Gaussian Processes in Machine Learning

Carl Edward Rasmussen

Max Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany
carl@tuebingen.mpg.de
WWW home page: http://www.tuebingen.mpg.de/~carl

Abstract. We give a basic introduction to Gaussian Process regression models. We focus on understanding the role of the stochastic process and how it is used to define a distribution over functions. We present the simple equations for incorporating training data and examine how to learn the hyperparameters using the marginal likelihood. We explain the practical advantages of Gaussian Process and end with conclusions and a look at the current trends in GP work.

Supervised learning in the form of regression (for continuous outputs) and classification (for discrete outputs) is an important constituent of statistics and machine learning, either for analysis of data sets, or as a subgoal of a more complex problem.

Traditionally parametric[1] models have been used for this purpose. These have a possible advantage in ease of interpretability, but for complex data sets, simple parametric models may lack expressive power, and their more complex counterparts (such as feed forward neural networks) may not be easy to work with in practice. The advent of kernel machines, such as Support Vector Machines and Gaussian Processes has opened the possibility of flexible models which are practical to work with.

In this short tutorial we present the basic idea on how Gaussian Process models can be used to formulate a Bayesian framework for regression. We will focus on understanding the stochastic process and how it is used in supervised learning. Secondly, we will discuss practical matters regarding the role of hyperparameters in the covariance function, the marginal likelihood and the automatic Occam's razor. For broader introductions to Gaussian processes, consult [1], [2].

1 Gaussian Processes

In this section we define Gaussian Processes and show how they can very naturally be used to define distributions over functions. In the following section we continue to show how this distribution is updated in the light of training examples.

[1] By a parametric model, we here mean a model which during training "absorbs" the information from the training data into the parameters; after training the data can be discarded.

Definition 1. *A Gaussian Process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions.*

A Gaussian *process* is fully specified by its mean function $m(x)$ and covariance function $k(x, x')$. This is a natural generalization of the Gaussian *distribution* whose mean and covariance is a vector and matrix, respectively. The Gaussian distribution is over vectors, whereas the Gaussian process is over functions. We will write:

$$f \sim \mathcal{GP}(m, k), \tag{1}$$

meaning: "the function f is distributed as a GP with mean function m and covariance function k".

Although the generalization from distribution to process is straight forward, we will be a bit more explicit about the details, because it may be unfamiliar to some readers. The individual random variables in a vector from a Gaussian distribution are indexed by their position in the vector. For the Gaussian process it is the argument x (of the random function $f(x)$) which plays the role of index set: for every input x there is an associated random variable $f(x)$, which is the value of the (stochastic) function f at that location. For reasons of notational convenience, we will enumerate the x values of interest by the natural numbers, and use these indexes as if they were the indexes of the process – don't let yourself be confused by this: the index to the process is x_i, which we have chosen to index by i.

Although working with infinite dimensional objects may seem unwieldy at first, it turns out that the quantities that we are interested in computing, require only working with finite dimensional objects. In fact, answering questions about the process reduces to computing with the related distribution. This is the key to why Gaussian processes are feasible. Let us look at an example. Consider the Gaussian process given by:

$$f \sim \mathcal{GP}(m, k), \quad \text{where} \quad m(x) = \tfrac{1}{4}x^2, \text{ and } k(x, x') = \exp(-\tfrac{1}{2}(x - x')^2). \tag{2}$$

In order to understand this process we can draw samples from the function f. In order to work only with finite quantities, we request only the value of f at a distinct finite number n of locations. How do we generate such samples? Given the x-values we can evaluate the vector of means and a covariance matrix using Eq. (2), which defines a regular Gaussian distribution:

$$\begin{aligned} \mu_i &= m(x_i) = \tfrac{1}{4}x_i^2, \quad i = 1, \ldots, n \quad \text{and} \\ \Sigma_{ij} &= k(x_i, x_j) = \exp(-\tfrac{1}{2}(x_i - x_j)^2), \quad i, j = 1, \ldots, n, \end{aligned} \tag{3}$$

where to clarify the distinction between process and distribution we use m and k for the former and μ and Σ for the latter. We can now generate a random vector from this distribution. This vector will have as coordinates the function values $f(x)$ for the corresponding x's:

$$\mathbf{f} \sim \mathcal{N}(\mu, \Sigma). \tag{4}$$

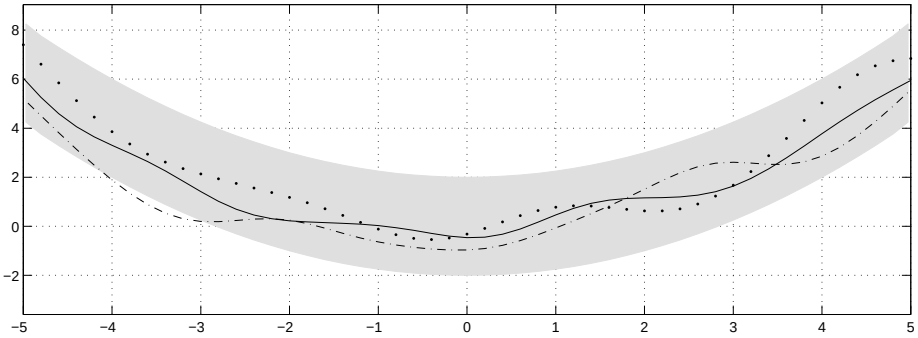

Fig. 1. Function values from three functions drawn at random from a GP as specified in Eq. (2). The dots are the values generated from Eq. (4), the two other curves have (less correctly) been drawn by connecting sampled points. The function values suggest a smooth underlying function; this is in fact a property of GPs with the squared exponential covariance function. The shaded grey area represent the 95% confidence intervals.

We could now plot the values of f as a function of x, see Figure 1. How can we do this in practice? Below are a few lines of Matlab[2] used to create the plot:

```
xs = (-5:0.2:5)'; ns = size(xs,1); keps = 1e-9;
m = inline('0.25*x.^2');
K = inline('exp(-0.5*(repmat(p'',size(q))-repmat(q,size(p'')))).^2)');
fs = m(xs) + chol(K(xs,xs)+keps*eye(ns))'*randn(ns,1);
plot(xs,fs,'.')
```

In the above example, m and k are mean and covariances; `chol` is a function to compute the Cholesky decomposition[3] of a matrix.

This example has illustrated how we move from process to distribution and also shown that the Gaussian process defines a distribution over functions. Up until now, we have only been concerned with random functions – in the next section we will see how to use the GP framework in a very simple way to make inferences about functions given some training examples.

2 Posterior Gaussian Process

In the previous section we saw how to define distributions over functions using GPs. This GP will be used as a *prior* for Bayesian inference. The prior does not depend on the training data, but specifies some properties of the functions; for

[2] Matlab is a trademark of The MathWorks Inc.

[3] We've also added a tiny `keps` multiple of the identity to the covariance matrix for numerical stability (to bound the eigenvalues numerically away from zero); see comments around Eq. (8) for a interpretation of this term as a tiny amount of noise.

example, in Figure 1 the function is smooth, and close to a quadratic. The goal of this section is to derive the simple rules of how to update this prior in the light of the training data. The goal of the next section is to attempt to learn about some properties of the prior[4] in the the light of the data.

One of the primary goals computing the posterior is that it can be used to make predictions for unseen test cases. Let $\mathbf{f}$ be the known function values of the training cases, and let $\mathbf{f}_*$ be a set of function values corresponding to the test set inputs, X_*. Again, we write out the joint distribution of everything we are interested in:

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\Sigma & \Sigma_*\\ \Sigma_*^\top & \Sigma_{**}\end{bmatrix}\right), \tag{5}$$

where we've introduced the following shorthand: $\boldsymbol{\mu} = m(x_i), i = 1, \ldots, n$ for the training means and analogously for the test means $\boldsymbol{\mu}_*$; for the covariance we use Σ for training set covariances, Σ_* for training-test set covariances and Σ_{**} for test set covariances. Since we know the values for the training set $\mathbf{f}$ we are interested in the conditional distribution of $\mathbf{f}_*$ given $\mathbf{f}$ which is expressed as[5]:

$$\mathbf{f}_*|\mathbf{f} \sim \mathcal{N}\big(\mu_* + \Sigma_*^\top \Sigma^{-1}(\mathbf{f} - \boldsymbol{\mu}),\ \Sigma_{**} - \Sigma_*^\top \Sigma^{-1}\Sigma_*\big). \tag{6}$$

This is the posterior distribution for a specific set of test cases. It is easy to verify (by inspection) that the corresponding posterior process is:

$$\begin{aligned} f|\mathcal{D} &\sim \mathcal{GP}(m_\mathcal{D}, k_\mathcal{D}),\\ m_\mathcal{D}(x) &= m(x) + \Sigma(X, x)^\top \Sigma^{-1}(\mathbf{f} - \mathbf{m})\\ k_\mathcal{D}(x, x') &= k(x, x') - \Sigma(X, x)^\top \Sigma^{-1}\Sigma(X, x'), \end{aligned} \tag{7}$$

where $\Sigma(X, x)$ is a vector of covariances between every training case and x. These are the central equations for Gaussian process predictions. Let's examine these equations for the posterior mean and covariance. Notice that the posterior variance $k_\mathcal{D}(x, x)$ is equal to the prior variance $k(x, x)$ minus a positive term, which depends on the training inputs; thus the posterior variance is always smaller than the prior variance, since the data has given us some additional information.

We need to address one final issue: noise in the training outputs. It is common to many applications of regression that there is noise in the observations[6]. The most common assumption is that of additive i.i.d. Gaussian noise in the outputs. In the Gaussian process models, such noise is easily taken into account; the

[4] By definition, the prior is independent of the data; here we'll be using a hierarchical prior with free parameters, and make inference about the parameters.

[5] the formula for conditioning a joint Gaussian distribution is:
$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}A & C\\ C^\top & B\end{bmatrix}\right) \implies \mathbf{x}|\mathbf{y} \sim \mathcal{N}\big(a + CB^{-1}(\mathbf{y} - b),\ A - CB^{-1}C^\top\big).$

[6] However, it is perhaps interesting that the GP model works also in the noise-free case – this is in contrast to most parametric methods, since they often cannot model the data exactly.

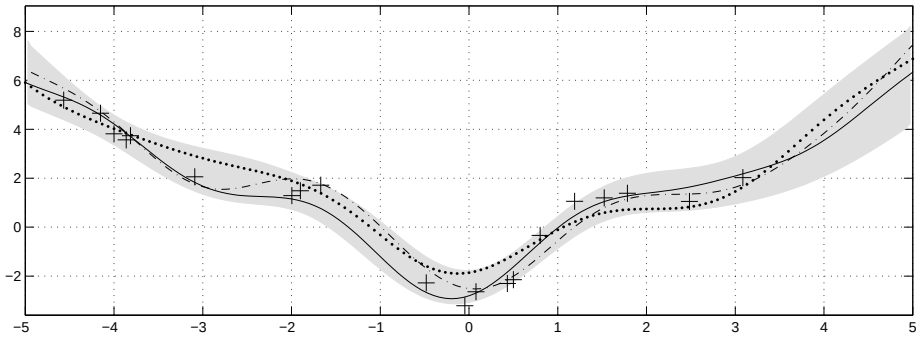

Fig. 2. Three functions drawn at random from the posterior, given 20 training data points, the $\mathcal{GP}$ as specified in Eq. (3) and a noise level of $\sigma_n = 0.7$. The shaded area gives the 95% confidence region. Compare with Figure 1 and note that the uncertainty goes down close to the observations.

effect is that every $f(x)$ has a extra covariance with itself only (since the noise is assumed independent), with a magnitude equal to the noise variance:

$$\begin{aligned} y(x) &= f(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma_n^2), \\ f &\sim \mathcal{GP}(m, k), \qquad y \sim \mathcal{GP}(m, k + \sigma_n^2 \delta_{ii'}), \end{aligned} \tag{8}$$

where $\delta_{ii'} = 1$ iff $i = i'$ is the Kronecker's delta. Notice, that the indexes to the Kronecker's delta is the identify of the cases, i, and not the inputs x_i; you may have several cases with identical inputs, but the noise on these cases is assumed to be independent. Thus, the covariance function for a noisy process is the sum of the signal covariance and the noise covariance.

Now, we can plug in the posterior covariance function into the little Matlab example on page 69 to draw samples from the posterior process, see Figure 2. In this section we have shown how simple manipulations with mean and covariance functions allow updates of the prior to the posterior in the light of the training data. However, we left some questions unanswered: How do we come up with mean and covariance functions in the first place? How could we estimate the noise level? This is the topic of the next section.

3 Training a Gaussian Process

In the previous section we saw how to update the prior Gaussian process in the light of training data. This is useful if we have enough prior information about a dataset at hand to confidently specify prior mean and covariance functions. However, the availability of such detailed prior information is not the typical case in machine learning applications. In order for the GP techniques to be of value in practice, we must be able to chose between different mean and covariance

functions in the light of the data. This process will be referred to as *training*[7] the GP model.

In the light of typically vague prior information, we use a hierarchical prior, where the mean and covariance functions are parameterized in terms of hyperparameters. For example, we could use a generalization of Eq. (2):

$$\begin{aligned} f \ &\sim \mathcal{GP}(m,\ k), \\ m(x) \ &= ax^2 + bx + c, \quad \text{and} \quad k(x, x') \ = \ \sigma_y^2 \exp\big(-\frac{(x - x')^2}{2\ell^2}\big) + \sigma_n^2 \delta_{ii'}, \end{aligned} \tag{9}$$

where we have introduced *hyperparameters* $\theta = \{a, b, c, \sigma_y, \sigma_n, \ell\}$. The purpose of this hierarchical specification is that it allows us to specify vague prior information in a simple way. For example, we've stated that we believe the function to be close to a second order polynomial, but we haven't said exactly what the polynomial is, or exactly what is meant by "close". In fact the discrepancy between the polynomial and the data is a smooth function plus independent Gaussian noise, but again we're don't need exactly to specify the characteristic length scale ℓ or the magnitudes of the two contributions. We want to be able to make inferences about all of the hyperparameters in the light of the data.

In order to do this we compute the probability of the data given the hyperparameters. Fortunately, this is not difficult, since by assumption the distribution of the data is Gaussian:

$$L \ = \ \log p(\mathbf{y}|\mathbf{x}, \theta) \ = \ -\tfrac{1}{2}\log|\Sigma| - \tfrac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{y} - \boldsymbol{\mu}) - \tfrac{n}{2}\log(2\pi). \tag{10}$$

We will call this quantity the log *marginal likelihood*. We use the term "marginal" to emphasize that we are dealing with a non-parametric model. See e.g. [1] for the weight-space view of Gaussian processes which equivalently leads to Eq. (10) after marginalization over the weights.

We can now find the values of the hyperparameters which optimizes the marginal likelihood based on its partial derivatives which are easily evaluated:

$$\begin{aligned} \frac{\partial L}{\partial \theta_m} \ &= \ -(\mathbf{y} - \boldsymbol{\mu})^\top \Sigma^{-1} \frac{\partial m}{\partial \theta_m}, \\ \frac{\partial L}{\partial \theta_k} \ &= \tfrac{1}{2}\operatorname{trace}\big(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}\big) + \tfrac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \frac{\partial \Sigma}{\partial \theta_k}\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}(\mathbf{y} - \boldsymbol{\mu}), \end{aligned} \tag{11}$$

where θ_m and θ_k are used to indicate hyperparameters of the mean and covariance functions respectively. Eq. (11) can conveniently be used in conjunction with a numerical optimization routine such as conjugate gradients to find good[8] hyperparameter settings.

[7] Training the GP model involves both model selection, or the discrete choice between different functional forms for mean and covariance functions as well as adaptation of the hyperparameters of these functions; for brevity we will only consider the latter here – the generalization is straightforward, in that marginal likelihoods can be compared.

[8] Note, that for most non-trivial Gaussian processes, optimization over hyperparameters is not a convex problem, so the usual precautions against bad local minima should be taken.

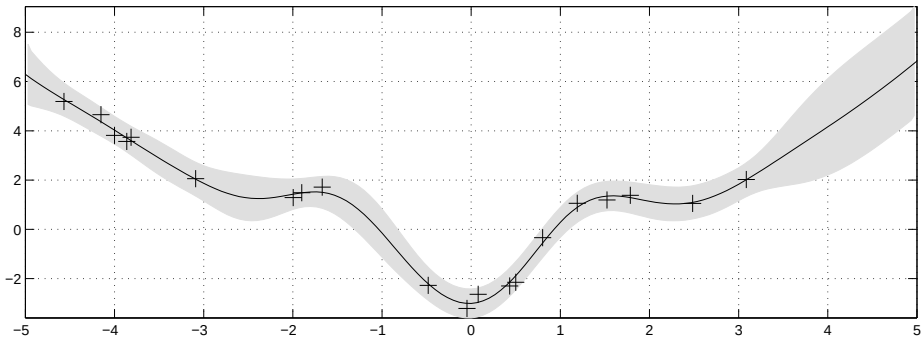

Fig. 3. Mean and 95% posterior confidence region with parameters learned by maximizing marginal likelihood, Eq. (10), for the Gaussian process specification in Eq. (9), for the same data as in Figure 2. The hyperparameters found were $a = 0.3, b = 0.03, c = -0.7, \ell = 0.7, \sigma_y = 1.1, \sigma_n = 0.25$. This example was constructed so that the approach without optimization of hyperparameters worked reasonably well (Figure 2), but there is of course no guarantee of this in a typical application.

Due to the fact that the Gaussian process is a non-parametric model, the marginal likelihood behaves somewhat differently to what one might expect from experience with parametric models. Note first, that it is in fact very easy for the model to fit the training data exactly: simply set the noise level σ_n^2 to zero, and the model produce a mean predictive function which agrees exactly with the training points. However, this is not the typical behavior when optimizing the marginal likelihood. Indeed, the log marginal likelihood from Eq. (10) consists of three terms: The first term, $-\frac{1}{2}\log|\Sigma|$ is a *complexity penalty term*, which measures and penalizes the complexity of the model. The second term a negative quadratic, and plays the role of a data-fit measure (it is the only term which depends on the training set output values $\mathbf{y}$). The third term is a log normalization term, independent of the data, and not very interesting. Figure 3 illustrates the predictions of a model trained by maximizing the marginal likelihood.

Note that the tradeoff between penalty and data-fit in the GP model is automatic. There is no weighting parameter which needs to be set by some external method such as cross validation. This is a feature of great practical importance, since it simplifies training. Figure 4 illustrates how the automatic tradeoff comes about.

We've seen in this section how we, via a hierarchical specification of the prior, can express prior knowledge in a convenient way, and how we can learn values of hyperparameters via optimization of the marginal likelihood. This can be done using some gradient based optimization. Also, we've seen how the marginal likelihood automatically incorporates Occam's razor; this property of of great practical importance, since it simplifies training a lot.

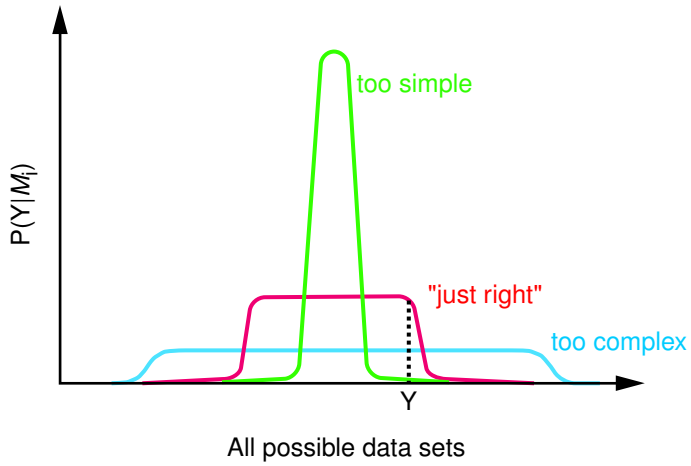

Fig. 4. Occam's razor is automatic. On the x-axis is an abstract representation of all possible datasets (of a particular size). On the y-axis the probability of the data given the model. Three different models are shown. A more complex model can account for many more data sets than a simple model, but since the probabilities have to integrate to unity, this means more complex models are automatically penalized more.

4 Conclusions and Future Directions

We've seen how Gaussian processes can conveniently be used to specify very flexible non-linear regression. We only mentioned in passing one type of covariance function, but in fact any positive definite function[9] can be used as covariance function. Many such functions are known, and understanding the properties of functions drawn from GPs with particular covariance functions is an important ongoing research goal. When the properties of these functions are known, one will be able to chose covariance functions reflecting prior information, or alternatively, one will be able to interpret the covariance functions chosen by maximizing marginal likelihood, to get a better understanding of the data.

In this short tutorial, we have only treated the simplest possible case of regression with Gaussian noise. In the case of non-Gaussian likelihoods (such as e.g. needed for classification) training becomes more complicated. One can resort to approximations, such as the Laplace approximation [3], or approximations based on projecting the non-Gaussian posterior onto the closest Gaussian (in a KL sense) [4] or sampling techniques [5].

Another issue is the computational limitations. A straightforward implementation of the simple techniques explained here, requires inversion of the covariance matrix Σ, with a memory complexity of $\mathcal{O}(n^2)$ and a computational complexity of $\mathcal{O}(n^3)$. This is feasible on a desktop computer for dataset sizes of n

[9] The covariance function must be positive definite to ensure that the resulting covariance matrix is positive definite.

up to a few thousands. Although there are many interesting machine learning problems with such relatively small datasets, a lot of current work is going into the development of approximate methods for larger datasets. A number of these methods rely on sparse approximations.

Acknowledgements

The author was supported by the German Research Council (DFG) through grant RA 1030/1.

References

1. Williams, C.K.I.: Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In Jordan, M.I., ed.: Learning in Graphical Models. Kluwer Academic (1998) 599–621
2. MacKay, D.J.C.: Gaussian processes — a replacement for supervised neural networks? Tutorial lecture notes for NIPS 1997 (1997)
3. Williams, C.K.I., Barber, D.: Bayesian classification with Gaussian processes. IEEE Transactions on Pattern Analysis and Machine Intelligence **20(12)** (1998) 1342–1351
4. Csató, L., Opper, M.: Sparse on-line Gaussian processes. Neural Computation **14** (2002) 641–668
5. Neal, R.M.: Regression and classification using Gaussian process priors (with discussion). In Bernardo, J.M., et al., eds.: Bayesian statistics 6. Oxford University Press (1998) 475–501

```
xs=(-5:0.2:5)';
ns=size(xs,1);
Keps=1e-9;
m=inline('0.25*x.^2');
K=inline('exp(-0.5*(repmat(p'',size(q))-repmat(q,size(p'')))).^2)');

%%%%%the confidence interval
Sigma=max(max(chol(K(xs,xs)+Keps*eye(ns))));
bxs=(-10:0.2:10)';
bup=0.25*bxs.^2+3*Sigma;
bbo=0.25*bxs.^2-3*Sigma;
up=area(bxs,bup,-5,'FaceColor',[0.8,0.8,0.8])
hold
bo=area(bxs,bbo,-5,'FaceColor',[1,1,1])
set(up,'linestyle',':')
set(bo,'linestyle',':')
%%%%%%

a=['+','o','*','x','s','d','p','h','#','$'];
for i=1:1:10
fx=m(xs)+chol(K(xs,xs)+Keps*eye(ns))'*randn(ns,1);
xlim([-5,5]);
plot(xs,fx,a(i))
end
```