The GaussianProcess package

Anand Patil

April 3, 2007

Contents

Chapter 1

Introduction

Gaussian processes (GPs) are probability distributions for functions. They're useful if a functional form is unknown a priori.

GPs are not hard to understand at a conceptual level, but implementing them on a computer can require fairly involved linear algebra. This makes it hard for beginners to get started, and even when you have experience it's annoying.

This package provides Python classes that represent the components of the Gaussian process. They are meant to support many types of Gaussian process usage, from intuitive exploration to high-performance deployment in MCMC, with smooth transitions between.

The package also provides a class which produces Gaussian process-valued PyMC parameters and a PyMC sampling method to handle it. That means the Gaussian process-related objects can be directly incorporated into larger probability models.

You'll need to understand Python and a little bit of Bayesian statistics. Refs.

Chapter 2

Tutorial

2.1 Preliminaries

You need to know a bit about Python and numpy. Also a bit about Bayesian statistics and the normal distribution. Refs. All the code in the tutorial is in module/folder `examples`, which is distributed with this package.

2.2 A first look at Gaussian processes

Gaussian processes are probability distributions for functions. The statement 'f has a Gaussian process distribution with mean M and covariance C' is usually written as follows:

$$f \sim \mathrm{GP}(M, C). \tag{2.1}$$

Gaussian processes have two parameters, which are analogous to the parameters of the normal distribution:

- M is the mean function. Like the mean parameter of the normal distribution, M gives the central tendency for f. In Bayesian statistics, M is usually considered a prior guess for f. $M(x)$ gives the expectation of $f(x)$.
- C is the covariance function. Its role in the distribution of f is harder to understand than the mean function, but among other things it regulates:
  - the amount by which f may deviate from M
  - the smoothness of f
  - the wiggliness of f.

  $C(x,y)$ gives the covariance of $f(x)$ and $f(y)$; $C(x,x)$ gives the covariance of $f(x)$.

  Understanding covariance functions is essential for sensible application of Gaussian processes, but for the time being don't worry about them too much. Section 2.3 is all about covariance functions.

2.2.1 Instantiating a Gaussian process

It will be much easier to understand the role of the covariance function if we generate a Gaussian process to play around with.

Covariance function

The following code will produce a covariance function:

```
# example_cov.py

from GaussianProcess import *
from numpy import *

def exp_cov(x,y,pow,amp,scale):
    """
    amp and scale must be positive
    pow must be positive and less than 2
    """
    C = zeros((len(x), len(y)))
    for i in xrange(len(x)):
        for j in xrange(len(y)):
            C[i,j] = amp * exp(-(abs(x-y) / scale) ** pow)
    return C

C = Covariance(eval_fun = exp_cov, nu = .49, amp = 1., scale = .3)
C.plot(x=arange(-1.,1.,.1), y=arange(-1.,1.,.1))
```

The first argument, `eval_fun`, gives the Python function from which the covariance function will be made, in this case `exp_cov`. The arguments `pow, amp` and `scale` will be passed to `exp_cov`. See The last line displays a filled contour plot of $C(x, y)$ on the mesh ($-1 < x < 1$,$-1 < y < 1$). Try playing around with the parameters of `exp_cov` and see how C looks.

Mean function

The following code will produce a mean function M which is associated with C:

```
# example_mean.py

from GaussianProcess import *
from numpy import *

def exp_cov(x,y,pow,amp,scale):
    """
    amp and scale must be positive
    pow must be positive and less than 2
    """
    C = zeros((len(x), len(y)))
    for i in xrange(len(x)):
        for j in xrange(len(y)):
            C[i,j] = amp * exp(-(abs(x-y) / scale) ** pow)
    return C

C = Covariance(eval_fun = exp_cov, nu = .49, amp = 1., scale = .3)

def linfun(x, m, b):
    return m * x + b

M = Mean(eval_fun = linfun, C = C, m = 1., b = 0.)
M.plot(x=arange(-1.,1.,.1))
```

As with `Covariance`, the first argument to `Mean`'s init method is a Python function, in this case `linfun`. The second argument, `C`, is the `Covariance` instance with which the mean function is associated. The arguments m and b will be passed to linfun. The last line plots $M(x)$ on $-1 < x < 1$, and as expected the plot is a straight line through the origin with slope 1.

Realizations

Now let's generate some realizations (draws) from the Gaussian process defined by M and C and take a look at them.

```
# example_realizations.py

from GaussianProcess import *
from numpy import *

def exp_cov(x,y,pow,amp,scale):
    """
    amp and scale must be positive
    pow must be positive and less than 2
    """
    C = zeros((len(x), len(y)))
    for i in xrange(len(x)):
        for j in xrange(len(y)):
            C[i,j] = amp * exp(-(abs(x-y) / scale) ** pow)
    return C

C = Covariance(eval_fun = exp_cov, nu = .49, amp = 1., scale = .3)

def linfun(x, m, b):
    return m * x + b

M = Mean(eval_fun = linfun, C = C, m = 1., b = 0.)

f=[]
for i in range(3):
    f.append(Realization(M,C))

plot_envelope(M,C)
for i in range(3):
    plot(arange(-1.,1.,.1), f[i])
```

The dashdot black line in the middle is M, and the gray band is the ± 1 standard deviation envelope generated by C.

2.3 The role of the covariance function

Various visualization methods of covariance, relationship between covariance's shape and differentiability of realizations.

2.4 Nonparametric regression: observing Gaussian processes

The condition function, what it does. Some ecological examples.

2.5 The array aspect

$$\begin{aligned}
&x \sim \mathrm{N}(\mu, V) && x,\ y,\ V \text{ are scalars}\\
&\vec{x} \sim \mathrm{N}(\vec{\mu}, C) && \vec{x} \text{ and } \vec{\mu} \text{ are vectors, } C \text{ is a scalar}\\
&f \sim \mathrm{GP}(M, C) && f \text{ and } M \text{ are functions of one variable, } C \text{ is a function of two variables}
\end{aligned} \tag{2.2}$$

The base mesh

2.6 Incorporating Gaussian processes in probability models with PyMC

Chapter 3

API reference

Doxygen-style class reference here.