RATE Code Validation

Jonathan Ish-Horowicz

October 2019

1 Aim

The aim is to validate the KLD calculations in the Python code against the R script.

2 Description of the Two Code Versions

There are two versions of the code:

1. The R script that generated the results in the Annals of Applied Statistics paper. This has been V1 in Dana's tests and I have wrapped it using rpy2 so it can be called from Python.
2. The Python package that I wrote originally for the NeurIPS submission and was then used for ICML. This code had a bug for both of these submissions that has now been fixed. This has been V3 in Dana's tests.

Note: The version of the Python code that was used to generate the results for NeurIPS and ICML (V2 in Dana's tests) is known to be buggy and so hasn't been tested here. The aim here is to ensure that the Python code we will use moving forward is correct by checking it against the original R code.

The differences between the R and Python codes are summarised in Table 1.

Table 1: Summary of the differences between the two versions of the RATE code.

	R	Python
Effect Size Analogue Projections	Pseudoinverse originally, with Covariance added for Python-wrapped version	Pseudoinverse and Covariance
Matrix Factorisation (Pseudoinverse only)	Always originally, can be switched off in Python-wrapped version	No
Posterior Evaluation Method	Sampling	Closed-form
KLD calculation	approximate	approximate or exact

The approximate KLD calculation is

$$\mathrm{KLD}(\widetilde{\beta}_j) \approx \frac{1}{2}\boldsymbol{\lambda}_{-j}^T \boldsymbol{\Lambda}_{-j}^{-1} \boldsymbol{\lambda}_{-j} \mu_j^2 \,,$$

while the exact one is

$$\mathrm{KLD}(\widetilde{\beta}_j) = \frac{1}{2}\left[\mathrm{trace}(\boldsymbol{\Omega}_{-j}\boldsymbol{\Lambda}_{-j}) - \log|\boldsymbol{\Omega}_{-j}\boldsymbol{\Lambda}_{-j}| - (p-1) + \boldsymbol{\lambda}_{-j}^T \boldsymbol{\Lambda}_{-j}^{-1} \boldsymbol{\lambda}_{-j} \mu_j^2\right].$$

3 Test Workflow

Since we are only testing that the results from each version of the code matches, there is no need for a model - we simply generate a mean vector and a positive semi-definite covariance for the logit posterior. I assume we have the posterior distribution of the logits $p(\mathbf{f} \mid \mathbf{X}, \mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is full rank (i.e. is of rank n for n data points).

For the unit tests I used $n = 100$ data points and $p = 10$ variables.

1. Generate $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, the parameters of the logit posterior $p(\mathbf{f} \mid \mathbf{X}, \mathbf{y})$. Also generate $\mathbf{X}$, the design matrix.
2. Compute RATE using the Python code (using the approximate KLD, to match the R code).
3. For `n_draws` in $[10^2, 3\times10^2, 10^3, 3\times10^3, 10^4, 3\times10^4, 10^5, 3\times10^5, 10^6]$, repeat the following two steps 10 times.
   (a) Compute RATE using the R code with `n_draws` draws from the posterior.
   (b) Calculate (1) the L2-norm, (2) the Pearson and (3) the Spearman correlation between the Python and R KLD result.

The R script uses sampling and the Python uses closed-form, but we can check the convergence of the R KLD result to the Python KLD result as the number of posterior samples increases. If the Python code is correct, the L2-norm between the RATE values according to the Python and R codes will tend to zero as the number of posterior draws increases. The correlation measures will tend to one.

4 Test Results

4.1 Testing using the pseudoinverse projection

The R code originally only contained the pseudoinverse projection, so we start by validating against that. If we turn off the matrix factorisation (and invert $\boldsymbol{\Sigma}$ directly) the result should be the same as the Python code.

Figure 1 illustrates the $L2$ norm, as well as correlations, between the RATE values produced by R and Python – when no matrix factorisation is used – as the number of posterior samples increase. If we do the matrix factorisation, the result is the very similar (Figure 2). This is to be expected in this case as $\boldsymbol{\Sigma}$ is full-rank. Note that, as hoped, the RATE values produced by the R and Python codes are very similar if the R code use $!0^6$ samples from the posterior. They differ for smaller sample sizes.

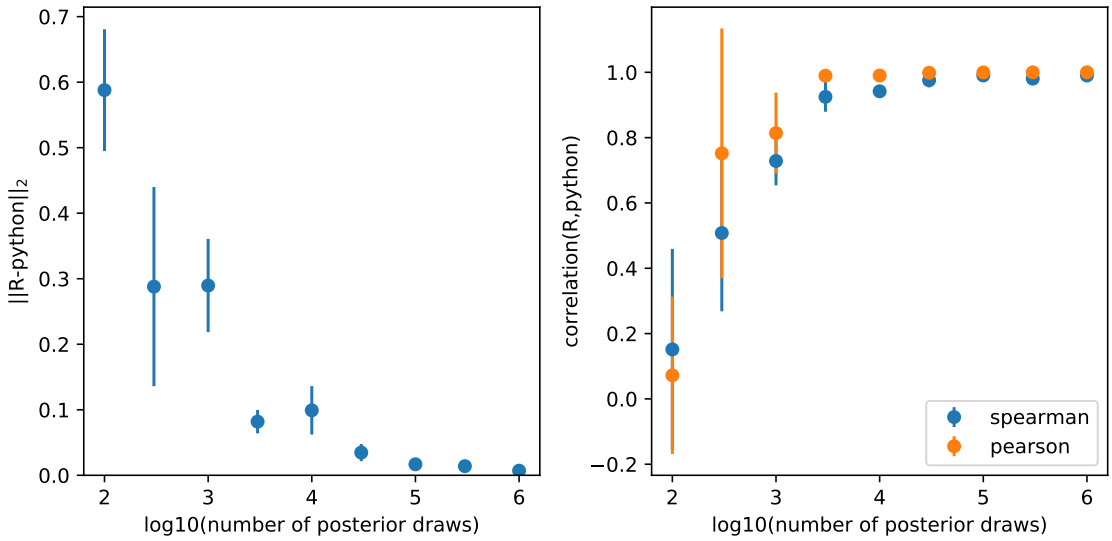

Figure 1: Comparing the RATE values according to the Python and R code for different number of posterior draws. Pseudoinverse effect side analogue, without the matrix factorisations. The RATE values using the R code were computed 10 times for a given number of posterior draws - each data point is the mean ± the standard deviation of a value over those 10 sets of draws.

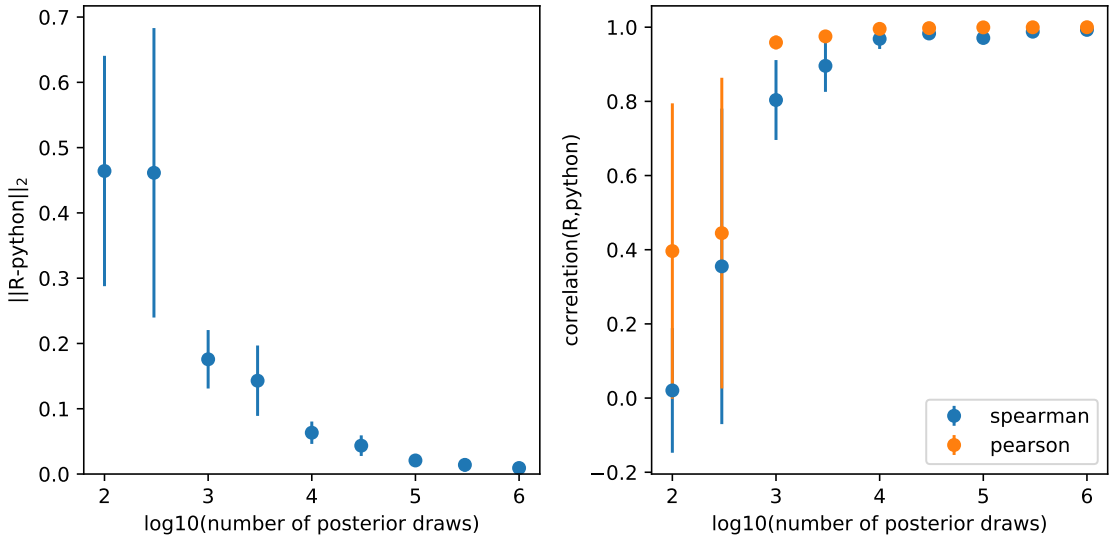

Figure 2: Comparing the RATE values according to the Python and R code for different number of posterior draws. Pseudoinverse effect side analogue, with the matrix factorisations. The RATE values using the R code were computed 10 times for a given number of posterior draws - each data point is the mean ± the standard deviation of a value over those 10 sets of draws.

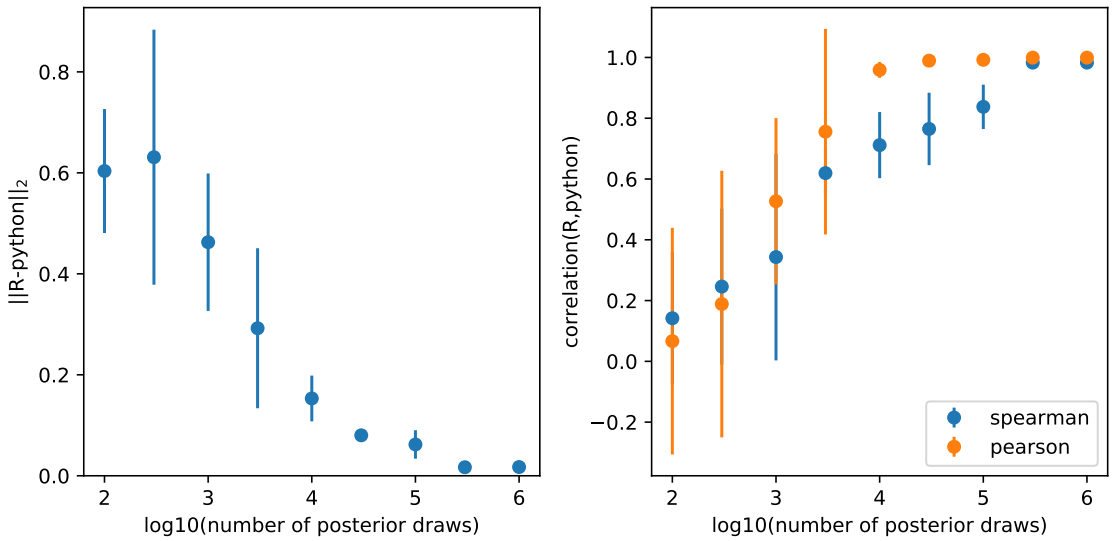

Figure 3: Comparing the RATE values according to the Python and R code. Covariance projection. The KLD values according to R tend to the Python result as the number of posterior draws increases. The R result was computed 10 times for a given number of posterior draws - each data point is the mean ± the standard deviation of a value over those 10 sets of draws.

4.2 Testing the covariance projection

We have added a covariance projection (effect size analogue) to the R code to test against the Python version. The covariance projection was not originally in the R code as we only introduced it in the NeurIPS submission. Figure 3 illustrates the $L2$ norm, as well as correlations, between the RATE values produced by R and Python as the number of posterior samples increase.