Use of Gaussian Process Regression in Processing Scientific Data

J. Hájek

Aeronautical Research and Test Institute, Prague, Czech Republic.

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. We present the method of Gaussian Process Regression (GPR) and discuss the issues of training and prediction and their efficient realization. Further, we consider the application of GPR in design optimization and modeling experimental data.

Introduction

In many areas of science and engineering, we are interested in some physical or mathematical quantity that is known or assumed to be dependent on several other quantities. Often, we are not able to specify a closed-form mathematical model (with unknown parameters) of this dependence; we just assume it behaves "reasonably" and yet we want to model the dependent quantity based only on this assumption. This is where non-parametric regression techniques become important. In this article, we consider the Gaussian Process Regression non-parametric regression method. For a broad introduction into the topic, see e.g. *Rasmussen and Williams* [2006]. We introduce an open-source software package OctGPR implementing the GPR and discuss some efficiency and robustness implementation issues, and explain how these issues were solved in OctGPR. We then describe two types of application of GPR (and the software package itself) in processing scientific data. Finally, we provide a real-life example with processing experimental data from turbine measurements.

Gaussian Process Regression

Let us formulate the basic problem of non-parametric regression as follows: We are given a set of vectors $\boldsymbol{x}_i \in \mathbf{R}^d, i = 1, \ldots, n$, which we will call *inputs*, along with their corresponding *outputs* $y_i \in \mathbf{R}, i = 1, \ldots, n$. We shall call these the *training data*. Given a new input, $\hat{\boldsymbol{x}}$, we are interested in predicting the corresponding output $\hat{y}$ (which is unknown), based on the training data. Optionally, we may also be interested in some measure of uncertainty of the prediction.

The method of Gaussian process regression makes the assumption that the data can be well modeled as a sample of a Gaussian process

$$y(\boldsymbol{x}) = \mu(\boldsymbol{x}) + Z(\boldsymbol{x}) + Z_0 \tag{1}$$

Where $\mu(\boldsymbol{x})$ is a deterministic part that models the unconditional expectation of the process, $Z(\boldsymbol{x})$ is a Gaussian random field with zero mean and covariance

$$\mathrm{E}Z(\boldsymbol{x})Z(\boldsymbol{y}) = k(\boldsymbol{x}, \boldsymbol{y})$$

and Z_0 is a random variable called *noise*. To simplify matters, we shall make further assumptions that the mean is a constant plus a linear term

$$\mu(\boldsymbol{x}) = \mu_0 + \boldsymbol{\mu}^T \boldsymbol{x} \tag{2}$$

and the covariance is stationary, and depends only on scaled Euclidean distance

$$k(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 r(\|\boldsymbol{x} - \boldsymbol{y}\|_\theta^2) \tag{3}$$

where

$$\|\boldsymbol{x}-\boldsymbol{y}\|_\theta^2 = \sum_{k=0}^{d} \theta_k^2 (x_k - y_k)^2 \tag{4}$$

and $\sigma^2 = \mathrm{var} Z(\boldsymbol{x})$. We shall also denote $\sigma_0^2 = \mathrm{var} Z_0$ and $\nu = \sigma_0/\sigma$.

Let us briefly describe the basic GPR equations under these assumptions. We shall denote by $\mathbf{X}$, $\boldsymbol{y}$ the matrix of training inputs (one input per row) and the column vector of training outputs, respectively. We denote as $\mathbf{R}$ the correlation matrix with elements

$$R_{ij} = r(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_\theta^2), \quad i,j = 1 \ldots N. \tag{5}$$

and C the full covariance matrix

$$C = \sigma^2 \mathbf{R} + \sigma_0^2 \mathbf{I} = \sigma^2 (\mathbf{R} + \nu^2 \mathbf{I}) \tag{6}$$

where $\mathbf{I}$ is the identity matrix. Under these assumptions, given our training data, the prediction at a new point $\boldsymbol{x}_0$ is given by

$$\hat{y}(\boldsymbol{x}) = \mu_0 + \boldsymbol{\mu}^T \boldsymbol{x}_0 + \boldsymbol{r}(\boldsymbol{x})^T (\mathbf{R} + \nu^2 \mathbf{I})^{-1} (\boldsymbol{y} - \boldsymbol{m}) \tag{7}$$

where

$$\boldsymbol{m} = \boldsymbol{y} - \mu_0 - \mathbf{X}\boldsymbol{\mu} \tag{8}$$

and $\boldsymbol{r}(\boldsymbol{x})$ is a vector with elements

$$r_i = r(\|\boldsymbol{x} - \boldsymbol{x}_i\|_\theta^2).$$

This is actually a maximum likelihood estimate as well as a best linear unbiased predictor (BLUP). A very important feature of GPR models is that they are able to predict also the standard deviation of the prediction:

$$\hat{\sigma}^2(\boldsymbol{x}) = \sigma^2 - \sigma^2 \boldsymbol{r}(\boldsymbol{x})^T (\mathbf{R} + \nu^2 \mathbf{I})^{-1} \boldsymbol{r}(\boldsymbol{x}) \quad \{ +\sigma_0^2 \} \tag{9}$$

which can serve as a measure of certainty of the model's prediction, i.e. the model can assess how certain its prediction is at a particular test input. Note that the σ_0^2 term may be included to stress the fact that the prediction is never certain if noise is present (i.e. even repeated evaluations need not give identical results). Usually we ignore the noise in prediction variance, however.

Another attractive feature of GPR is the possibility of automatic tuning of adjustable parameters by maximum likelihood estimation (MLE). The parameters in question are $\boldsymbol{\theta}$, σ, σ_0, $\boldsymbol{\mu}$ and μ_0, usually called *hyperparameters*. Note that $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ are vectors, with individual hyperparameters being θ_k and μ_k, $k = 1, \ldots, d$. By using Bayesian statistics, we can evaluate the marginal likelihood of the data given certain hyperparameters. The marginal likelihood can be evaluated exactly, based on the assumption that $Z(\boldsymbol{x})$ is a Gaussian process:

$$\text{likelihood} = (2\pi)^{-n/2} \det(\mathbf{C})^{-1/2} \exp\left(-\frac{(\boldsymbol{y} - \boldsymbol{m})^T \mathbf{C}^{-1} (\boldsymbol{y} - \boldsymbol{m})}{2} \right) \tag{10}$$

This quantity tells us how likely are our data to occur given a particular model, and thus can serve as a score of how well the model *explains* the data. Note that the situation is fundamentally different from neural networks where a mean squared error is used which measures the quality of *fit* - any set of hyperparameters produces a GPR model that *fits* the data well (it fits exactly when $\nu = 0$), but the marginal likelihood allows us to discriminate between different GPR models.

We now seek to find the hyperparameters that maximize the likelihood; this is called maximum likelihood estimation. Because likelihoods are multiplicative and often very small or large numbers we usually prefer to work with their logarithms. And because minimization is more standard, we will formulate the training of a GPR model as minimizing the negative log likelihood. In this simplest form, we would have a $(2d + 3)$-dimensional optimization problem. Fortunately, it turns out that the negative log likelihood is quadratic w.r.t. m_0 and $\boldsymbol{m}$, so that these can be optimized exactly and thus eliminated from the optimization. Further, if we substitute $\sigma_0 = \sigma\nu$, σ can also be eliminated. By substituting in the optimal values, we obtain the so-called concentrated negative log likelihood (CNLL):

$$\mathrm{nll}(\boldsymbol{\theta}, \nu) = \frac{1}{2}\log\det(\mathbf{R}) + \frac{n}{2}\left(\log 2\pi + \log\bar{\sigma}^2\right) \tag{11}$$

$$\bar{\sigma}^2 = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{m})^T(\mathbf{R} + \nu^2\mathbf{I})^{-1}(\boldsymbol{y} - \boldsymbol{m}) \tag{12}$$

which we will seek to minimize.

Implementation

We have implemented the GPR method in the OctGPR package for the Octave computing environment. It is part of the Octave-Forge project, which comprises many additional Octave packages. The OctGPR package allows building GPR models with several types of correlation functions and selective inclusion of variables into the linear trend $\mu(\boldsymbol{x})$. Training of hyperparameters is realized by minimizing the concentrated negative log likelihood (12). Because its evaluation involves the factorization of a possibly large matrix, we want the optimization procedure to be efficient in terms of the number of factorizations required. To accomplish this, we use a gradient-based Quasi-Newton procedure, using matrix calculus to differentiate (12).

For efficiency and robustness, the inverse matrix $(\mathbf{R} + \nu^2\mathbf{I})^{-1}$ is not formed explicitly in prediction and CNLL calculation. Rather, the Cholesky factorization of the positive definite matrix $\mathbf{R}+\nu^2\mathbf{I}$ is used. Since the inverse is needed to evaluate derivatives efficiently, however, we later reuse the Cholesky factorization to turn it into an inverse. In this way, we ensure better accuracy for the CNLL itself, while maintaining maximum efficiency for derivatives computations. The derivative w.r.t. θ_k can be expressed as

$$\frac{\partial}{\partial\theta_k}\mathrm{nll} = \frac{1}{2}\mathrm{tr}((\mathbf{R} + \nu^2\mathbf{I})^{-1}\frac{\partial\mathbf{R}}{\partial\theta_k}) - \frac{1}{2}\frac{\boldsymbol{o}^T\frac{\partial\mathbf{R}}{\partial\theta_k}\boldsymbol{o}}{\bar{\sigma}^2} \tag{13}$$

where

$$\boldsymbol{o} = (\mathbf{R} + \nu^2\mathbf{I})^{-1}(\boldsymbol{y} - \boldsymbol{m}) \tag{14}$$

and $\partial\mathbf{R}/\partial\theta_k$ is an element-wise partial derivative of the matrix R, obtained by differentiating (5). The derivative w.r.t. ν^2 is

$$\frac{\partial}{\partial\nu^2}\mathrm{nll} = \frac{1}{2}\mathrm{tr}((\mathbf{R} + \nu^2\mathbf{I})^{-1}) - \frac{1}{2}\frac{\boldsymbol{o}^T\boldsymbol{o}}{\bar{\sigma}^2} \tag{15}$$

Here we use $\partial/\partial\nu^2$ because the resulting expressions are simpler; it is easy to turn the expressions into derivatives w.r.t. ν using the chain rule.

Note that once $(\mathbf{R} + \nu^2\mathbf{I})^{-1}$ is computed from the Cholesky factorization, the derivative w.r.t. each hyperparameter can be calculated efficiently in $O(n^2)$ time (see Appendix). The use of matrix calculus for efficient evaluation has been suggested in Rasmussen and Williams [2006] as well as several papers Rasmussen and Williams [1996], Seeger [2004]. However, these do not seem to use the concentrated likelihood to simplify the optimization. Thus, our method may be unique by using this combination. In fact we have gone yet further in this direction by noting that the *second* derivative w.r.t. ν^2 can also be evaluated efficiently given the precomputed inverse:

$$\frac{\partial^2}{(\partial \nu^2)^2}\mathrm{nll} = -\frac{1}{2}\mathrm{tr}((\mathbf{R}+\nu^2\mathbf{I})^{-2}) + \frac{\boldsymbol{o}^T(\mathbf{R}+\nu^2\mathbf{I})^{-1}\boldsymbol{o}}{\bar{\sigma}^2} - \frac{1}{2n}\frac{(\boldsymbol{o}^T\boldsymbol{o})^2}{\sigma^4} \tag{16}$$

The matrix calculus techniques used to derive these expressions can be found in Appendix. For technical details of the implementation, we refer the interested reader directly to OctGPR source code. Since ν is typically the most sensitive parameter, the exact second derivative is very helpful information for the optimization solver. It poses a problem, however: standard quasi-Newton optimization codes typically provide no option to exploit such an extra bit of information. That is part of the reason we decided to develop a custom optimization code for OctGPR.

The optimization code used for training hyperparameters in OctGPR is a custom quasi-Newton trust-region gradient solver. Trust-region techniques have reputation as providing an excellent combination of robust global and efficient local convergence. For more information on trust-region algorithms, see Nocedal and Wright [1999]. The algorithm always requires evaluating the gradient after the objective, which is very fit for our problem since it allows us to use just two level-3 (i.e. $O(n^3)$) LAPACK calls (`DPOTRF` and `DPOTRI`) to get the objective as well as all the derivatives. An approximate quadratic model is refined at each step using the SR1 (symmetric rank-1) update, again see Nocedal and Wright [1999]. However, the estimated second derivative w.r.t. ν is replaced by the exact value given by equation (15). The SR1 update doesn't require positive semidefiniteness of the quadratic model, and thus allows good approximations even in directions of negative curvature.

Another special feature of our trust-region algorithm is the form of the trust region. We note that (4) actually depends on θ_k^2, which means that θ_k need not be constrained to be positive - one simply corrects the sign after the optimization. In principle, the same can be said for ν; however, we often wish to still constrain ν for two reasons: first, ν acts as a safeguard against ill-conditioning of the matrix $\mathbf{R}+\nu^2\mathbf{I}$, so that we may want to keep it above a certain threshold if the (symmetric positive definite) matrix $\mathbf{R}$ tends to be ill-conditioned; and second, we sometimes have a guess for the minimum level of noise (for instance, the measurement error in our data). In fact, we never allow ν to become an exact zero in OctGPR training. For these reasons, we chose a rather special form of the trust-region subproblem, which can be called a *mixed norm* trust-region: In each step, choose $\delta\boldsymbol{\theta}$, $\delta\nu$ that minimize a local approximate quadratic model of the CNLL:

$$\frac{1}{2}\delta\boldsymbol{\theta}^T\mathbf{B}\delta\boldsymbol{\theta} + \delta\nu\boldsymbol{b}^T\delta\boldsymbol{\theta} + \frac{1}{2}a\delta\nu^2 + \boldsymbol{g}_\theta^T\delta\boldsymbol{\theta} + g_\nu\delta\nu \tag{17}$$

subject to

$$\|\delta\boldsymbol{\theta}\|_{\theta_R} \le \Delta \tag{18a}$$

$$\Delta_\nu^L \le \delta\nu \le \Delta_\nu^U \tag{18b}$$

where $\boldsymbol{\theta}_R$ are a reference length scales, computed as reciprocal standard deviations of the input variables. This strategy makes the training completely insensitive w.r.t. scaling of variables - a very desirable property.

Compared to the previous software version, which simply wrapped the famous `L-BFGS-B` code for the purpose of training, we have seen up to 5-times reduction in the number of factorizations (and consequently training time) and also improved robustness (the `L-BFGS-B` code often failed with abnormal termination in line search).

The OctGPR package is part of OctaveForge, can be downloaded from its website and installed into an existing Octave installation using Octave's package management functions. The URL is `http://octave.sourceforge.net/packages/octgpr.html`.

Applications

Design Optimization

The use of regression techniques in optimization has found much attention during the past decade or two. Basically, the idea is to combine the robustness of stochastic global search methods like genetic algorithms or evolutionary strategies (for overview of these methods, see Deb [2002]) with the efficiency of traditional gradient-based algorithms. The key point to note is that the gradient procedures gain their efficiency through building a model for the unknown function, whereas most stochastic techniques just compare function values. In the most general sense, a model-based optimization algorithm proceeds as follows:

1. Build a local model for the unknown function, along with a trust region for the model.
2. Optimize the local model within the trust region.
3. Validate the optimization result, update the local model, and restart unless terminating conditions are met.

It is evident that we shall aim to use a non-parametric regression technique in step 1 and a stochastic optimization algorithm in step 2. Note that in some cases, if we have sufficient data in advance and we do not require validation, it may be unnecessary to restart.

Measurement error estimation

Another interesting application of the GPR method is to use it for estimating measurement errors. When we measure the effects of certain phenomena in physics, we are often interested in an estimate of the errors of our measurement, without the necessity to repeat experiments. Usually this is done by fitting a parametric model to the measured data and using the residual errors to build an estimate of the measurement error. The problem with the parametric approach is that it is necessary to specify a correct parametric model for the data, i.e. describe the underlying physical process by an equation, otherwise a systematic error is introduced. If such an equation is not available (or is very complicated), we may, however, still estimate the measurement error using non-parametric data models. GPR, with its automatic MLE estimation of the hyperparameters, seems an ideal candidate for this task: the noise term in equation (1) can be readily associated with the measurement error. This means that we can estimate the measurement error (standard deviation of measurement) by training the GPR model on the measurement data and then calculate $\sigma_0 = \nu\sigma$.

A practical example

We have used the GPR method to process data from tunnel measurements of a rotor blade of a steam turbine. The measured quantity was the loss of entropy, dependent on the angle of attack and the inlet Mach number. An interested reader can find more details in Luxa et al. [2007] Since it was not possible to control Mach number exactly, the measurements are scattered in the second dimension, i.e. they are semi-gridded. Note that this makes no difference in GPR; the data could as well be completely scattered.

We analyse the data with two goals in mind: first, we want to locate local optima of the entropy loss, and second, we want an estimate of the measurement error. As we have explained in previous sections, both goals are achieveable using GPR. A grayscale map of the prediction of a GPR model trained for this experimental data is shown in Figure 1. The local minima can be readily indentified on this map; of course, their exact locations could be found by simple numerical methods.

The absolute error (i.e. the standard deviation) was estimated from the GPR model to be 0.25% (here the *absolute* error is expressed in % because the measured quantity is dimensionless and expressed in %).

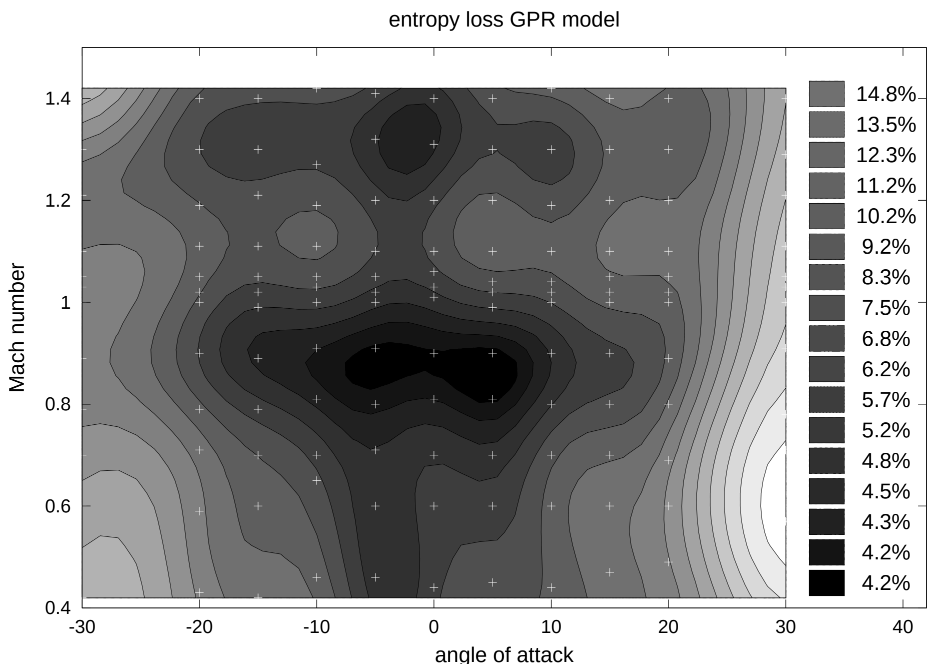

Figure 1. Map of entropy loss dependence on blade angle of attack and Mach number. Darker areas correspond to smaller loss.

Conclusions and future research directions

We have presented the GPR method for data modeling along with its efficient Open Source implementation, and shown some unique features of the implementation. We have also shown an application to real problem.

The Gaussian Process Regression has received much interest from the machine learning and statistical community during the past decade. It has proven itself to be a very effective tool for data analysis, often outperforming neural networks based on its mathematical elegance and insight. Because its principal disadvantage is the $O(N^3)$ scaling, many papers concerned with GPR seek ways how to improve this scaling (e.g., Seeger, Williams and Lawrence [2003]). Most of these focus on some finite-dimensional approximations of posterior distribution - for an excellent review, see Quiñonero-Candela and Rasmussen [2005].

We are currently investigating some completely new approach which does not use sparse approximation but rather a dense one with structure. This direction seems to hold much promise, but the research is far from being complete. It seems, however, that the fact that GPR considers the mutual influence of every pair of training inputs (and consequently, it needs to deal with a dense $N \times N$ matrix), is where it gets its good features from. It thus seems unlikely that we could reduce it much without losing some features (and the above mentioned papers support this idea). And indeed, if the data we analyse comes from expensive experimental measurements or expensive computations (e.g. CFD), we do not even expect to have more than, say, a thousand of measurements (and usually less), which the GPR can easily handle on today's computers. Another interesting option is using derivative information in GPR. This is not very complicated in the absence of noise but issues arise when noise is present. We shall aim to provide the possibility for specifying derivatives as well as relative noise variation in future versions of OctGPR.

References

Rasmussen C. E., Williams, C. K. I., Gaussian Processes for Machine Learning, *MIT Press*, 2006, ISBN 0-262-18253-X.

Rasmussen C. E., Williams, C. K. I., Gaussian Processes for Regression, *Advances in Neural Information Processing Systems*, 8, 1996.

Seeger, M., Gaussian Processes for Machine Learning, *International Journal of Neural Systems*, 14, 2004.

Nocedal, J., Wright, S. J., Numerical optimization, *Springer Verlag*, 1999, ISBN 0-387-98793-2.

Seeger, M., Williams, C. K. I., Lawrence, N. D., Fast Forward Selection to Speed Up Sparse Gaussian Process Regression, *Workshop on AI and Statistics*, 9, 2003.

Quiñonero-Candela, J, Rasmussen, C. E., A unifying view of sparse approximate Gaussian process regression, *Journal of Machine Learning Research*, 6, 2005.

Deb, K., Multi-Objective Optimization using Evolutionary Algorithms, *Wiley & Sons*, 4, 2002, ISBN 0-471-87339-X.

Luxa, M., Simurda, D., Synac, J., Safarik, P., Aerodynamics at off Design Performance of Root Section of Rotor Blade Last Stage of Large Output Steam Turbine, str.4/1-8, *Steam Turbines and Other Turbomachinery 2007*, SKODA POWER, Plzen, 2007.