COM6509/4509—Assignment Sheet 1: Regression in Python

Neil Lawrence — October 15th, 2013

Remember to check what a command does, simply type:

```
np.random.randn?
```

You should store your answers to the questions in an ipython notebook file which will be submitted at the assignment deadline. Any answer where you are required to write something down should be done using a markdown field in the notebook file. The answers will be submitted before the start of the lab class in Week 7 via MOLE. **Late submissions will get zero** as we will mark the submissions directly after they are received in the lab class. We will try to run your notebook by loading it and selecting 'Run All' from the menu.
Make sure it answers each of the questions below!
At the beginning of your script you'll want to set it up so that the plots appear *inline*

```
%pylab inline
```

which will also import the `numpy` library as `np` and the pylab plotting libraries as `plt`.

Assignment

Olympic Data The marks allocated here will make up 20% of your mark for the entire course. Marks are given beside each question. We are going to load in some real the Olympic marathon data we have seen in the lecture and lab classes. To load the data we first need to download it from the web. To do this we can use `urllib`.

```
import urllib
```

which allows us to download the data from `http://staffwww.dcs.shef.ac.uk/people/N.Lawrence/dataset_mirror/olympic_marathon_men/olympicsMarathonTimes.csv`.

```
url = ("http://staffwww.dcs.shef.ac.uk/"
    + "people/N.Lawrence/dataset_mirror/"
    + "olympic_marathon_men/olympicMarathonTimes.csv")
urllib.urlretrieve(url, 'olympicMarathonTimes.csv')
olympics = np.loadtxt('olympicMarathonTimes.csv', delimiter=',')
```

This loads the data into a an array. We can extract the relevant portions using

```
x = olympics[:, 0:1]
```

to extract the Olympic years and

```
y = olympics[:, 1:2]
```

to extract the pace of the winning runner. We can plot these as before

```
plt.plot(x, y, 'rx')
```

1. **Linear Model**

   (a) Fit a linear model to the data as above. Plot the fit. Compute the final error. *5 marks*

   (b) Now we will fit a non-linear basis function model. Start by creating a quadratic basis

   ```
   Phi = np.hstack((np.ones_like(x), x, x**2))
   ```

   and plot the fit, again computing the final error. *5 marks*

   (c) Large order polynomial fits tend to do fairly extraordinary things outside an input range of -1 to 1. Have a think about why this is, in particular, what is the result of 2012^8? Do you think that is a suitable basis? What could be wrong with it? *5 marks*

2. **Radial Basis Functions** We turn to a fit based on exponentiated quadratic (or Gaussian or radial) basis functions:

$$\phi(x) = \exp\left(-\frac{(x-\mu)^2}{2\ell^2}\right)$$

   For each model, you can place the basis functions uniformly across the input space. So we have:

   ```
   mu_vector = np.linspace(1896,2012,num_basis)
   ```

   as a one dimensional array containing the centres of the basis, where `num_basis` is the number of basis functions. You can construct the basis matrix using

   ```
   # Allocate the matrix for the basis set
   Phi = np.zeros((x.shape[0], num_basis))
   # Set the width of the basis functions
   width = 2*(x.max() - x.min())/num_basis
   # precompute the scale of the exponent
   scale = -1/(2*width*width)
   for i in range(0, num_basis):
       Phi[:, i] = np.exp(scale*(x.flatten()-mu_vector[i])**2)
   ```

   Note here that we are setting the width of the basis functions according to the distance between the basis functions.

(a) Plot the basis function values for the inputs of years between 1896 and 2012. On the x-axis should be years, on the y-axis the values of the basis functions. Create the plot for 2, 5 and 10 basis functions. Make sure the basis functions are nicely overlapping (i.e. that you've set the width correctly) and that they are spread out across the input range. *5 marks*

Hints: for plotting the function given by your prediction you can use code like this.

```
# Make predictions every year from 1896 until 2012
x_pred = np.arange(1896,2012,1)
# Allocate the basis function matrix
Phi_pred = np.zeros((x_pred.shape[0], num_basis))
for i in range(0, num_basis):
    Phi_pred[:, i] = np.exp(scale*(x_pred.flatten()-mu_vector[i])**2)
```

Plot the basis set along the input line using

```
plt.plot(x_pred, Phi_pred)
```

(b) Now let's observe what type of functions these basis sets can generate by sampling a weight vector, $\mathbf{w}$, from a Gaussian density and using it to create a function.

```
# Sample a candidate weight vector from a Gaussian.
w = np.random.randn(num_basis, 1)
f = np.dot(Phi_pred, w)
plt.plot(x_pred, f)
```

Do this a few different times, obtaining a different sample from the Gaussian for $\mathbf{w}$ each time. Do this for models containing 2, 5 and 10 basis functions. *5 marks*

(c) Fit basis function models with between 1 and 27 basis functions (for each number of basis functions have your code create the plot, show the error as the title `plt.title('errorValue')`, pause until a key is pressed and then create the next plot. Store the error values and plot them in a separate plot against the number of basis functions. *10 marks*

Hints: Don't forget to fit the noise variance!!

Once you have the estimate of $\mathbf{w}$ (`w_star`), you can make the prediction using a matrix multiplication.

```
y_pred = np.dot(Phi_pred, w_star)
```

(d) **Data Fit** Which number of basis functions has the lowest error? Do you think this is a good fit? *1 mark*

(e) **Validation Data** Let's assume that we only have the data up until 1980. Hold-out the data from 1984 onwards. Now retrain your models and test the result on this held out data. What is the difference in your results? Which number of basis functions now has the best error? *10 marks*

(f) **Cross Validation** For each number of basis compute and plot:

i. the leave-one-out cross validation error. *10 marks*

ii. the 5 fold cross validation error. *10 marks*

Which model should be chosen according to each of these model selection criterion?

Note: The data doesn't divide into 5 equal size partitions for the five fold cross validation error. Don't worry about this too much. Two of the partitions will have an extra data point. You might find `np.random.randperm?` useful.

3. **Bayesian Solution** In this question you will perform Bayesian inference on the data. Use the same basis sets from the previous question. Select a Gaussian prior for the mapping vector, $\mathbf{w}$,

$$p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha\mathbf{I}\right),$$

where α is the prior variance. Assume a standard deviation of $\sigma = 0.3$ for the noise (i.e. a variance of $\sigma^2 = 0.09$) and a variance of $\alpha = 100$ for the prior variance on $\mathbf{w}$.

(a) **Marginal Likelihood** Now compute and plot the log *marginal likelihood*, $\log p(\mathbf{Y})$, for each of the models

$$\log p(\mathbf{Y}) = -\frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{K}| - \frac{1}{2}\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y},$$

where

$$\mathbf{K} = \alpha\mathbf{\Phi}\mathbf{\Phi}^\top + \mathbf{I}\sigma^2.$$

Find the model with the best fit according to the marginal likelihood. Does it match that chosen by cross-validation or validation? *10 marks*

(b) **Posterior Mean Prediction and Variance** For the selected model compute the mean and covariance of the posterior density for $\mathbf{w}$.

$$\left\langle \mathbf{w}\mathbf{w}^\top \right\rangle = \mathbf{C}_w + \left\langle \mathbf{w} \right\rangle \left\langle \mathbf{w} \right\rangle^\top$$

$$\mathbf{C}_w = \left[\frac{1}{\sigma^2}\mathbf{\Phi}^\top\mathbf{\Phi} + \alpha^{-1}\mathbf{I}\right]^{-1}$$

$$\left\langle \mathbf{w} \right\rangle = \mathbf{C}_w\sigma^{-2}\mathbf{\Phi}^\top\mathbf{y}$$

Plot the mean fit to the data and one standard deviation above and below. For this plot extend the input range of your plot to be between 1850 and 2050. *10 marks*

(c) **Posterior Samples** For the same model take 10 samples from this posterior density to plot 10 fits to the data on top of the data. Plot all the samples in

the same plot (continue to use the 200 year input range from 1850 to 2050). Check `np.random.multivariate_normal?` for sampling from a multivariate Gaussian. *5 marks*

(d) **Quality of Error Bars** Comment on the quality of the error bars. Are the variances for the predictions reasonable across the input range? *9 marks*

Useful commands: `np.linalg.det?`, `np.pi?`