Machine Learning 4/M. Bayesian regression lab 2

Simon Rogers

14th Feb 2017

Aims

To get more experience in the use of Bayesian models for regression, in particular predictions.

Tasks

Generating some data

- We will now move away from the Olympic data and generate some data from a polynomial function. Generate some x values, evenly spaced between -1 and 1. It doesn't matter exactly how many, but at least 10 would be sensible. `np.linspace` is a good function for this.
- Create an $\mathbf{X}$ matrix, up to a power of your choosing (squared or cubic are good bets)
- Choose some *true* w values. If you put your values in a vector, experiment by plotting x against $\mathbf{Xw}$. You want a function that isn't too boring and simple. A good suggestion is:

```
true_w = np.array([[1.0,2.0,3.0]]).T
```

- Generate some training t values by computing $\mathbf{Xw}$ and then adding some Gaussian noise. Let `sig_sq` be your chosen noise variance:

```
t = np.dot(X,true_w) + np.random.normal(scale=np.sqrt(sig_sq),size
= x.shape)
```

Defining the priors

- You will need a prior mean vector (same shape as `true_w`) and a prior covariance matrix. The following are sensible (where `l=len(true_w)`), but experiment:

```
prior_mean = np.zeros((1,1))
```

```
prior_cov = 10*np.eye(1)
```

- Sample 10 functions from the prior and plot along with the data. To sample:

```
w_samples = np.random.multivariate_normal(prior_mean.flatten(),prior_cov,10)
```

Posterior inference

- Using the expressions for the posterior mean and covariance, compute the mean and covariance matrix.

Generate 10 posterior samples and plot them

- Sampling code as given above

Making predictions

We will now look at the relationship between sampling from the posterior and making predictions using the analytical formula. In the lectures we saw how we could compute the exact predictive distribution by performing an expectation (i.e. averaging over the posterior of w). We can always approximate an expectation by drawing samples and averaging the samples they give:

- Create an x vector for testing. Use `x = 1.5` (you'll need to make a vector with the different powers in it).
- Generate 1000 samples from the posterior, and for each one compute the predicted value. If the sth sample is $\mathbf{w}_s$ then the prediction is $\mathbf{w}_s^T\mathbf{x}$. Store all 1000 samples.
- Plot a histogram of the samples.
- Now compute the exact predictive distribution. It is a Gaussian with mean and variance given by:

$$a = \boldsymbol{\mu}^T\mathbf{x}$$

$$b^2 = \sigma^2 + \mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the posterior mean and covariance respectively.

- Plot the pdf of this Gaussian and compare with the histogram. They should have near identical shape (although the histogram represents counts so will have a larger scale on the y-axis). In fact, the histogram is a

sample-based approximation of the exact expectation used to comptue the exact predictive Gaussian. `Scipy` has a normal pdf function.