# ML4/M Bayesian Regression – Olympic example

Simon Rogers

17th February 2016

## Aims

To implement Bayesian inference over the parameters of the linear model for the Olympic data and plot the predicted mean and variance for different polynomial orders.

## Tasks

- Download the 100m data (again)
- For some polynomial order $k$ (I suggest you start with the linear model, but write your code such that it's straightforward to change this to quadratic, cubic, etc), define the mean and covariance of a Gaussian prior over $\mathbf{w}$. I.e. create a mean vector (same size as $\mathbf{w}$) that is all zeros and a covariance matrix that has zeros everywhere except on the diagonal, (`np.zeros((k,1))` and `100.0*np.identity(k+1)` will help here, if `k` is the polyonial order) where the values are 100. Have a think about what kind of models are likely from the prior. If you're feeling adventurous, sample some $\mathbf{w}$ values from this prior (see below) and plot the models.
- Create the $\mathbf{X}$ object (column of 1s, column of $x$, etc)
- Compute the mean and covariance of the posterior over $\mathbf{w}$ (expressions below). If you're feeling adventurous, you could derive this in the same manner we derived it for the one-dimensional case in the lecture. If you're doing this, I suggest you write the likelihood as an $N$-dimensional Gaussian with mean $\mathbf{Xw}$ and covariance $\sigma^2\mathbf{I}$.
- Define a test set over the range of Olympic years of interest. You will use this for plotting. Make the $\mathbf{X}$ for the test data. Compute the mean (expression below) and the variance (ditto) of the predictive Gaussian at the test points. The errorbar function is a good way of doing this. Does it look right?

# Some useful things

## Sampling from a multivariate Gaussian

If you want to visualise samples from the prior or posterior, `numpy` gives you a function to sample from multivariate Gaussians. It is `numpy.random.multivariate_normal(mean,covariance)`. Note that the mean should be a one-dimensional object. You might find yours is 2-dimensional. If that's the case, use the `.flatten()` method to get rid of the extra dimension when you call the random function.

## Posterior mean and covariance

If the prior mean is zero and the prior covariance is denoted as $\mathbf{\Sigma}_0$ then the posterior covariance is given by:

$$\mathbf{\Sigma} = \left(\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X} + \Sigma_0^{-1}\right)^{-1}$$

and the mean by

$$\boldsymbol{\mu} = \frac{1}{\sigma^2}\mathbf{\Sigma}\mathbf{X}^T\mathbf{t}$$

Set $\sigma^2$ to 0.05.

## Making predictions

The predictive mean is given by $\mathbf{X}_{test}\boldsymbol{\mu}$ and the predictive variance for a single test point by $\sigma^2 + \mathbf{x}_*^T\mathbf{\Sigma}\mathbf{x}_*$

## Plotting errorbars

If you have the predictive mean for all of your test points in a vector `pred_mean` and the variances in `pred_var` then the following will plot you errorbars (assming `pylab` or `matplotlib` are imported as `plt`:

```
plt.errorbar(testx,pred_mean,pred_var)
```

If you do this for a 2nd order polyomial and a fine grid of test points between -10 and 50 (assuming you have re-scaled the $x$ values by subtracting 1896 and dividing by 4) you should get something that looks like the figure below. Play around with the prior covariance values and $\sigma^2$ to see what happens.

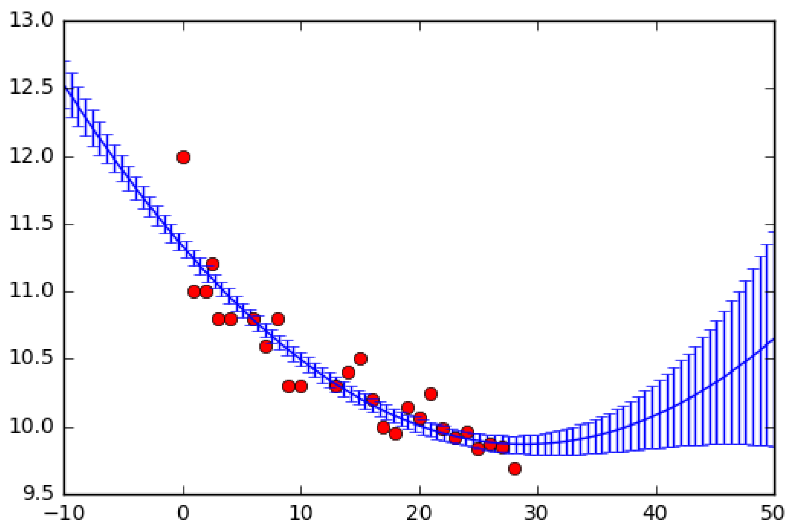


Figure 1: