Becoming More Bayesian: Documentation

Kim A. Sharp, Ph.D.
Github repository: kimandsharp/bmb510

Overview

This code was written for BMB510 “Data Analysis and Scientific Inference” a class taught to first year Ph.D students in the Biochemistry and Molecular Biophysics Graduate Group at the University of Pennsylvania. The aims were: to provide a better alternative to the usual null-hypothesis/significance testing way of teaching stats; introduce students to Bayesian statistics and the Bayesian way of thinking about data analysis; act as a prequel to more computationally advanced Bayesian statistics as described by A. Gelman et al. "Bayesian Data Analysis", 3rd ed., and John Kruschke "Doing Bayesian Data Analysis", as implemented in PyMC3.

The code implements Bayesian equivalents of most basic analyses used in conventional statistics. However, the emphasis is on estimation of parameters and their credible intervals, not hypothesis testing. Parameters include the mean and standard deviation of populations, the difference in means between two populations (cf. T-tests), Poisson processes (rate-type parameters), Binomial data (fraction or proportion type parameters), linear regression parameters (correlation, slope and intercept), population size, rank tests, as well as Bayesian versions of some more advanced analyses: The Weibull distribution (survival curve) parameters, periodic data, and polynomial (non-linear) curve fitting parameters.

The code is a set of standalone Python programs, each designed to do one thing well. Each is easy to run and to input data. So learning to use the code is largely a matter of identifying the class of data you have, or the parameter(s) you want to estimate and picking the corresponding program.

The Code

Dependencies

Requires Python3, the math, matplotlib, random and numpy packages. The supplied module SciInf_utilities.py contains global parameters and utility defs required by the programs.

Computation

Since the number of model parameters is small, 'exact' numerical integration can be used for marginalization integrals when required and to obtain the posterior cumulative probability distribution function (cdf) from the posterior probability density function (pdf). The default of 2501 points used for numerical integration has been found to be more than accurate enough. If necessary this can be changed by assigning a different value to the global parameter NPOINT in the file SciInf_utilities.py.

Input

Type the name of a program at the command line with no arguments and it will prompt you for input file name(s) and parameters as required. Otherwise, for most programs these can be supplied as arguments on the command line. The csh script file run_tests.com shows how to run every program, as well as running the code tests.

Input file formats are very simple: lines starting with # are treated as comments and may appear anywhere in the file. Anything to the right of a # on a data line is ignored. The usual format is one data element per line – i.e. for singleton data, one float or integer per line. For doublet type data (e.g. x,y pairs for linear regression), one x,y pair per line, separated by one or more spaces/tabs, for triplets similarly. There are plenty of example input files in the testdata directory. The csh script file run_tests.com shows which test data files are for which programs.

Output

The programs report median values of parameters and 95% credible intervals. The credible interval range is set in the module SciInf_utilities.py by the global percentile variables CREDIBLE_MIN and CREDIBLE_MAX and so it can easily be changed by the user. Mean and modal (maximum likelihood) parameter values are also reported for completeness; However, these are less preferred than the median (50th percentile value), since the latter is consistent with the percentile based credible interval.

Most programs also provide plots of the posterior pdf and cdf for each estimated parameter. If the cursor is placed within the plot area Matplotlib provides an interactive readout of the x and y values of the cursor in the plot units. This renders the cumulative probability (cdf) particularly useful in various ways: * The parameter value corresponding to any desired percentile is obtained by putting the cursor at the desired percentile, moving it horizontally to meet the cdf curve and reading off the x-coordinate of the cursor. * To obtain say the 95% credible interval, use the previous procedure to obtain the parameter values corresponding to the 2.5% and 97.5% percentiles. * To obtain the probability that the parameter has a given value of *x* or *less* place the cursor at the *x* value of the parameter, move it vertically to meet the cdf and read off the percentile. * To obtain the probability that the parameter lies within the range x_1 and x_2 repeat the previous procedure for both values of *x* and subtract the probabilities.

For three programs in particular, the graphical output provides additional information.

1) PeriodicSeries.py provides a plot of the posterior probability vs. period. While the program attempts to find the most probable period/frequency and prints this value, the plot may reveal other prominent peaks that support a different period or multiple periods.

2) SurvivalWeibull.py. This program models survival data using the Weibull distribution, printing the shape and scale (half-life) parameters with their credible intervals. The program also plots the survival curve (in Kaplan-Meier form) with error ranges calculated using the binomial variance formula, the corresponding Weibull curves calculated with the upper and lower credible interval values for the shape and scale parameters, and the hazard as a function of time.

3) CurveFitBIC.py. The program fits a set of (x,y) data to an Nth order polynomial, sequentially fitting from 0th order (a constant) to 6th order, using the Bayesian information criterion to down-weight increased number of parameters. The fit line and the residuals are plotted for each order polynomial, and the program reports the order (model) with the highest log probability. However, the user might choose a different order polynomial based on examination of the pattern of residuals.

Many of the programs output a file containing the posterior pdf and cdf for each parameter. These files have the extension *_pdf_cdf.dat and are suitable for plotting. The format is one set of (x_i, pdf(x_i), cdf(x_i)) per line, where x_i is the i'th value of the parameter. Two of these *_pdf_cdf.dat files can also be used as input to the program DiffPdf.py. See the section on posterior analysis below.

Testing the code

Run the csh script run_tests.com from the directory containing src and testdata, redirecting the output to a file. You might want to change the global variable MAKEPLOT in SciInf_utilities.py to False first, so you don’t have to keep closing the graphic windows (then change it back to True when done!). Compare the output to the run_tests_<date>.out file provided.

Posterior Analysis

If we have the *_pdf_cdf.dat files for some parameter from two different conditions, or from two different samples or populations, e.g. control/experiment or background/test, the posterior pdf and cdf of the *difference* in that parameter between those two different cases can easily be computed by marginalization using the utility programs DiffPdf.py or CombineTwoPdfs.py. Obviously, to be meaningful the two *_pdf_cdf.dat files used by these programs must contain the distributions of the same parameter. CombineTwoPdfs.py can also multiply two pdfs. This enables one to multiply say a posterior pdf obtained with a uniform prior by some alternative prior