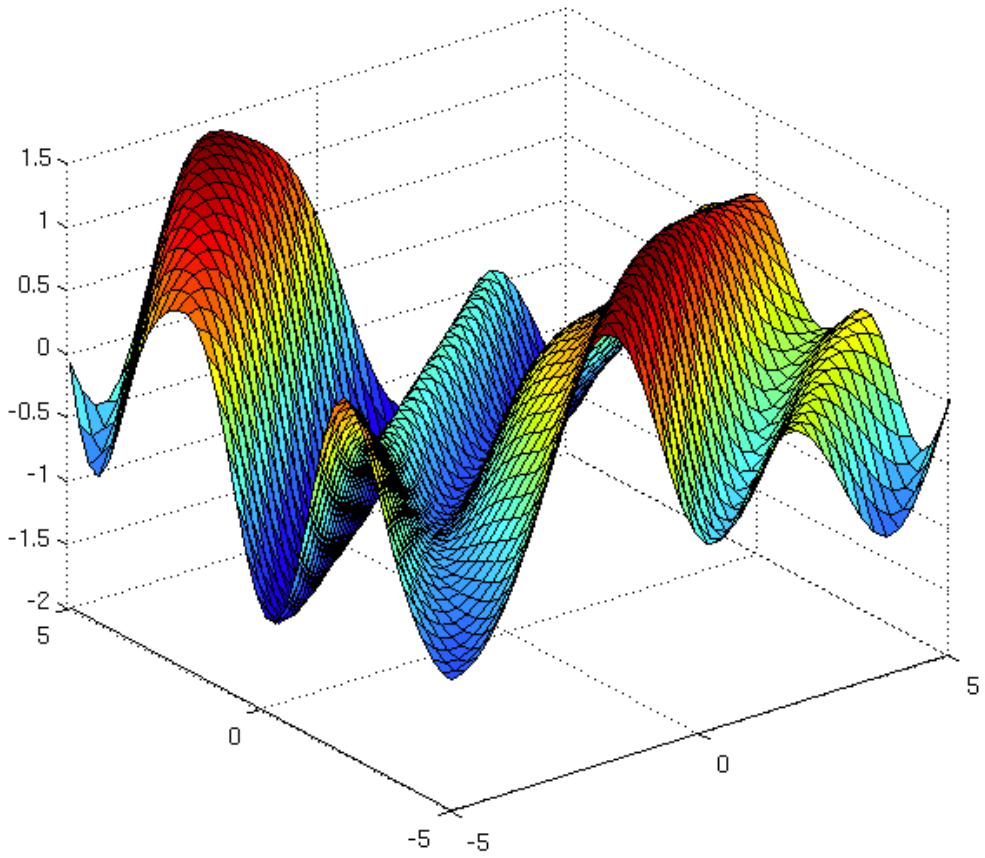
Katherine Bailey

AI, Machine Learning, Data Science, Language

Gaussian Processes for Dummies

Aug 9, 2016 · 10 minute read · 27 Comments



Source: The Kernel Cookbook by David Duvenaud

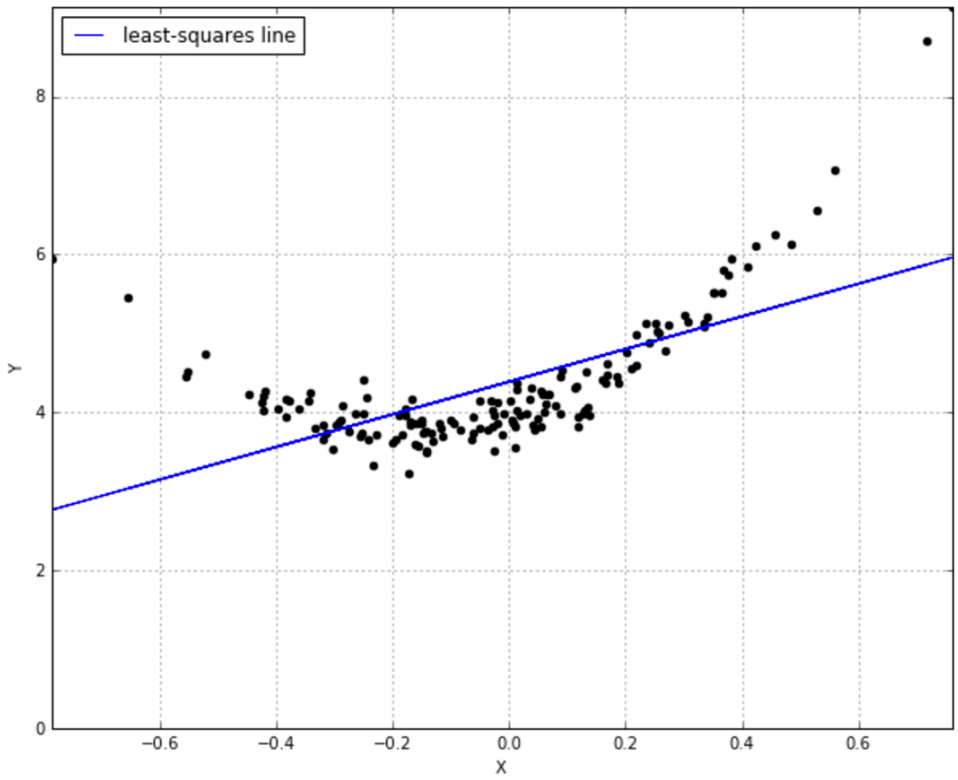
It always amazes me how I can hear a statement uttered in the space of a few seconds about some aspect of machine learning that then takes me countless hours to understand. I first heard about Gaussian Processes on an episode of the Talking Machines podcast and thought it sounded like a really neat idea. I promptly procured myself a copy of the classic text on the subject, Gaussian Processes for Machine Learning by Rasmussen and Williams, but my tenuous grasp on the Bayesian approach to machine learning meant I got stumped pretty quickly. That's when I began the journey I described in my last post, From both sides now: the math of linear regression.

Gaussian Processes (GPs) are the natural next step in that journey as they provide an alternative approach to regression problems. This post aims to present the essentials of GPs without going too far down the various rabbit holes into which they can lead you (e.g. understanding how to get the square root of a matrix.)

Recall that in the simple linear regression setting, we have a dependent variable y that we assume can be modeled as a function of an independent variable x, i.e. $y = f(x) + \epsilon$ (where ϵ is the irreducible error) but we assume further that the function f defines a linear relationship and so we are trying to find the parameters θ_0 and θ_1 which define the intercept and slope of the line respectively, i.e. $y = \theta_0 + \theta_1 x + \epsilon$. Bayesian linear regression provides a probabilistic approach to this by finding a distribution over the parameters that gets updated whenever new data points are observed. The GP approach, in contrast, is a *non-parametric* approach, in that it finds a distribution over the possible **functions** $f(x)$ that are

consistent with the observed data. As with all Bayesian methods it begins with a prior distribution and updates this as data points are observed, producing the posterior distribution over functions.

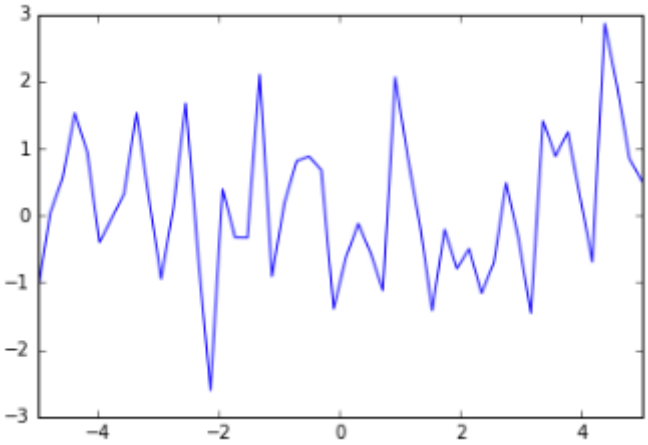
To get an intuition about what this even means, think of the simple OLS line defined by an intercept and slope that does its best to fit your data.



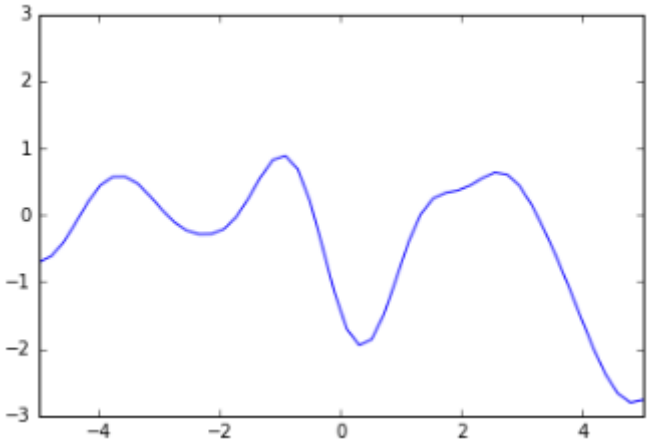
The problem is, this line simply isn't adequate to the task, is it? You'd really like a curved line: instead of just 2 parameters θ_0 and θ_1 for the function $\hat{y} = \theta_0 + \theta_1 x$ it looks like a quadratic function would do the trick, i.e. $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$. Now we'd need to learn 3 parameters. But what if we don't want to specify upfront how many parameters are involved? We'd like to consider every possible function that matches our data, with however many parameters are involved. That's what non-

parametric means: it's not that there aren't parameters, it's that there are infinitely many parameters.

But of course we need a prior before we've seen any data. What might that look like? Well, we don't really want ALL THE FUNCTIONS, that would be nuts. So let's put some constraints on it. First of all, we're only interested in a specific domain — let's say our x values only go from -5 to 5. Now we can say that within that domain we'd like to sample functions that produce an output whose mean is, say, 0 and that are *not too wiggly*. Here's an example of a very wiggly function:



And here's a much smoother function:



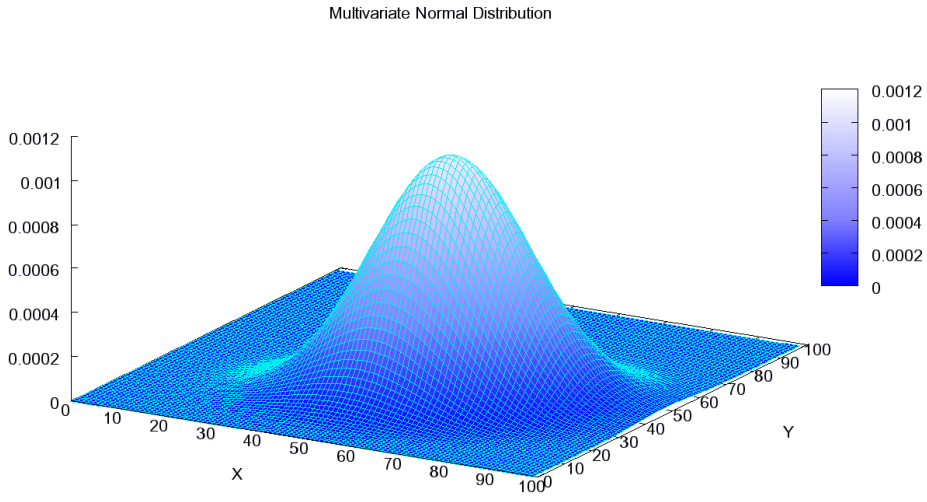
There's a way to specify that smoothness: we use a **covariance matrix** to ensure that values that are close together in input space will produce output values that are close together. This covariance matrix, along with a mean function to output the expected value of $f(x)$ defines a Gaussian Process.

Here's how Kevin Murphy explains it in the excellent textbook Machine Learning: A Probabilistic Perspective:

> A GP defines a prior over functions, which can be converted into a posterior over functions once we have seen some data. Although it might seem difficult to represent a distrubtion over a function, it turns out that we only need to be able to define a distribution over

> the function's values at a finite, but arbitrary, set of points, say $x_1, \ldots, x_N$. A GP assumes that $p(f(x_1), \ldots, f(x_N))$ is jointly Gaussian, with some mean $\mu(x)$ and covariance $\sum(x)$ given by $\sum_{ij} = k(x_i, x_j)$, where k is a positive definite kernel function. The key idea is that if x_i and x_j are deemed by the kernel to be similar, then we expect the output of the function at those points to be similar, too.

The mathematical crux of GPs is the multivariate Gaussian distribution.



Source: Wikipedia

It's easiest to imagine the bivariate case, pictured here. The shape of the bell is determined by the covariance matrix. If we imagine looking at the bell from above and we see a perfect circle, this means these are two independent normally distributed variables — their covariance is 0. If we assume a variance of 1 for each of the independent variables, then we get a covariance matrix of $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The diagonal will simply hold the variance of each variable on its own, in this case both 1's. Anything other than 0 in the top right would be mirrored in the bottom left and would indicate a correlation between the variables. This would give the bell a more oval shape when looking at it from above.

If we have the joint probability of variables x_1 and x_2 as follows:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} \right)$$

it is possible to get the *conditional* probability of one of the variables *given* the other, and **this is how, in a GP, we can derive the posterior from the prior and our observations**. It's just that we're not just talking about the joint probability of two

variables, as in the bivariate case, but the joint probability of the values of $f(x)$ for all the x values we're looking at, e.g. real numbers between -5 and 5.

So, our posterior is the joint probability of our outcome values, some of which we have observed (denoted collectively by f) and some of which we haven't (denoted collectively by f_*):

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix}\right)$$

Here, K is the matrix we get by applying the kernel function to our observed x values, i.e. the similarity of each observed x to each other observed x. K_* gets us the similarity of the training values to the test values whose output values we're trying to estimate. K_{**} gives the similarity of the test values to each other.

I'm well aware that things may be getting hard to follow at this point, so it's worth reiterating what we're actually trying to do here. There are some points x for which we have observed the outcome $f(x)$ (denoted above as simply f). There are some points x_* for which we would like to estimate $f(x_*)$ (denoted above as f_*). So we are trying to get the probability distribution $p(f_*|x_*, x, f)$ and we are assuming that f and f_* together are jointly Gaussian as defined above.

About 4 pages of matrix algebra can get us from the joint distribution $p(f, f_*)$ to the conditional $p(f_*|f)$. I am conveniently going to skip past all that but if you're interested in the gory details then the Kevin Murphy book is your friend. At any rate, what we end up with are the mean, μ_* and covariance matrix Σ_* that define our distribution $f_* \sim \mathcal{N}(\mu_*, \Sigma_*)$

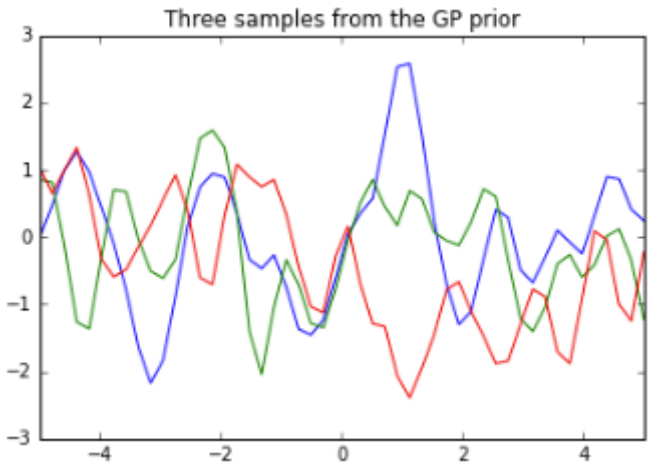
Now we can sample from this distribution. Recall that when you have a univariate distribution $x \sim \mathcal{N}(\mu, \sigma^2)$ you can express this in relation to *standard normals*, i.e. as $x \sim \mu + \sigma(\mathcal{N}(0, 1))$. And generating standard normals is something any decent mathematical programming language can do (incidently, there's a very neat trick involved whereby uniform random variables are projected on to the CDF of a normal distribution, but I digress...) We need the equivalent way to express our multivariate normal distribution in terms of standard normals: $f_* \sim \mu + B\mathcal{N}(0, I)$, where B is the matrix such that $BB^T = \Sigma_*$, i.e. the square root of our covariance matrix. We can use something called a Cholesky decomposition to find this.

OK, enough math — time for some code. The code presented here borrows heavily from two main sources: Nando de Freitas' UBC Machine Learning lectures (code for GPs can be found here) and the PMTK3 toolkit, which is the companion code to Kevin Murphy's textbook Machine Learning: A Probabilistic Perspective.

Below we define the points at which our functions will be evaluated, 50 evenly spaced points between -5 and 5. We also define the kernel function which uses the

Squared Exponential, a.k.a Gaussian, a.k.a. Radial Basis Function kernel. It calculates the squared distance between points and converts it into a measure of similarity, controlled by a tuning parameter. Note that we are assuming a mean of 0 for our prior.

```
import numpy as np
import matplotlib.pyplot as pl

# Test data
n = 50
Xtest = np.linspace(-5, 5, n).reshape(-1,1)

# Define the kernel function
def kernel(a, b, param):
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a,
    return np.exp(-.5 * (1/param) * sqdist)

param = 0.1
K_ss = kernel(Xtest, Xtest, param)

# Get cholesky decomposition (square root) of the
# covariance matrix
L = np.linalg.cholesky(K_ss + 1e-15*np.eye(n))
# Sample 3 sets of standard normals for our test points,
# multiply them by the square root of the covariance matrix
f_prior = np.dot(L, np.random.normal(size=(n,3)))

# Now let's plot the 3 sampled functions.
pl.plot(Xtest, f_prior)
pl.axis([-5, 5, -3, 3])
pl.title('Three samples from the GP prior')
pl.show()
```

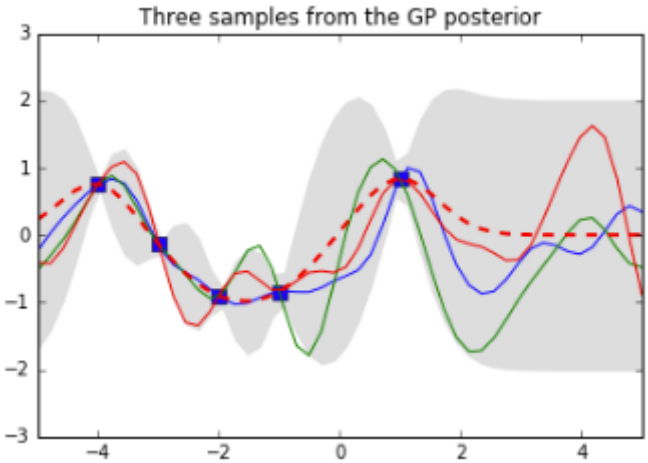


Note that the K_ss variable here corresponds to K_{**} in the equation above for the joint probability. It will be used again below, along with K and K_*

Now we'll observe some data. The actual function generating the y values from our x values, unbeknownst to our model, is the sin function. We generate the output

at our 5 training points, do the equivalent of the above-mentioned 4 pages of matrix algebra in a few lines of python code, sample from the posterior and plot it.

```
# Noiseless training data
Xtrain = np.array([-4, -3, -2, -1, 1]).reshape(5,1)
ytrain = np.sin(Xtrain)

# Apply the kernel function to our training points
K = kernel(Xtrain, Xtrain, param)
L = np.linalg.cholesky(K + 0.00005*np.eye(len(Xtrain)))

# Compute the mean at our test points.
K_s = kernel(Xtrain, Xtest, param)
Lk = np.linalg.solve(L, K_s)
mu = np.dot(Lk.T, np.linalg.solve(L, ytrain)).reshape((n,))

# Compute the standard deviation so we can plot it
s2 = np.diag(K_ss) - np.sum(Lk**2, axis=0)
stdv = np.sqrt(s2)
# Draw samples from the posterior at our test points.
L = np.linalg.cholesky(K_ss + 1e-6*np.eye(n) - np.dot(Lk.T, Lk))
f_post = mu.reshape(-1,1) + np.dot(L, np.random.normal(size=(n,3)))

pl.plot(Xtrain, ytrain, 'bs', ms=8)
pl.plot(Xtest, f_post)
pl.gca().fill_between(Xtest.flat, mu-2*stdv, mu+2*stdv, color="#dddddd")
pl.plot(Xtest, mu, 'r--', lw=2)
pl.axis([-5, 5, -3, 3])
pl.title('Three samples from the GP posterior')
pl.show()
```



See how the training points (the blue squares) have “reined in” the set of possible functions: the ones we have sampled from the posterior all go through those points. The dotted red line shows the mean output and the grey area shows 2 standard deviations from the mean. Note that this is 0 at our training points (because we did not add any noise to our data). Also note how things start to go a bit wild again to the right of our last training point $x = 1$ — that won’t get reined in until we observe some data over there.

This has been a very basic intro to Gaussian Processes — it aimed to keep things as simple as possible to illustrate the main idea and hopefully whet the appetite for a more extensive treatment of the topic such as can be found in the Rasmussen and Williams book.

(@bayesianmixer) via VC, do let me know if you're interested, Jon

Reply • Share ›

katbailey Mod ➚ Jonathan Sedar • 5 years ago

It did indeed :) https://www.meetup.com/Bost...
I'd be happy to present it for the Bayesian Mixer group, thanks for the invitation! Let's follow up via DM to figure out when as I have a lot on at work over the next couple of months.

Reply • Share ›

Ashwani Kumar ➚ katbailey • 4 years ago

Such an amazing description of GPs. I thank you a million times :). Also, if possible, could you please also provide any of your talk (presentation) videos?

Reply • Share ›

Karro • 4 years ago

Thank you. I'v spent half the day looking through papers and sources that I hoped would give me the intuitive idea behind GPs without getting bogged down in the math that I don't want to work through. This was the only description I found that fit. Fantastic.

4 2 • Reply • Share ›

Mohammad Saber • a year ago

This is a really great explanation of a difficult topic. Thanks a lot.
I have a question about the mean output at training data points. Mean value curve passes through the training points (looks like Interpolation). But, in python library scikit-learn tutorial and examples, this curve doesn't pass through training point (looks like regression). I am a little confused about it.

Ref:
https://scikit-learn.org/st...

1 • Reply • Share ›

Mike • 3 years ago

Thanks for this blog post. It is always a true contribution when someone who can actually write and think clearly elucidates a technical topic.

1 • Reply • Share ›

Tom Oberheiden • 3 years ago

This is a really great explanation of a difficult topic. Great stuff. Thanks! I enjoyed reading it from the start.

1 • Reply • Share ›

umber • 4 years ago

can you explain this line
I k = np.linalg.solve(L, K_s)

LK = np.linalg.solve(L, K_s)

1 ^ | v • Reply • Share ›

Numair Mansur • 4 years ago

awesome

1 ^ | v • Reply • Share ›

Mike Croucher • 4 years ago

Nice intro! May I ask where the 0.00005 comes from in

L = np.linalg.cholesky(K + 0.00005*np.eye(len(Xtrain)))

please?

1 ^ | v • Reply • Share ›

katbailey Mod → Mike Croucher • 4 years ago

Thanks Mike! Regarding your question, that appears to be some noise I am adding to the training points, even though the comment above says "noiseless training data". You often need to add some "jitter" anyway to ensure that the matrix will be positive semi-definite for cholesky decomp, but the number seems high for that. I should probably just remove the comment about the data being noiseless and add an explanation...

3 ^ | v • Reply • Share ›

David Refaeli • 10 months ago • edited

You could have also just do:

```
f_prior = np.random.multivariate_normal(mean=np.zeros(n), cov=K_ss, size=3).T
```

For for the posterior, you need to invert the Kernel matrix:

```
# Apply the kernel function to our training points
K = kernel(Xtrain, Xtrain, param)
K_inv = np.linalg.inv(K)
# and to our mutual
K_s = kernel(Xtrain, Xtest, param)
# compute mean
mu = K_s.T@K_inv@ytrain.squeeze()
# compute covariance matrix, and corresponding stdv (sqrt diagonal values)
cov = K_ss-K_s.T@K_inv@K_s
stdv = np.sqrt(np.diag(cov)).squeeze()
# Draw samples from the posterior at our test points.
f_post = np.random.multivariate_normal(mean=mu, cov=cov, size=3).T
```

But instead of inverse, one could use Cholesky decomposition, which is faster

see more

^ | v • Reply • Share ›

Bharath • 3 years ago

This is the best introduction to GP.

There is also another blog post with a bit more focus on the visualization of the intermediary steps.

https://distill.pub/2019/vi...

Reply • Share ›

Mohamed ahmed • 3 years ago

What is the difference between the observed, training and test data?

Reply • Share ›

JohnCwok • 3 years ago

I'm having trouble at "define a prior over functions" is it meant that our prior is a probability distribution over the space of possible functions ? because then the rest of the post doesn't really make sense if it is like that.

Reply • Share ›

dana ballard • 3 years ago

you're the GOAT!

Reply • Share ›

Kuo-I • 3 years ago

Thank you for such a great explanation and examples, they are pretty inspiring!

Reply • Share ›

Pierre Villeneuve • 3 years ago

Your write up is a great read. What do you think about applying GP methods to making a forward prediction of a vector given a history of prior observations? I'd like to make a prediction of my data vector before I measure it, and then compared the two. Just like a Kalman Filter application.

Reply • Share ›

Stijn Decubber • 3 years ago

Nice explanation, thanks!

Reply • Share ›

Manikandan S • 3 years ago

The best explanation is the most simple one. I cannot thank you enough for this, really great job!

Reply • Share ›

Leti • 3 years ago

The best explanation I've seen so far! :)
If anyone wants to play more with the covariance and length-scale, there I found this awesome visualisation: http://rpradeep.webhop.net/...

Reply • Share ›

kossikater • 3 years ago

Very appealing introduction, thank you!

However I got a question regarding the kernel function: Xtrain and Xtest are vectors of different length, so how can one compute the kernel? From what I understood so far, the kernel is required to be symmetric and thus a square matrix.
I noticed that the kernel function can return a non-square matrix, so it's technically possible. Can you please give some explanation on this?
Thx.

Reply • Share ›

Prashansa Gupta • 4 years ago

Hi Katherine Bailey! This post was amazing! Helped a lot! Thank you very much :)

Reply • Share ›

Arturo Lira Barria • 4 years ago

It was to nice for me to read this! It is really a good introduction, and it is simple for something that sounds very complex, thank you!

Reply • Share ›

sandipan karmakar • 4 years ago

I have a question... Why the performance of Gaussian Process based regression models is described in terms of RMSE or any other except R^2 (as in conventional Linear Regression)? I searched a lot about this but could not find any explanation...

Reply • Share ›

DH • 4 years ago

I'm not very good with statistics or math, but I've been tasked to apply a GP to a problem our lab is trying to solve (more as a pedagogical experience, since our PI could do this quite easily, I'm sure). This blog is extremely helpful. Thank you for taking the time to communicate these concepts to a more general audience.

Reply • Share ›

Wanxin Jin • 4 years ago

Thank you so much, this post is so clear and clean.

Reply • Share ›

Lay González • 4 years ago

Thanks for avoiding the rabbit holes! It is a nice introduction and probably all I need to understand for know.

Reply • Share ›

Ian Moo • 4 years ago

Good write up, but I think the estimator approach to GP's is a bit abstract because data is usually received in a spatial or temporal sequence . What Gaussian processes really describe is state estimation. In the continuous case this is the Kalman Filter and in the discrete case Hidden Markov Models

case this is the Kalman Filter and in the discrete case Hidden Markov Models. Based on what we know about the Cov of the data, and the previous point, how can we bound the next step's mean and variance? After you apply your state estimation filter to your timeseries, then you start least squares fitting a bunch of different models or just spline to find the function that underlies your model.

Not that your interpretation isn't technically sound, because it is, but I think there is a certain attachment to the physical world lost through the pure estimator intuition instead of state models.

^ | v • Reply • Share ›

Ian Moo ➚ Ian Moo • 4 years ago

Function estimator approach to GP's**

^ | v • Reply • Share ›

Zach Ward-Elms • 4 years ago

Great explanation, really helped me with my understanding of the Gaussian process! Could you tell me why you chose param = 0.1 for the kernel function and how you would go about picking the parameters in the multivariate case?

^ | v • Reply • Share ›

kris • 4 years ago

Thanks

^ | v • Reply • Share ›

yogi • 4 years ago

Great post! Your code is super helpful as well.
If anyone is looking for more details about math, but in a nutshell, following link could be a good resource.
https://www.robots.ox.ac.uk...

^ | v • Reply • Share ›

Dieter Verbeemen • 4 years ago

Hey, i've a question
What do you do when you want to set 2, 3, 4 .... hyper parameters ?

At this moment, it is very clear to find the best point for a value x and its f(x), but i would like to know, how i can use it on x, y ,z and f(x,y,z)

Can you help me with that? because most of the examples, refer back to a 1 d example

^ | v • Reply • Share ›

Isaac Medeiros • 4 years ago

Thank you! It was a big help.

^ | v • Reply • Share ›

Maaz Sarfaraz • 4 years ago

Loved it ! Thanks :D

Reply • Share ›

hoangcuong2011 • 4 years ago

I have a question though: I found many cases where kernel matrix is not positive definite and then my GPs are not valid. It happens many times even with this case: my GPs are valid with test samples with 120 data points, but my GPs are unvalid with test samples with 1200 data points.

Do you know how to fix this?

Inspired by your post, I wrote an one here :D btw.

https://github.com/hoangcuo...

Thanks a lot for your post!

Reply • Share ›

Nawaf • 4 years ago

Amazing post! Thank you very much. Please keep them coming...

Reply • Share ›

Adarsh Patodi • 4 years ago

Thankyou for detailing the topic ..

Reply • Share ›

George Miliotis • 5 years ago

Very nice introduction! Quite clear and thorough. What I am trying to understand tho, is why we need to sample from the posterior (unless it is done only for demonstration purposes) since we already know the distribution. We know the mean of the posterior, hence if we want to have a "best guess" for the test points then the mean should be quite good. And if we need confidence intervals etc, then the variance is also available. Why would someone have to draw lets say 1000 samples from the posterior and then average them to get an estimate for the test points when this (ie the estimate) is already available mathematically. Many thanks!

Reply • Share ›

Martin Stepancic ➚ George Miliotis • 4 years ago

I think your question is important when modelling dynamical systems. When modelling a nonlinear dynamical system with the model structure in form of difference equation, the dynamical system response is an iterated function and it should be calculated recursively. Let we describe a dynamical system y(k+1) = F(y(k)), where F is the Gaussian process. I have two options:
1. I could treat y(k+1) as random variable and at the next time step I would have to deal somehow with the distribution of the non-normal random variable y(k+2) and all the non-normal random variables at the subsequent time instants. This approach is known as uncertainty propagation.
2. The other possibility is the MC way: calculate a realisation of GP

first and then you may also calculate a single realisation of dynamical system response, based on the realisation of GP. In my opinion, this approach is more direct.

Reply • Share ›

hipoglucido • 5 years ago

Good job. Thanks!

Reply • Share ›