EAHS Gaussian Processes Project

Neale Gibson & Stephen Roberts

12-16 March 2012

TASKS

1) First check the GP module is working properly - run GPGenerateRandomVector.py, GPClassPeriodicEg_1.py and GPClassPeriodicEg_2.py. These modules contain most of the functions/class methods required for this project.

e.g. `$ ./GPClassPeriodicEg_1.py`

2) Kernel functions: Use a squared exponential covariance kernel to produce a prior distribution over functions, using a few hundred samples. Now draw random functions from the GP and plot. Vary the hyperparameters and observe the way the functions change. Hint: look at the GPGenerateRandomVector.py example.

```
#example code - first import modules
[ip] import GaussianProcesses as GP
[ip] import pylab # not always needed with pylab
[ip] import numpy as np # not always needed with pylab
#create inputs
[ip] x = np.arange(101)
[ip] X = np.array([x]).T
#define the hyperparameters for SE kernel
[ip] hp = [10.,10.,0.]]  #height scale, length scale, white noise
[ip] K = GP.CovarianceMatrix(hp,X) #create the covariance matrix
#(SE default)
[ip] pylab.plot(x,GP.RandomVector(K),'-') #create a random draw
# and plot using matplotlib
```

Now consider some (toy) data:

3) Try and understand the examples from GPClassPeriodicEg_1.py and GPClassPeriodicEg_2.py, but don't optimise the hyperparameters just yet! (i.e. leave out MyGP.MaxLikelihood() function). Fit the data using the GP with fixed hyperparameters (MyGP.predict). Vary the hyperparameters again - how does this affect the fit?

4) Bring in the optimisers to find the maximum likelihood hyperparameters: use MyGP.MaxLikelihood (default Nelder-Mead simplex algorithm)

Load create some toy periodic data. e.g.

```
x1 = np.concatenate([np.arange(100.)/10.- 5.,np.arange(100.)/10.+15.])
X = np.matrix([x1]).T
#create target data func = lambda x1:  10.*np.sin(2*np.pi*0.1*x1)
f = func(x1) + np.random.normal(0,2.,x1.shape)
f -= f.mean() #mean centre the data
```

5) Try and think about how to fit periodic datasets using GPs, and write your own kernel (Look at the source code to get the function inputs/outputs correct). Hint: modify the SE covariance kernel to include periodicity. Make random draws as before varying the hyperparameters, and fit some toy periodic data using the kernel, and optimise the hyperparameters.

Now load in the real data: HD189733_flux.dat

```
[ip] time,flux = np.loadtxt('HD189733_flux.dat',unpack=1)
[ip] pylab.plot(time,flux,'.')
```

7) This is long-baseline photometric observations of the exoplanet host star HD 189733. The quasi-periodic variations are caused by star spots rotating into view and evolving over time. How do we create a kernel for quasi-preiodic data? Apply all that you know and find:

a) Most likely period of flux variation (stellar rotation period)
b) Amplitude of variation
c) Evolutionary scale over which periodicity itself varies.

We have only explored the basics of GPs. In general we may want to incorporate mean functions (e.g. transit mean functions) and marginalise over the hyperparameters of the mean function and kernel. We also have python code for this! Think about how you might incorporate GPs into your research.

BREAKDOWN OF CODE

(see the module source code for more detailed descriptions)

```
#import the GaussianProcess module, now called as GP
import GaussianProcesses as GP

#create the data set (ie training data)
#the input vectors must be given as an N x D matrix
#(in our case D will always equal 1)
X = np.matrix([x1]).T #input matrix (N x D)

#create the target variables
#often we mean subtract the data
f = np.array([-1.6,-1.3,-0.40, 0.10, 0.5 ,0.7])
f -= f.mean() #mean centre the data

#create the covariance matrix and plot draw from prior
hp = [10.,10.,0.]  #hyperparameters for sq exponential
K = GP.CovarianceMatrix(hp,X) #generate covariance matrix
pylab.plot(x,GP.RandomVector(K),'-')

#create the predictive x values
#these are the points used to plot the function/ranges etc
#should be in the same format as X
x1 pred = np.arange(-5.0,5.0,0.1)
X_pred = np.matrix([x1_pred]).T #convert to data matrix (q x D)

#the following functions define the GP class
#create the GP class, the training data t, inputs X etc can be defined here
MyGP = GP.GP(t=f,X=X,kernel='SqExp')
#or alternatively the parameters can be reset at any time using
#the set attributes method
MyGP.set attributes(hyperparams=[1.,1.,0.3])
#the describe method prints out the current GP parameters/functions
MyGP.describe()
#find maximum likelihood (hyper)parameters - default is NM simplex method
MyGP.MaxLikelihood()

#get predictive values and uncertainties from GP regression
#this only returns the diagonal of the covariance matrix as the error
f_pred, f_pred_err = MyGP.predict(X_pred=X_pred,WhiteNoise=True)

#handy functions to plot the data and regression
GP.PlotData(x1,f,MyGP.hyperparams[-1])
GP.PlotRanges(x1_pred,f_pred,f_pred_err,"Gaussian Process Regression
Test")
```

#draw a random vector from the posterior of the GP, ie conditioned on the data/hyperparameters
```
draw_vector = MyGP.GetRandomVector()
pylab.plot(x1_pred,MyGP.GetRandomVector())

#wait for input from the terminal
raw_input()
```