such a procedure continues until all of the dataset are used, and the final optimum learned from the full dataset will be used for the posterior inference and predictions.

The aforementioned binary classification framework can be theoretically extended for multi-class problems; however, practical strategies exist for reducing a multi-class problem into a series of binary sub-problems that, when combined together, reconstruct the original problem (Hsu et al., 2002). Two such schemes are the one-against-all (OAA) and one-against-one (OAO) approaches. Specifically, consider an \( N \)-class problem.

1. The OAA approach constructs each binary problem by selecting one of the \( N \) classes as \( +1 \) and all the remaining classes as \(-1\), thus, resulting in \( N \) binary classifiers in total. In predication, the class label of a new data point will be assigned to the \( +1 \) class of the binary classifier with the highest \( +1 \) probability.

2. The OAO approach totally builds \( N(N-1)/2 \) binary classifiers from all combinatorial pairs of the \( N \) classes. In prediction, each binary classifier gives a vote to the winning class, and the class label will be chosen as the one with the highest accumulated votes.

Both the OAA and OAO approaches employ the “winner-takes-all” strategy. In case of a tie, i.e., that two or more classes have the same score in terms of probability or votes, the class with the highest prior probability can be used for class label assignment, and the prior probability can be estimated simply as the ratio of the sample size of a class to the entire sample size if training samples are collected randomly and independently (Melgan and Bruzzone, 2004). Although the OAO comprises a larger ensemble of classifiers than the OAA, the training with the OAO is usually faster because the training samples of an OAO binary classifier usually is smaller as compared to the OAA. It was also reported previously that, for classification using SVMs, the OAO yielded the best accuracy among several multi-class schemes that includes the OAA (Melgan and Bruzzozone, 2004). Hence, this study adopted the OAO scheme for both GP and SVM classification in the following experiments.

**Experiments**

The hyperspectral dataset used for assessment of CPC performance is a 145 pixels \( \times \) 145 pixels scene of the AVIRIS image with 220 bands, 20 of which are badly contaminated by atmospheric absorption (Figure 3). This image was acquired in June 1992 over the Indian Pines of northern Indiana, and is available online on the FTP site of the Laboratory for Applications of Remote Sensing (LARS) at Purdue University. This image has been also investigated for testing algorithms in several studies (Guaiti and Cromp, 1998; Melgan and Bruzzone, 2004). This scene mainly comprises forests and agriculture fields with several different crops. The companion ground reference data include totally 16 classes, but seven of them have very limited numbers of pixels; therefore, following Melgani and Bruzzone (2004), only the remaining nine classes are used for the subsequent experiments. A summary of the retained classes and their sample sizes is

![Figure 3. The AVIRIS hyperspectral image used for our experiments: (a) Band No. 100 and (b) Band No. 1 that appears noisy.](image-url)
listed in Table 1. To test the GPCLs, we designed three experiments, all of which were conducted in a Lenovo ThinkPad® T60 laptop equipped with 2.0 GHz CPU and the 1.0 GB RAM. Of particular note, the data were first centered with respect to the band means and then normalized by the band standard deviations for numerical benefits.

The classifiers we tested include a variety of GPCLs as well as the KNN and the SVM. For notational conveniences, we used the following conventions to refer to a specific classifier (e.g., GP_LS_GA, GP_LP_LG_GA, SVM_GA): A GPCL begins with “GP”, and the SVM with “SVM”. Among GPCLs, a least-squares classifier is denoted by “LS”, and a GPC classifier with sigmoid response functions is denoted by “LG” and “PB,” respectively, for the logistic and “probit” functions. As to the approximation schemes of a GPC classifier, we used “LP” and “EP” to stand for the LP method and the Expectation-Propagation algorithm, respectively. As to kernels, we used “GA,” “NN,” and “ARD” to indicate the Gaussian, the Neural Network, and the ARD kernels, respectively. We use “GP_TL_SMO”, specifically, to denote the sparse GP model with the Gaussian kernel and the trigonometric loss function because it employs the SMO algorithm for training. As to SVMs, we always used the Gaussian kernel due to its superior performances over other kernels as reported previously (Huang et al., 2002; Melgani and Bruzzone, 2004). In addition, we used “KNN-1” and “KNN-5” to represent the KNNs with one and five nearest neighbors, respectively. For ease of reference, the relevant acronyms for the kernel-based classifiers are summarized in Table 2. On the other hand, in all the experiments of this study, kernel parameters of GP models including “GP_TL_SMO” were learned by the ML-II method as described above, while kernel and trade-off parameters of SVMs were learned by referring to a five-fold cross-validation with the nested grid-searching method as detailed in Hsu et al. (2003) (see Figure 4).

### Table 1. Labels and Sizes of Nine Classes in the Classification Scheme

<table>
<thead>
<tr>
<th>Class</th>
<th>Class Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Corn-no till</td>
<td>1434</td>
</tr>
<tr>
<td>C2</td>
<td>Corn-min till</td>
<td>834</td>
</tr>
<tr>
<td>C3</td>
<td>Grass/Pasture</td>
<td>497</td>
</tr>
<tr>
<td>C4</td>
<td>Grass/Tree</td>
<td>747</td>
</tr>
<tr>
<td>C5</td>
<td>Hay-windrowed</td>
<td>489</td>
</tr>
<tr>
<td>C6</td>
<td>Soybean-no till</td>
<td>968</td>
</tr>
<tr>
<td>C7</td>
<td>Soybean-min till</td>
<td>2468</td>
</tr>
<tr>
<td>C8</td>
<td>Soybean-clean till</td>
<td>614</td>
</tr>
<tr>
<td>C9</td>
<td>Woods</td>
<td>1294</td>
</tr>
</tbody>
</table>

### Table 2. Acronyms for the Kernel Classifiers Evaluated in the Experiments

<table>
<thead>
<tr>
<th>Kernel machine</th>
<th>SVM</th>
<th>Support Vector Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td></td>
<td>Gaussian Process</td>
</tr>
<tr>
<td>GA</td>
<td></td>
<td>the Gaussian</td>
</tr>
<tr>
<td>NN</td>
<td></td>
<td>the Neural Network</td>
</tr>
<tr>
<td>ARD</td>
<td></td>
<td>the Automatic Relevance Determination</td>
</tr>
<tr>
<td>LS</td>
<td></td>
<td>the least-squares</td>
</tr>
<tr>
<td>LP</td>
<td></td>
<td>the Laplace’e method</td>
</tr>
<tr>
<td>EP</td>
<td></td>
<td>the Expectation-Propagation algorithm</td>
</tr>
<tr>
<td>LG</td>
<td></td>
<td>the logistic</td>
</tr>
<tr>
<td>PB</td>
<td></td>
<td>the probit</td>
</tr>
<tr>
<td>TL</td>
<td></td>
<td>the trigonometric loss function</td>
</tr>
<tr>
<td>Optimization</td>
<td>SMO</td>
<td>Sequential Minization Optimization for GP_TL and SVM</td>
</tr>
</tbody>
</table>

**Experiment 1**

The first experiment aims to compare several typical GP classifiers (GPCL) with the KNN and the SVM in terms of classification accuracy and computational load. The KNN is chosen as a reference because it is a most popular standard nonparametric classifier that also keeps fair performances, and the reason to choose SVM is because it reportedly claimed superior performances to most other classifiers, especially on high-dimensional data. In this experiment, following Gualtieri and Cromp (1998), we abandoned the 20 contaminated bands as a rudimentary way of coarse feature selection, thus leaving 200 bands as inputs.

Besides KNN and SVM, we assessed six GPCLs, including GP_LS_GA, GP_LP_LG_GA, GP_LP_PB_GA, GP_EP_pb_gA, and GP_TL_SMO. For comparisons of computational costs, we divided the training stage into two phases: one concerning the tuning or learning of parameters, and the other concerning the training by using the optimal parameters learned in the tuning phase. We timed the two phases separately for all the classifiers except KNN. In addition, the original data were randomly split into training and testing subsets. The size of training subset varied from 10 percent to 50 percent of the entire data with a 10 percent
abilities. The GPCLs considered in this experiment consist of ARD kernels in GPCLs as a means for feature selection, and the results were evaluated and compared against those of SVM. Unlike the first experiment that discarded the 20 noisy bands, the second experiment made use of all 220 bands by presuming that we have no prior information at all on the fact that the 20 bands are contaminated. Large characteristic lengths of the ARD kernel could downplay or even eliminate these irrelevant bands, thus allowing for discriminating between the effective and ineffective (e.g., noisy bands) bands. The 20 noisy bands are expected to have larger characteristic length because they contribute little or no discrimination abilities. The GPCLs considered in this experiment consist of GP_LS_ARD, GP_LP_ARD, GP_LP_BA, and GP_EP_PB_ARD. These four GPCLs used the ARD kernel with 220 characteristic length parameters that were optimized using the ML-II procedure as well. Meanwhile, the SVM-GA, i.e., the SVM with the Gaussian kernel, was also considered for comparison purposes. As to the scheme of splitting data, we used five sizes of training subsets randomly selected from the entire data as in the first experiment, and the percentages also ranged from 10 percent to 50 percent with a step of 10 percent, with the remaining pixels for testing. For each percentage, we only ran a single trial.

Experiment 3
The third experiment deals with a binary classification problem to assess the sensitivity of GPCLs to outliers present in the training data, as compared to the SVM. The two classes involved are Soybean-no till (C6) and Soybean-min till (C7); we chose these two classes because there is considerable spectral overlapping between them, which usually causes errors in collecting training samples especially when relying on the on-screen examination procedure for identifying class pixels. To control the amount of outliers, we purposefully introduced outlier pixels into each of the two classes by borrowing some percent of pixels from one of the two classes to be exchanged into the other class, and the outlier percentage we used ranged from 1 percent to 40 percent. In particular, two outlier percentages, the remaining pixels after selecting out the outlier pixels were split into a 20 percent training subset and an 80 percent testing subset. Then the outliers were combined into the 20 percent training subset to generate a mixed new training dataset that was used to train the classifiers. Note that the final training data contain outliers whereas the test data do not. In the experiment, we also discarded the 20 noisy bands. Along with the SVM, we tested seven GPCLs including GP_LS_GA, GP_LS_ARD, GP_LS_NN, GP_LP_AR, GP_LP_GA, GP_LP_BA, and GP_EP_ARD.

Results and Discussion
The kNN and all the GPCLs were implemented in Matlab (Mathworks, Inc.), except that the GP_TL_GA_SMO was implemented in C/C++ (Chang and Lin, 2001). The results of the above experiments are reported and discussed below in sequel.

Experiment 1
For each training data size, Table 3 summarizes the mean overall accuracies (OA) averaged over the ten random trials of that training size, together with the corresponding variation of OA among ten trials as represented by the standard deviation. The two kNN classifiers produced accuracies that were significantly lower than those of the kernel classifiers, with KNN-5 slightly better than KNN-1 across all training sizes. The poor performances of kNN could be attributed both to the ineffectiveness of the Euclidean distance as a metric to measure the proximity between points in the manifold of the high-dimensional training data, and to the inadequate number of training points relative to the high dimensions of feature space; these two factors make it difficult for kNN to estimate local properties of the feature space because only a sparse number of data points are found locally. On the other hand, the kernel classifiers including GP and SVM resort to the kernel trick that somehow mitigates this curse of dimensionality.

Among all the kernel classifiers we applied, GP LS NN consistently yielded the highest mean OA across all training data sizes, followed closely by the GP LS GA that was consistently better than the remaining classifiers. No kernel classifier always performed worst, although GP_TL_SMO produced the worst mean OA in three out of five training sizes (20 percent, 30 percent, 40 percent). In particular, two points are worthy of mentioning. First, although GP LS NN had the highest mean OA averaged over ten trials, it might have a lower OA than others at a single trial (e.g., 92.36 percent for GP LS NN versus 92.69 percent for SVM at one random trial using a training data size of 50 percent). Second, differences in the OA between two classifiers may be statistically insignificant; for example, at the training size of 50 percent, the gain of 0.61 percent by GP LS NN (93.72 percent) over

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Mean 10%</th>
<th>SD 10%</th>
<th>Mean 20%</th>
<th>SD 20%</th>
<th>Mean 30%</th>
<th>SD 30%</th>
<th>Mean 40%</th>
<th>SD 40%</th>
<th>Mean 50%</th>
<th>SD 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN-1</td>
<td>73.86</td>
<td>0.75</td>
<td>76.43</td>
<td>0.77</td>
<td>78.21</td>
<td>0.05</td>
<td>78.96</td>
<td>0.21</td>
<td>79.83</td>
<td>0.09</td>
</tr>
<tr>
<td>KNN-5</td>
<td>73.47</td>
<td>0.74</td>
<td>77.43</td>
<td>0.31</td>
<td>79.32</td>
<td>0.16</td>
<td>80.29</td>
<td>0.51</td>
<td>81.37</td>
<td>0.52</td>
</tr>
<tr>
<td>GP LS GA</td>
<td>85.48</td>
<td>0.52</td>
<td>89.27</td>
<td>0.47</td>
<td>91.27</td>
<td>0.24</td>
<td>92.62</td>
<td>0.36</td>
<td>93.11</td>
<td>0.41</td>
</tr>
<tr>
<td>GP LS NN</td>
<td>86.31</td>
<td>0.46</td>
<td>90.48</td>
<td>0.38</td>
<td>92.62</td>
<td>0.13</td>
<td>93.15</td>
<td>0.23</td>
<td>93.72</td>
<td>0.65</td>
</tr>
<tr>
<td>GP LP ARD</td>
<td>84.72</td>
<td>0.29</td>
<td>89.61</td>
<td>0.59</td>
<td>90.45</td>
<td>1.68</td>
<td>91.59</td>
<td>0.41</td>
<td>92.71</td>
<td>0.51</td>
</tr>
<tr>
<td>GP LP PB GA</td>
<td>83.86</td>
<td>0.28</td>
<td>89.42</td>
<td>0.57</td>
<td>91.08</td>
<td>0.19</td>
<td>91.56</td>
<td>0.42</td>
<td>91.75</td>
<td>1.33</td>
</tr>
<tr>
<td>GP EP PB GA</td>
<td>85.14</td>
<td>0.37</td>
<td>89.49</td>
<td>0.58</td>
<td>91.13</td>
<td>0.14</td>
<td>91.69</td>
<td>0.28</td>
<td>93.03</td>
<td>0.43</td>
</tr>
<tr>
<td>SVM GA</td>
<td>84.18</td>
<td>1.17</td>
<td>88.91</td>
<td>0.25</td>
<td>91.12</td>
<td>0.12</td>
<td>91.83</td>
<td>0.31</td>
<td>93.02</td>
<td>0.28</td>
</tr>
<tr>
<td>GP TL_SMO</td>
<td>84.36</td>
<td>0.13</td>
<td>89.19</td>
<td>0.69</td>
<td>90.57</td>
<td>0.34</td>
<td>91.26</td>
<td>0.28</td>
<td>92.53</td>
<td>0.24</td>
</tr>
</tbody>
</table>
GP_LS_GA (93.11 percent) was only marginally significant (p-value = 0.082) while a statistically significant gain is observed between GP_LS_NN (93.72 percent) and SVM (92.53 percent) with a p-value of 0.0008, both using the test statistics derived by assuming binomial distribution for the number of correctly classified pixels (van der Heijden et al., 2004). It is interesting to notice that the least-squares GP classifier (GP_LS_NN) yielded the best results, although it is more intuitive or natural to develop a classifier with the probit or logistic likelihood model as in GP_LP_LG or GP_LP_PB. In fact, previous studies reported that the regularized least-squares classifiers, including the least-squares GPCL of this study, e.g., GP_LS_NN, have comparable performances to SVMs (Rifkin and Klautau, 2004), and they are conceptually simpler and more straightforward to implement. Theoretical evidence also suggested that as long as the regression model of a least-squares classifier is flexible enough, the fitted model f(x) will converge to the class probability p(+|x) (or a linear scaling of it) when the number of training data is sufficiently large (Rasmussen and Williams, 2006). In addition, the NN kernel (covariance function) allows for good fitting to functions that saturate at two constant values in the opposite directions (e.g., a step function in 1-D case) (Neal, 1996), thus providing a possible explanation about why the GP_LS_NN tended to produce better results than GP_LS_GA since the data to be fit in this experiment had constant values (i.e., +1.0 and −1.0) on either side of the class boundary. Overall, for this experiment, we can securely conclude that all the GPCLs yielded accuracies that were better or at least comparable to that of SVM.

The computational expenses for each classifier are detailed in Table 4 where the “train” column indicates the CPU time consumed for training the classifiers with the “optimal” parameters learned in the “tuning” phase. Because learning parameters iteratively can take as much time as will be desired, in order to make fair comparisons, we attempted to keep training time per classifier to be the same order as much as possible, except for the GP_LS_GA since the data to be fit in this experiment had constant values (i.e., +1.0 and −1.0) on either side of the class boundary. Overall, for this experiment, we can securely conclude that all the GPCLs yielded accuracies that were better or at least comparable to that of SVM.

The computational costs for each classifier are averaged over ten trials in Experiment 1 for each classifier under all different sizes of training data: “tune”, “train”, and “predict” represent the CPU time in seconds consumed for learning parameters, for training with the learned optimal parameters, and for making prediction of a new pixel with the trained model, respectively. Note that all values are the average of ten random trials.

### Table 4. Computational Costs Averaged Over Ten Trials in Experiment 1 for Each Classifier under all Different Sizes of Training Data: “tune”, “train”, and “predict” represent the CPU Time in Seconds Consumed for Learning Parameters, for Training with the Learned Optimal Parameters, and for Making Prediction of a New Pixel with the Trained Model, Respectively. Note that all Values are the Average of Ten Random Trials

<table>
<thead>
<tr>
<th>Training Data Size</th>
<th>Phases</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tune</td>
<td>train</td>
<td>predict</td>
<td>tune</td>
<td>train</td>
<td>predict</td>
</tr>
<tr>
<td>KNN-1</td>
<td>—</td>
<td>—</td>
<td>5.6E-04</td>
<td>—</td>
<td>—</td>
<td>1.4E-03</td>
</tr>
<tr>
<td>KNN-5</td>
<td>—</td>
<td>—</td>
<td>4.5E-04</td>
<td>—</td>
<td>—</td>
<td>1.2E-03</td>
</tr>
<tr>
<td>GP_LS_GA</td>
<td>731</td>
<td>18</td>
<td>1.1E-01</td>
<td>3622</td>
<td>95</td>
<td>2.3E-01</td>
</tr>
<tr>
<td>GP_LS_NN</td>
<td>29</td>
<td>0.98</td>
<td>4.3E-03</td>
<td>101</td>
<td>3.4</td>
<td>5.9E-03</td>
</tr>
<tr>
<td>GP_LP_LG_GA</td>
<td>747</td>
<td>19</td>
<td>1.1E-01</td>
<td>3657</td>
<td>103</td>
<td>2.3E-01</td>
</tr>
<tr>
<td>GP_LP_PB_GA</td>
<td>747</td>
<td>19</td>
<td>1.1E-01</td>
<td>3657</td>
<td>103</td>
<td>2.3E-01</td>
</tr>
<tr>
<td>GP_LP_LG_PGA</td>
<td>747</td>
<td>19</td>
<td>1.1E-01</td>
<td>3657</td>
<td>103</td>
<td>2.3E-01</td>
</tr>
<tr>
<td>GP_EP_GA</td>
<td>951</td>
<td>50</td>
<td>1.1E-01</td>
<td>4407</td>
<td>361</td>
<td>2.1E-01</td>
</tr>
<tr>
<td>GP_TL_SMO</td>
<td>429</td>
<td>15</td>
<td>2.3E-03</td>
<td>1769</td>
<td>66</td>
<td>3.7E-03</td>
</tr>
<tr>
<td>SVM_GA</td>
<td>744</td>
<td>3.5</td>
<td>2.9E-03</td>
<td>2608</td>
<td>10</td>
<td>3.9E-03</td>
</tr>
</tbody>
</table>
Experiment 2

The GPCLs with ARD kernels improved the OAs over the SVM with the Gaussian kernel for all five training data sizes, and among the GPCLs, the GP_EP_ARD consistently provided the best OA. Also, all five classifiers including the SVM improved the OAs with increasing sizes of training data (Table 5). As compared to Experiment 1, the OAs for the SVM significantly decreased, e.g., the mean OAs of 84.18 percent and 73.89 percent for Experiment 1 and 2, respectively, when using the training data of size 10 percent. Such degradation was caused by the addition of the 20 atmospherically-contaminated bands. The 20 bands contain almost no information about spectral signatures of ground surfaces, therefore, somewhat confounding the training of SVMs. On the other hand, the GP classifiers used the ARD kernel to effectively determine these non-informative bands by referring to the magnitude of characteristic length parameters that were learned in light of the training data (Neal, 1996). However, the use of ARD kernel for the SVM remains impractical mainly due to the difficulty of the cross-validation procedure in pinpointing optimal parameters in a high-dimensional space (e.g., 221 in this case: one magnitude of variance of the kernel plus 220 characteristic lengths), unless the knowledge about kernel parameters is available a priori.

As an example, Figure 5 depicts the learned ARD characteristic length (scale) parameters for the binary classifier GP_LP_LG_ARD that discriminates C1 and C6 as applied to the training data of size 20 percent. After 15 iterations, the bands (x-axis) that protrude in the solid curve correspond to the 20 atmospheric-absorption bands [104–108, 150–163, 220] as well as the first band that is also quite noisy as displayed in Figure 3b. After 30 iterations, a more detailed curve of characteristic length was learned that favors bands at valleys while downplaying those at peaks. Afterwards, the convergence of the optimizer became slow as indicated by the small changes in curves from the 30 iterations to 45 iterations (Figure 5). This observation confirms the claim made previously that dozens of iterations are often sufficient to arrive at sensible values of kernel parameter in GP models when learned by a gradient-based optimizer.

Experiment 3

In this experiment, no single classifier outperformed others across all the trials, but the GPCLs with the ARD kernels consistently yielded better results than their counterparts with the Gaussian kernels as exemplified by the comparison between GP_LS_GA and GP_LS_ARD (Figure 6). Overall, though not always, the SVM showed poorer accuracies than the GPCLs, as partially demonstrated in Figure 6 where the differences in OA between the three GP least-squares classifiers and SVM were all significant except in the case with 9 percent outliers. More interestingly, for all the classifiers considered, the OA exhibited no overall decreasing trend with an increasing amount of outliers added into the training data. Contrarily, an overall trend slightly toward improving OA was observed as exhibited by the four classifiers in Figure 6, but none of these improvements were statistically significant, e.g., a mean OA increase of 0.2 percent for the GP_LS_GA (p-value = 0.45). The results suggest that in the experiment, all the kernel classifiers had low sensitivities to outliers in training data.

We also evaluated the GPC classifiers and the SVM with recourse to error-reject curves that can better characterize the performance of a classifier (MacKay, 2004). In the evaluation, the three GP least squares classifiers, i.e., GP_LS_GA, GP_LS_ARD and GP_LS_NN, are excluded because of their non-probabilistic prediction outputs, although ad-hoc methods exist that may use a sigmoid function to squeeze the least-squares output to the unit interval [0,1] for class probability outputs (Platt, 2000). It is found that the GP_EP_PB_ARD exhibited better error-reject characteristics across all ten trials, and this result is consistent with those of earlier studies (Rasmussen and Williams, 2006). The performance of SVM ranked consistently low, though not the worst, especially at small rejection rates.

Table 5. Overall Accuracy (%) for Different Sizes of Training Data in Experiment 2

<table>
<thead>
<tr>
<th>Training Data Size</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM_GA</td>
<td>73.89</td>
<td>84.8</td>
<td>86.43</td>
<td>88.40</td>
<td>90.12</td>
</tr>
<tr>
<td>GP_LS_ARD</td>
<td>78.75</td>
<td>87.8</td>
<td>89.33</td>
<td>90.31</td>
<td>91.19</td>
</tr>
<tr>
<td>GP_LP_LG_ARD</td>
<td>85.32</td>
<td>89.8</td>
<td>91.60</td>
<td>91.03</td>
<td>93.56</td>
</tr>
<tr>
<td>GP_LP_PB_ARD</td>
<td>87.63</td>
<td>90.4</td>
<td>91.18</td>
<td>91.93</td>
<td>93.82</td>
</tr>
<tr>
<td>GP_EP_ARD</td>
<td>89.94</td>
<td>90.8</td>
<td>91.96</td>
<td>92.59</td>
<td>93.95</td>
</tr>
</tbody>
</table>

Figure 5. The 220 characteristic length parameters of the ARD kernel learned at three iteration phases, plotted as a curve (y-axis in the logarithmic scale) against band numbers, for the binary classifier GP_LP_LG_ARD that discriminates C1 and C6. Note that a large characteristic length will downplay the associated input band.
Summary and Conclusion

In this paper, we proposed the use of GP models for classifying remotely sensed data, and evaluated the effectiveness of a variety of GPCLs as applied to the AVIRIS data. First, we provided a theoretical treatment to the Bayesian formulation of GPs and GPC; then briefly described the Laplace’s and the Expectation-Propagation algorithm to approximate the analytically intractable GPC models; and next presented the typical Bayesian approach to learning kernel hyperparameters of GP models. Some practical implementation issues of GPCLs were also discussed. Our experiments on the AVIRIS data demonstrated that GPCLs improve classification accuracies significantly over the conventional KNN classifier, and yield results comparable to or even better than the SVM.

GPs provide a useful kernel machine for supervised learning tasks such as regression and classification problems. One advantage of GP models over other kernel machines, such as SVM, is their natural Bayesian formulation by which the underlying latent function to be inferred is assigned a GP prior, thus making model outputs more probabilistically interpretable. More importantly, the Bayesian nature of GPs enables learning sensible kernel parameters in terms of maximizing the marginal likelihood by using general gradient-based optimizers, as opposed to the cross-validation procedures of SVMS that often prove difficult for optimizing parameters over high-dimensional spaces. Meanwhile, the GP models, either GPR or GPC, somehow circumvent the curse of dimensionality through the use of the kernel trick, but their model elicitation is based on statistical considerations rather than the geometrical consideration as taken in the classical formulation of SVMs that attempts to maximize margins between the separable planes in a kernel-induced space. Moreover, GP models allow for the use of the ARD kernel to automatically determine the relevant importance of input features (bands) so as to provide a feature selection mechanism; however, the ARD is less practical to be incorporated into SVMS due to the daunting computation involved for searching optimal parameters over high-dimensional spaces. The superior performance of GPs with the ARD kernel was confirmed in our second experiment.

Despite the aforementioned advantages, the training of GP models requires large computation especially when dealing with huge training datasets, and its naïve implementation scales \(O(n^3)\), which is slower than the fast training of SVMS. However, learning parameters of GP models usually
takes less iterations than that of SVMs, thus, somehow mitigating the high computation cost. Sparse GP models such as those with the trigonometric loss function could be used to reduce the computation cost because the training cost of these models is largely determined by the number of SVs in the same spirit as that of the fast training of SVMs. Indeed, the efficient and effective training of GP models currently remains an open area that attracts active research efforts, and there have been some advancements such as the informative vector machine (Lawrence et al., 2003) and the sparse online algorithm (Csató and Opper, 2002). Future studies may investigate the applicability of these sparse and efficient GP models for classification of remotely sensed data.

This study exploited the potential of GP models only for supervised classification of hyperspectral data. We envision that GPs should also provide a useful and attractive method for regression-related remote sensing applications, particularly due to such advantages as the Bayesian model adaptation, the mitigation of the curse of dimensionality, the automatic feature selection with ARD kernels, and the determination of error bars (prediction uncertainty) in making prediction. Hence, it is highly recommended that future studies should examine the utility of GPs as a regression tool for retrieving surface biophysical variables from remote sensing measurements, e.g., estimating chlorophyll concentration from hyperspectral data.

Acknowledgments

We thank Dr. David Landgrebe for providing the AVIRIS data and Drs. Wei Chu, Carl Edward Rasmussen, and Chris Williams for making their codes available; we also thank Mr. Jared Stukey for patient proofreading work. Our special thanks are due to the two guest editors for their constant support of the ERDAS Imagine® team during a summer internship in 2007 subsidized by the ASPRS Leica-geosystems Scholarship.

References


