Lab session 2: Sparse GPs and likelihood approximation

GP Summer School – Kampala, 6-9th of June 2013

The aim of this lab is to explore the sparse GP formulation developed in the lectures. We'll also play with a classification problem using Expectation Propagation (EP).

1 Getting started

Recall the simple set up we saw before:

```
import numpy as np
import pylab as pb
import GPy

pb.ion()
```

Create a simple data set by sampling from a GP:

```
X = np.sort(np.random.rand(50,1)*12)
k = GPy.kern.rbf(1)
K = k.K(X)
K+= np.eye(50)*0.01 # add some independence (noise) to K
y = np.random.multivariate_normal(np.zeros(50), K).reshape(50,1)
```

Build a straightforward GP model of our simulation. We'll also plot the posterior of **f**:

```
m = GPy.models.GPRegression(X,y)
m.ensure_default_constraints()
m.optimize()
m.plot()
mu, var = m._raw_predict(X) # this fetches the posterior of f
pb.vlines(X[:,0], (mu-2*np.sqrt(var))[:,0], (mu+2*np.sqrt(var))[:,0],"r")
```

Would you describe the information in the posterior as redundant? What happens to the model if you remove some data? Try:

```
GPy.models.GPRegression(np.delete(X,[0,1,2],0), np.delete(y,[0,1,2],0))
```

2 Build a sparse GP model

Now we'll build a sparse GP model:

```
Z = np.random.rand(3,1)*12
m = GPy.models.SparseGPRegression(X,y,Z=Z)
m.ensure_default_constraints()
print m
```

In GPy, the sparse inputs **Z** are abbreviated `"iip"`, for *inducing input*. Plot the posterior of **u** in the same manner as for the full GP:

```
mu, var = m._raw_predict(Z)
pb.vlines(Z[:,0], (mu-2*np.sqrt(var))[:,0], (mu+2*np.sqrt(var))[:,0],"r")
```

Question 1a Optimise and plot the model. The inducing inputs are marked – how are they placed? You can moved them around with e.g. `m["iip_2_0"] = 100`. What happens to the likelihood? What happens to the fit if you remove an input?

Question 1b How does the fit of the sparse compare with the full GP? Play around with the number of inducing inputs, the fit should improve as M increases. How many inducing points are needed? What do you think happens in higher dimensions?

3 Classification

We'll construct a toy data set to play with:

```
X = np.random.rand(100,2)
y = np.where(X[:,1:]>0.5*(np.sin(X[:,:1]*4*np.pi)+0.5),1,0)
X[:,1:] += y*0.1  # make the boundary well behaved
m = GPy.models.GPClassification(X,y)
m.randomize()
m.update_likelihood_approximation()
m.plot()
```

The decision boundary should be quite poor!

To optimize the model, we'll alternate between optimizing the hyper-parameters:

```
m.ensure_default_constraints()
m.optimize("bfgs") #your preferred optimizer here
```

and re-approximating the likelihood using EP:

```
m.update_likelihood_approximation()
```

Question 2 Write a for loop to optimize the model by iterating between EP and optimisation of the parameters. You may have to restart the EP algorithm occasionally (`m.likelihood.restart()`)

4 Sparse GP Classification

Here we'll build a super-simple application to classify images. The two class labels correspond to whether the subject of the image is wearing glasses.

Set up the ipython environment and download the data:

```
import urllib
from scipy import io

urllib.urlretrieve("http://www.cs.nyu.edu/~roweis/data/\
    olivettifaces.mat", "faces.mat")
face_data = io.loadmat("faces.mat")
```

Here's a simple way to visualise the data. Each pixel in the image will become an input to the GP:

```
faces = face_data["faces"].T
pb.imshow(faces[120].reshape(64,64,order="F"), interpolation="nearest",
    cmap=pb.cm.gray)
```

Now fetch the class labels:

```
urllib.urlretrieve("http://staffwww.dcs.sheffield.ac.uk/people/\
    J.Hensman/gpsummer/datasets/has_glasses.np", "has_glasses.np")

y = np.load("has_glasses.np")
y = np.where(y=="y",1,0).reshape(-1,1)
```

Divide the data into a training/testing set:

```
index = np.random.permutation(faces.shape[0])
num_training = 200
Xtrain = faces[index[:num_training],:]
Xtest = faces[index[num_training:],:]
ytrain = y[index[:num_training],:]
ytest = y[index[num_training:]]
```

Choose some inducing inputs. Is this a good scheme for choosing them? Can you devise a better one?

```
from scipy import cluster
M = 8
Z, distortion = cluster.vq.kmeans(Xtrain,M)
```

Finally, we're ready to build the classifier object.

```
k = GPy.kern.rbf(4096,lengthscale=50) + GPy.kern.white(4096,0.001)
m = GPy.models.SparseGPClassification(Xtrain, ytrain, kernel=k, Z=Z,
  normalize_X=True)
m.update_likelihood_approximation()
m.ensure_default_constraints()
```

Question 3a Look at the following figure. What is being shown? Why does it look like this?

```
pb.figure()
pb.imshow(m.dL_dZ()[0].reshape(64,64,order="F"),interpolation="nearest",
    cmap=pb.cm.gray)
```

Question 3b Write some code to evaluate the model's performance, using the held-out data that we separated earlier. How is the error rate? Is that better than random guessing?

Question 3c Write another simple for loop to optimize the model. How low can you get the error rate to go? Hint: this author can get less than 10% errors.
How could you make the model better? What kind of kernel do you think might be appropriate for this classification task?