Gaussian Process Models for Bayesian Regression and Classification

Eugenio Bonifazi

Contents

1 Introduction

1.1 Regression and Classification with Gaussian Process

This work is a study and implementation of Regression and Classification models based on the Gaussian Process, in particular following the perpective described in *"Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification"* (Neal 1997). Gaussian processes are a natural way of specifying prior distributions over functions of one or more input variables. In the first part I basically describe the Regression and Classification problem focusing then on the covariance function and, in the last part an implementation of both regression and classification using R and RStan occurs.

Assuming we have n observations, $(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), ..., (x^{(n)}, t^{(n)})$, where $x^{(i)} = x_1^{(i)}, .., x_p^{(i)}$ is the vector of predictors ("inputs") related to input i and $t^{(i)}$ is the associated target (response), a simple linear regression model can be written as:

$$t^{(i)} = \alpha + \sum_{u=1}^{p} x_u^{(i)} \beta_u + \epsilon^{(i)}$$

where $\epsilon^{(i)}$ is the Gaussian "noise" of the i^{th} observation, assumed to be independent from the other cases and normally distributed, with mean zero and variance σ_ϵ^2. We will assume that σ_ϵ^2 is known while α and β_u are unknown. We give to both parameters independent Gaussian priors with mean zero and variance σ_α^2 and σ_u^2. These priors over the parameters imply a prior distribution for the associated target values $t^{(1)}, t^{(2)}, ...$ that is a multivariate Normal with mean zero and and covariances given by:

$$Cov[t^{(i)}, t^{(j)}] = E\left[\left(\alpha + \sum_{u=1}^{p} x_u^{(i)} \beta_u + \epsilon^{(i)}\right)\left(\alpha + \sum_{u=1}^{p} x_u^{(j)} \beta_u + \epsilon^{(j)}\right)\right] = \sigma_\alpha^2 + \sum_{u=1}^{p} x_u^{(i)} x_u^{(j)} \sigma_u^2 + \delta_{ij}\sigma_\epsilon^2 \qquad (1)$$

where $\delta_{ij} = 1$ if $i = j$ and zero otherwise. This mean and covariance function are sufficient to define a "Gaussian process" giving a distribution over possible relationships between the inputs and the target.

Suppose now that we know $x^{(1)}, ..., x^{(n)}$ inputs as well as the $x^{(n+1)}$ observation for which we want to predict the target. We can then condition on the known targets to obtain the predictive distribution of $t^{(n+1)}$ given $t^{(1)}, ..., t^{(n)}$. The *predictive distribution* is Gaussian with mean and variance given as follows:

$$E\big[t^{(n+1)} \mid t^{(1)}, ..., t^{(n)}\big] = k^T C^{-1} t$$

$$Var\big[t^{(n+1)} \mid t^{(1)}, ..., t^{(n)}\big] = v - k^T C^{-1} k$$

where $\mathbf{C}$ is the n by n covariance matrix for the targets of the observed cases, $\mathbf{t}$= $[t^{(1)}, ..., t^{(n)}]^T$ is the vector of known targets, $\mathbf{k}$ is the vector of covariances between $t^{(n+1)}$ and the n known targets and v is the prior variance of $t^{(n+1)}$ (i.e. $Cov\big[t^{(n+1)}, t^{(n+1)}\big]$).

Models for **classification problems** can be defined in therms of a Gaussian process model for "latent values" associated with each case. When there are three or more classes, the model can be defined using K latent values for each case, $y_0^{(i)}, ..., y_{K-1}^{(i)}$, with class probabilities defined as follows:

$$P\big(t^{(i)} = k\big) = \frac{exp\big(-y_k^{(i)}\big)}{\sum_{s=0}^{K-1} exp\big(-y_s^{(i)}\big)}$$

The K latent values can be given independent Gaussian process priors, such as

$$Cov\big[y^{(i)}, y^{(j)}\big] = \eta^2 exp\Big(-\sum_{u=1}^{p} \rho_u^2 (x_u^{(i)} - x_u^{(j)})^2\Big) + \delta_{ij} J^2$$

in which J is a "jitter" component added for computational reasons and it is similar to the noise in a regression model (Neal 1997).

1.2 Covariance Matrix

The covariance function can have many different structures, depending on the condition that it must result in a positive semidefinite covariance matrix. In a Bayesian model, the covariance function usually depends on different "hyperparameters" that are given prior distributions as well. In Neal (1997) the covariance functions supported are the sum of one or more terms of the following types:

1) A constant part;

2) A linear part such as

$$\sum_{u=1}^{p} x_u^{(i)} x_u^{(j)} \sigma_u^2$$

3) A jitter part, which is zero for different cases, and a constant for the covariance of a case with itself. Jitter is used to improve the conditioning of the matrix computations, or to produce the effect of a probit classification model (Neal 1997);

4) Any number of exponential parts, each of which has the form:

$$\eta^2 \prod_{u=1}^{p} exp\Big(- \Big(\rho_u \Big| x_u^{(i)} - x_u^{(j)} \Big| \Big)^R \Big)$$

The plots below show four examples of Gaussian processes generated from different covariance functions using the Cholesky decomposition. These processes are produced following the stochastic representation of a multivariate normal distribution, that is

$$X = \mu + AZ$$

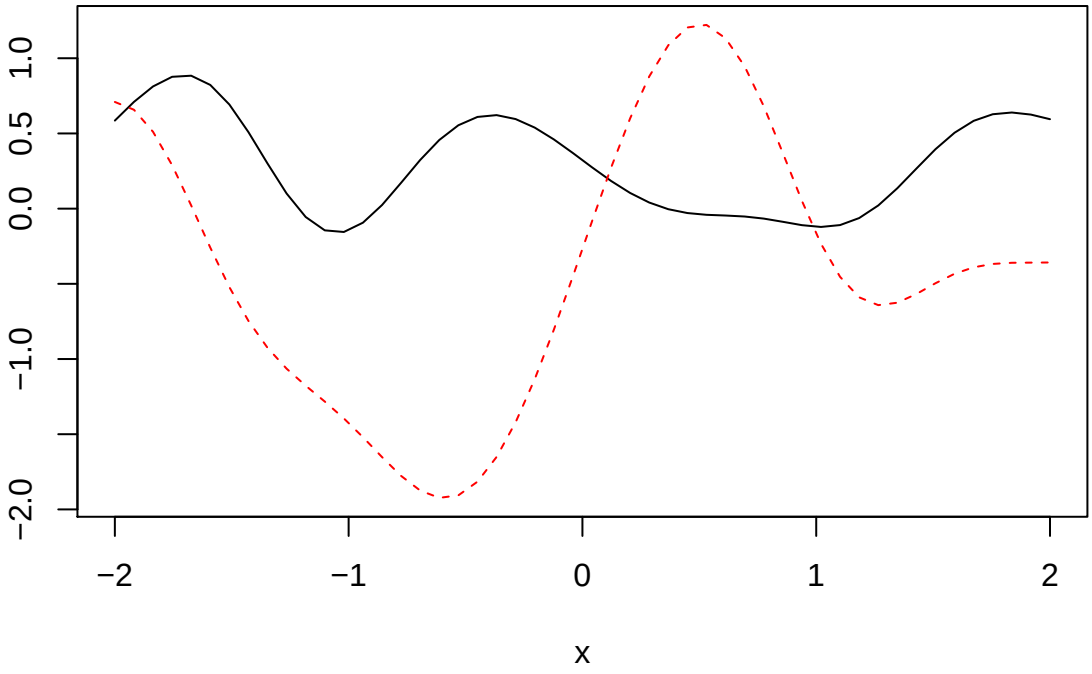
.

$Z = (Z_1, ...Z_k)$ is a k-dimentional vector with k independent normal random variables, A is *(d,k)* matrix and $\mu \in \mathbb{R}^d$ is the mean vector. The covariance matrix of X is $\Sigma = AA^T$, and the distribution of X (i.e. multivariate normal ditribution) is $X \sim N\big(\mu, \Sigma\big)$. Since Σ is a positive semidefinite matrix by definition, it can be decomposed as

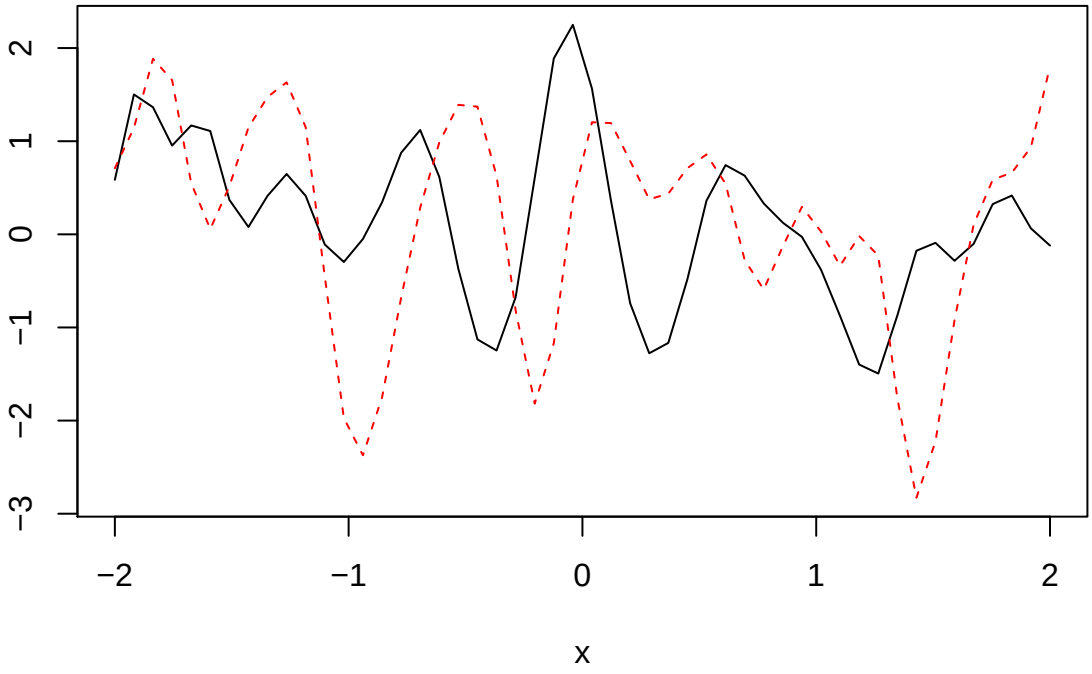
$$\Sigma = LL^T$$

where L is a lower triangular matrix with $L_{jj} \geq 0 \quad \forall j \in \{1, .., d\}$ and it's known as *Cholesky factor* in the *Cholesky decomposition* (Hofert 2013). Below some examples of Gaussian processes generated with different covariance functions with both expression and plot for each case.

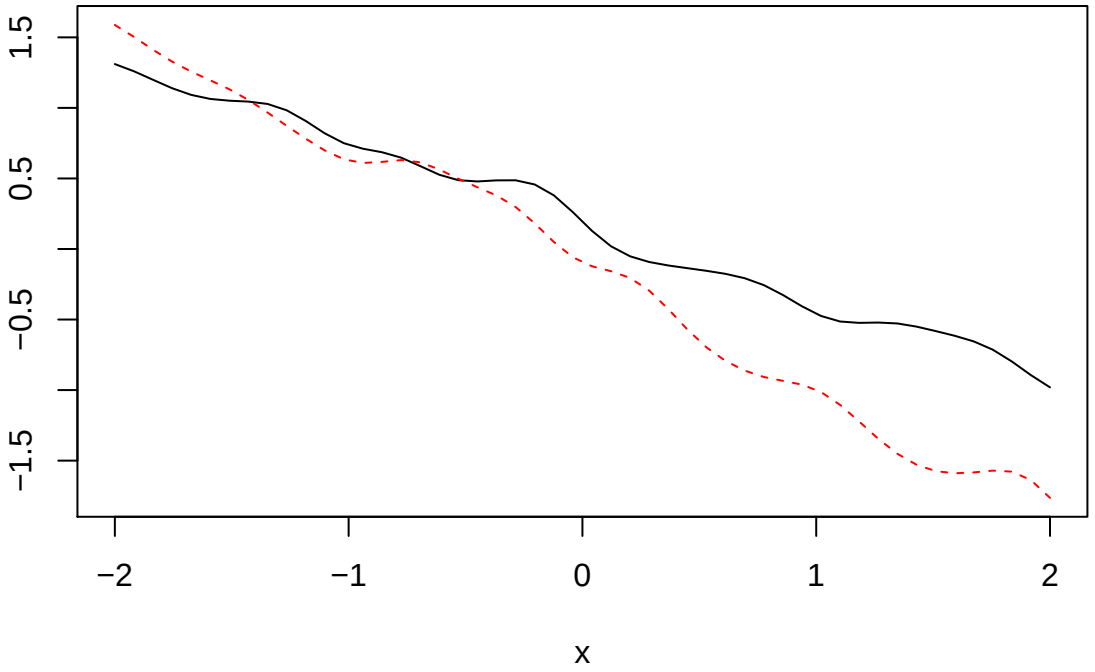
$$Cov\left[t^{(i)}, t^{(j)}\right] = exp(-2(x^{(i)} - x^{(j)})^2)$$



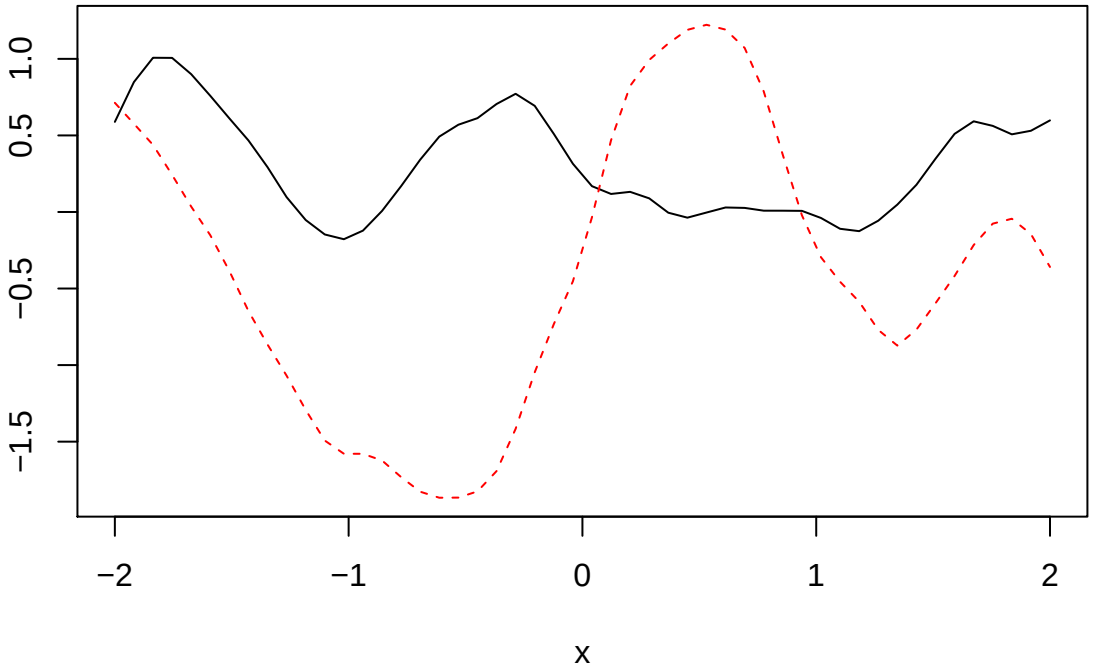
$$Cov\left[t^{(i)}, t^{(j)}\right] = exp(-5^2(x^{(i)} - x^{(j)})^2)$$



$$Cov\left[t^{(i)}, t^{(j)}\right] = 1 + x^{(i)}x^{(j)} + 0.1^2 exp(-3^2(x^{(i)} - x^{(j)})^2)$$



$$Cov\left[t^{(i)}, t^{(j)}\right] = exp(-2(x^{(i)} - x^{(j)})^2) + 0.1^2 exp(-5^2(x^{(i)} - x^{(j)})^2)$$



2 Applications

2.1 Regression models and Prior distributions

In this paragraph I'm going to describe the Regression model that will be used for the applications in next session (using RStan) and specify the priors choices for the hyperparameters. Following Neal's assumption over the prior's distribution, the model used for a Gaussian process with normal outcomes, $y \in \mathbb{R}^N$, with inputs $x \in \mathbb{R}^N$ is:

$$\rho \sim Gamma(5,5)$$
$$\eta \sim Cauchy(0,5)$$
$$\sigma \sim N(0,1)$$
$$f(x_i) \sim GP\Big(0, K\big(x \mid \rho, \eta\big)\Big)$$
$$y_i \mid f(x), \rho, \eta, \sigma \sim N\big(f(x_i), \sigma\big) \quad \forall \text{ i } \in 1, .., n$$

Bayesian predictive inference for Gaussian processes it's sped up by deriving the posterior analytically, then directly sampling from it. The predictive distribution has the following distribution:

$$p(\tilde{y}|\tilde{x}, y, x) = N(K^T\Sigma^{-1}y, \Omega - K^T\Sigma^{-1}K)$$

where $\Sigma = K(x \mid \eta, \rho, \sigma)$ is the result of applying the covariance function to the inputs x with observed outputs y, $\Omega = K(\tilde{x} \mid \eta, \rho)$ is the result of applying the covariance function to the inputs $\tilde{x}$ for which predictions are to be inferred, and K is the matrix of covariance between x and $\tilde{x}$.

The Stan model computes the analytic form of the posterior and provides the estimates sampling of the resulting multivariate normal through the Cholesky decomposition in order to cut down on the number of matrix-matrix multiplications when computing the conditional mean and the conditional covariance of $p(\tilde{y})$. The covariance function has the form

$$K(x|\eta, \rho, \sigma) = \eta^2 exp\bigg(-\frac{1}{2\rho^2}\sum_{d=1}^{D}\big(x_d^{(i)} - x_d^{(j)}\big)^2\bigg) + \delta_{ij}\sigma^2$$

The addition of σ^2 to the diagonal is important to ensure the positive definitness of the resulting matrix. The hyperparameter ρ is the *lenght-scale* and corresponds to the frequency of the functions represented by the Gaussian process prior with respect to the domain. Values of ρ closer to zero lead the GP to represent high-frequency functions, whereas larger values of ρ lead to low-frequency functions. The hyperparameter η is the *marginal standard deviation* and controls the magnitude of the range of the function represented by the GP (Documentation 2019).

Another implementation proposed in Neal (1997) is using a t-Student distribution for the gaussian noise; the model, with hyperparameters' priors are presented below.

$$\rho \sim Gamma(5,5)$$
$$\eta \sim Cauchy(0,5)$$
$$\sigma \sim St(4,0,1)$$
$$\nu \sim N(4,1)$$
$$f(x_i) \sim GP\Big(0, K\big(x \mid \rho, \eta\big)\Big)$$
$$y_i \mid f(x), \rho, \eta, \sigma \sim St\big(\nu, f(x_i), \sigma\big) \quad \forall \text{ i } \in 1, .., n$$

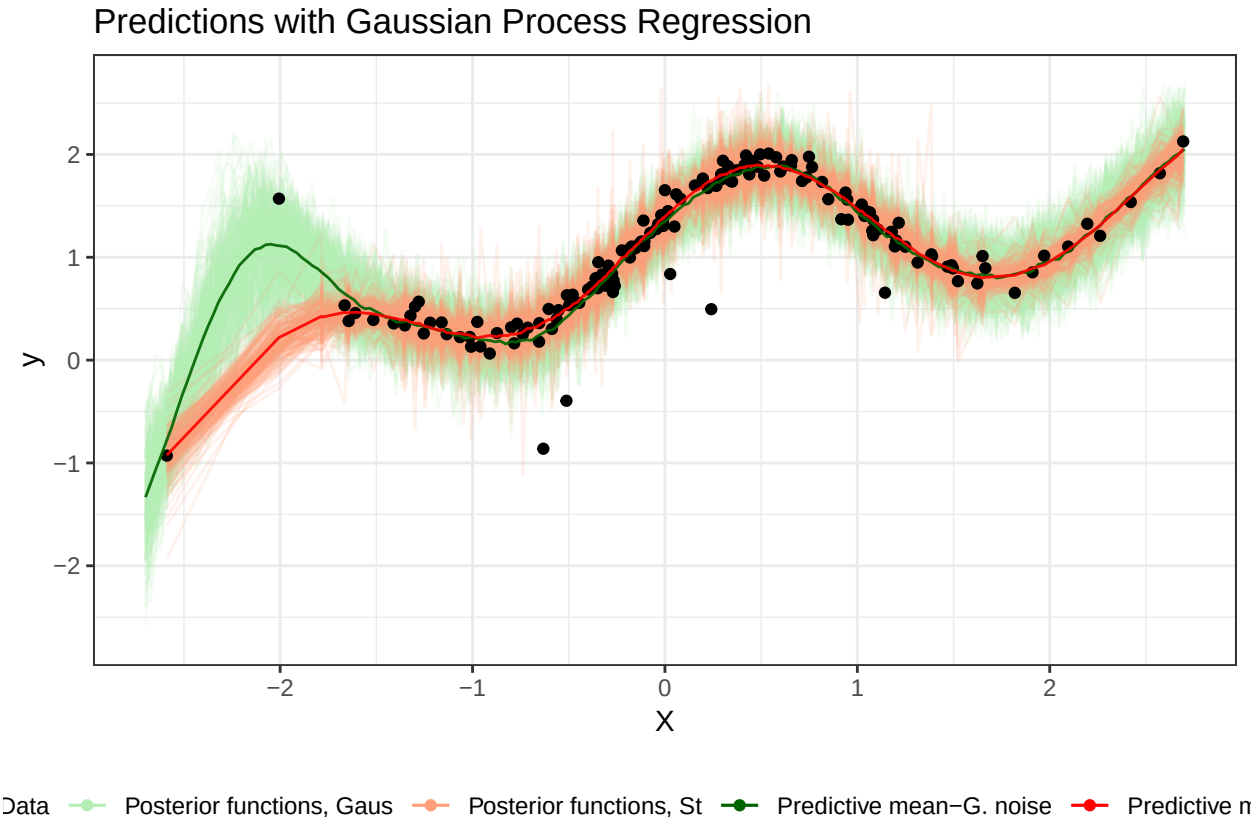
2.2 Regression problem with outliers

For both regression and classification examples I used the same data in Neal (1997). In this first example we have a single input generated from a standard Gaussian process and the corresponding target coming from a distribution with mean of

$$0.3 + 0.4x + 0.5sin(2.7x) + \frac{1.1}{1 + x^2}$$

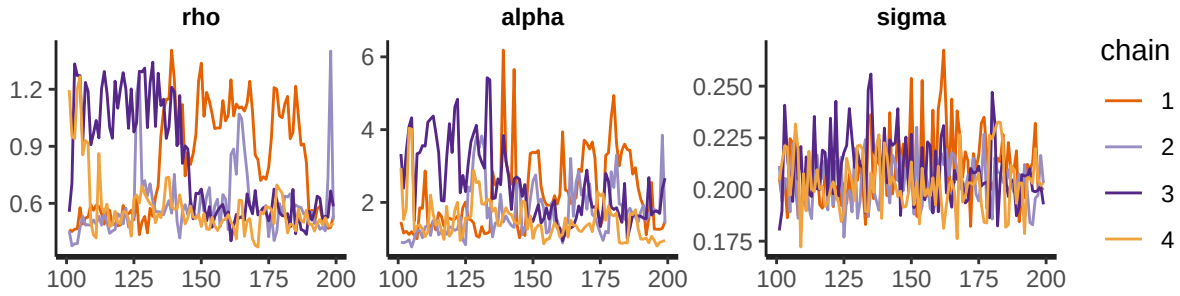
For most cases, the distribution of the target about this mean was Gaussian with standard deviation 0.1. However, with probability 0.05, a case was made an "outlier", for which the standard deviation was 1.0 instead (Neal 1997). I modeled these data with both models described in the previous paragraph, namely with Gaussian process for the expected value of the target, with the noise assumed to come from a t distribution with 4 degrees of freedom, and also under the assumption of Gaussian noise. For both models I used the Stan program used within R, with 200 iterations and 4 chains. Stan performs Hybrid Monte Carlo with Hamiltonian Monte Carlo updates for the hyperparameters and noise variance as well (with 10 leapfrog updates each sample).

The results below show that the regression model with t-Student noise better fits our data without being affected by the outliers with respect to the Gaussian noise model. Hence it is clear that the heavy tails of the t-distribution allow to these data to be modeled without the outliers having an undue effect. Moreover, the hyperpameters' simulations seem to converge reasonably faster in the t-student case and the distribution of the posterior's simulation better fits the "*true* y" distribution as well.

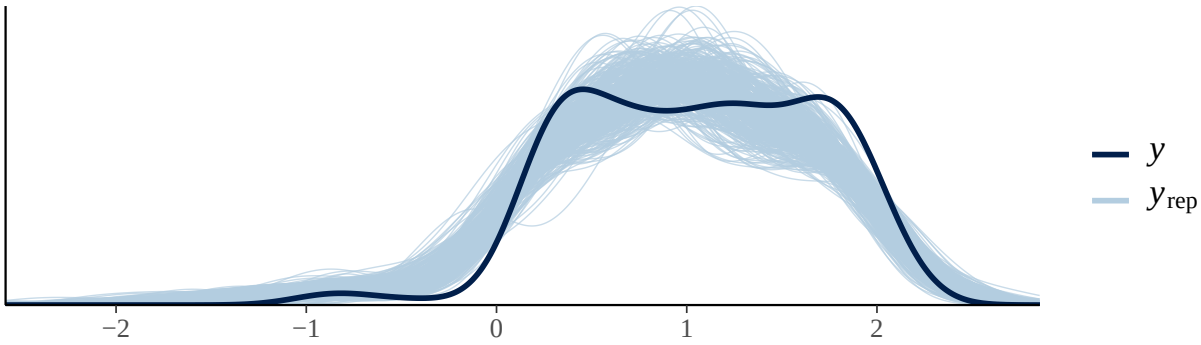


Gaussian noise regression plots

Parameters' traceplots

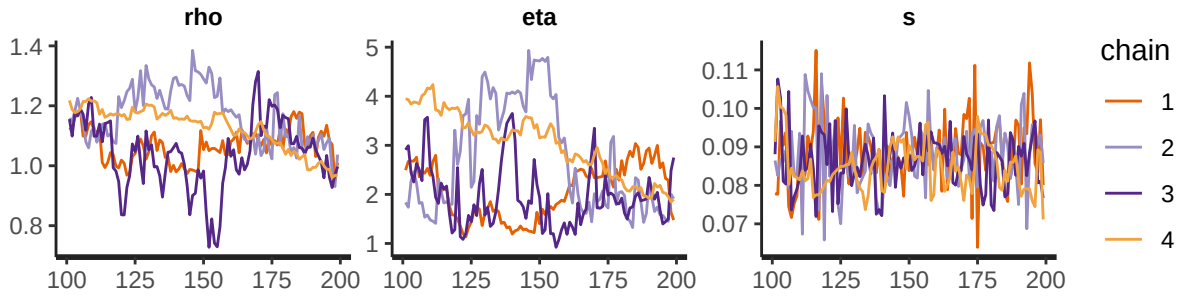


Distributions of both data and simulations

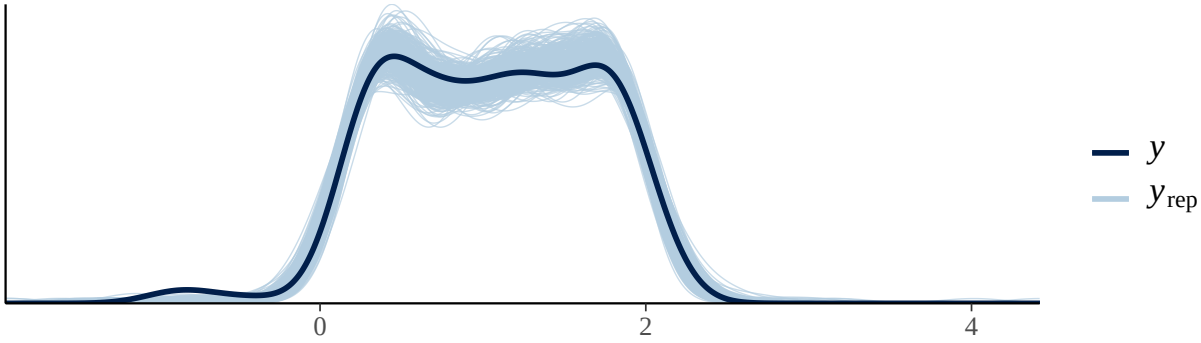


Student noise regression plots

Parameters' traceplots



Distributions of both data and simulations



GP-Regression with Gaussian noise *(Stan code)*

```
# functions {
#
#   vector gp_pred_rng(real[] x2,
#                      vector y1,
#                      real[] x1,
#                      real alpha,
#                      real rho,
#                      real sigma,
#                      real delta) {
#     int N1 = rows(y1);
#     int N2 = size(x2);
#     vector[N2] f2;
#     {
#       matrix[N1, N1] L_K;
#       vector[N1] K_div_y1;
#       matrix[N1, N2] k_x1_x2;
#       matrix[N1, N2] v_pred;
#       vector[N2] f2_mu;
#       matrix[N2, N2] cov_f2;
#       matrix[N2, N2] diag_delta;
#       matrix[N1, N1] K;
#       K = cov_exp_quad(x1, alpha, rho);
#       for (n in 1:N1)
#         K[n, n] = K[n,n] + square(sigma);
#       L_K = cholesky_decompose(K);
#       K_div_y1 = mdivide_left_tri_low(L_K, y1);
#       K_div_y1 = mdivide_right_tri_low(K_div_y1', L_K)';
#       k_x1_x2 = cov_exp_quad(x1, x2, alpha, rho);
#       f2_mu = (k_x1_x2' * K_div_y1);
#       v_pred = mdivide_left_tri_low(L_K, k_x1_x2);
#       cov_f2 = cov_exp_quad(x2, alpha, rho) - v_pred' * v_pred;
#       diag_delta = diag_matrix(rep_vector(delta, N2));
#
#       f2 = multi_normal_rng(f2_mu, cov_f2 + diag_delta);
#     }
#     return f2;
#   }
# }
# data {
#   int<lower=1> N1;
#   real x1[N1];
#   vector[N1] y1;
#   int<lower=1> N2;
#   real x2[N2];
# }
# transformed data {
#   vector[N1] mu = rep_vector(0, N1);
#   real delta = 1e-9;
# }
# parameters {
#   real<lower=0> rho;
#   real<lower=0> alpha;
```

```
#   real<lower=0> sigma;
#
# }
# model {
#   matrix[N1, N1] L_K;
#   {
#     matrix[N1, N1] K = cov_exp_quad(x1, alpha, rho);
#     real sq_sigma = square(sigma);
#
#     // diagonal elements
#     for (n1 in 1:N1)
#       K[n1, n1] = K[n1, n1] + sq_sigma;
#
#     L_K = cholesky_decompose(K);
#   }
#
#   rho ~ gamma(5, 5);
#   alpha ~ cauchy(0,5);
#   sigma ~ std_normal();
#
#   y1 ~ multi_normal_cholesky(mu, L_K);
# }
# generated quantities {
#   vector[N2] f2;
#   vector[N2] y2;
#
#   f2 = gp_pred_rng(x2, y1, x1, alpha, rho, sigma, delta);
#   for (n2 in 1:N2)
#     y2[n2] = normal_rng(f2[n2], sigma);
# }
```

GP-Regression with t-distributed noise *(Stan code)*

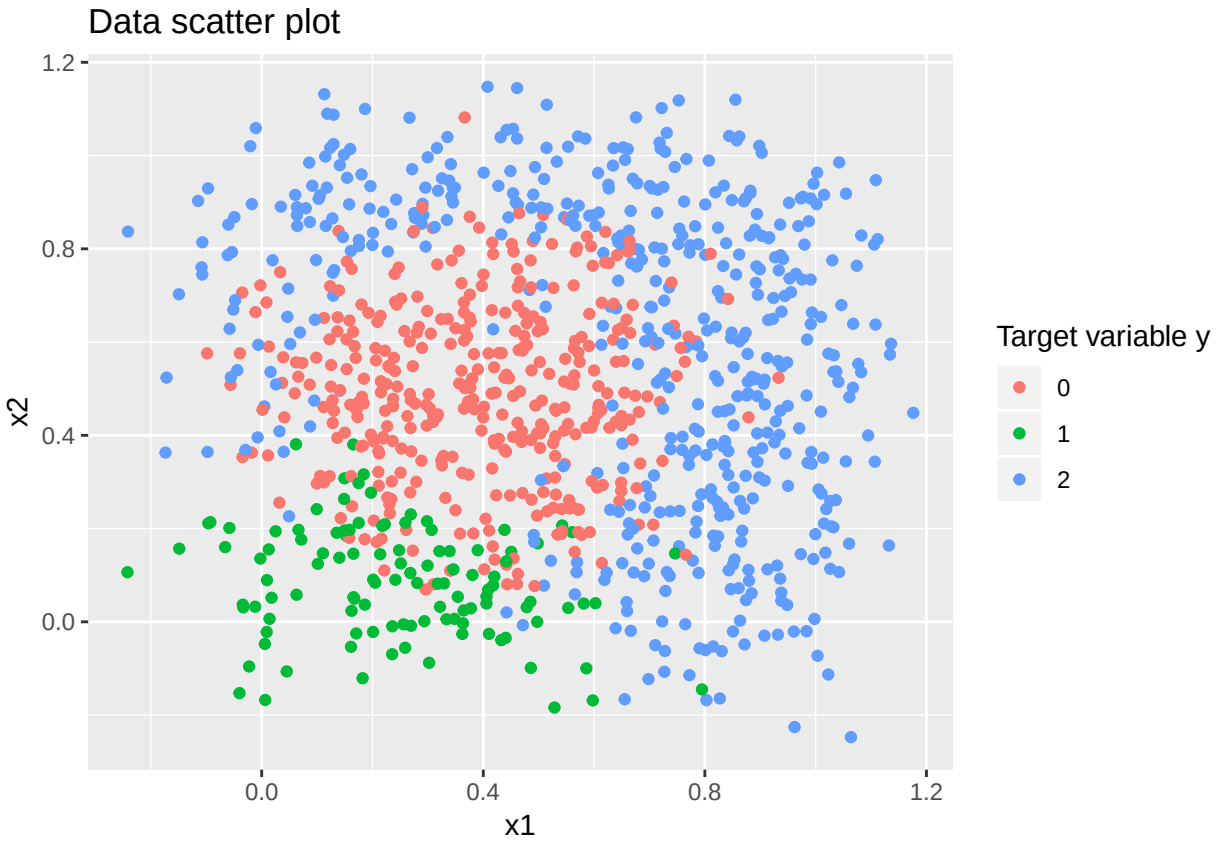
```
# data {
#   int<lower=1> N_predict;
#   real x_predict[N_predict];
#
#   int<lower=1> N_observed;
#   int<lower=1, upper=N_predict> observed_idx[N_observed];
#   real  y_observed[N_observed];
#
#
# }
#
# transformed data {
#
# }
#
# parameters {
#   vector[N_predict] f_tilde;
#   real<lower=0> dof; // Degrees of freedom for t
#   real<lower=0> s; // scale parametr for T
#   real<lower=0> rho; // lengthscale
#   real<lower=0> eta; //magnitude
# }
#
# transformed parameters {
#
#   matrix[N_predict, N_predict] cov =   cov_exp_quad(x_predict, eta, rho)
#                       + diag_matrix(rep_vector(1e-10, N_predict));
#   matrix[N_predict, N_predict] L_cov = cholesky_decompose(cov);
#   vector[N_predict] f_predict = L_cov * f_tilde;
# }
#
# model {
#   eta ~ cauchy(0,5);
#   rho ~ gamma(5,5);
#   f_tilde ~ normal(0, 1);
#   dof ~ normal (4,0.2);
#   s ~ student_t(4,0,1);
#   y_observed ~ student_t(dof, f_predict[observed_idx], s);//defines the likelihood
# }
#
# generated quantities {
#   vector[N_predict] y_predict;
#   for (n in 1:N_predict)
#     y_predict[n] = student_t_rng(dof, f_predict[n], s);// out of sample predictions
# }
```

2.3 A three-way classification problem

In this last paragraph we deal with the classification problem. We have a three-class target variable and four independent variables. I used Neal's data that are built in the following way: the four variables x_1, x_2, x_3 and x_4 are indepentently drawn from a $Unif(0,1)$; then the target variable takes value 0 if the euclidean distance between (x_1^i, x_2^i) and the point (0.4,0.5) is less than 0.35, we have $y = 1$ if $0.8 * x_1^i + 1.8x_2^i < 0.6$ and, finally, we have $y = 2$ otherwise. It is clear that the latent variable depends just on the first two variables (x_1 and x_2) and this relashionship between variables should demostrate the correct functioning of the model. For this classification problem the covariance structure has a different parametrization over the parameters:

$$K(x|\eta, \rho, \sigma) = c + \eta^2 exp\Big(-\sum_{d=1}^{D} \frac{1}{2\rho_u^2}\big(x_d^{(i)} - x_d^{(j)}\big)^2 \Big) + \delta_{ij}\sigma^2$$

where each variable has its own parameter ρ_u that can vary separetely from the others giving the model the capability to understand if an input is significant to explain the dependent value. on the contrary, if an input is irrelavant its posterior prediction should be close to zero. This estimation of ρ is defined as "*automatic relavance determination*" in Neal (2012), but this is misleading, because the magnitude the scale of the posterior for each ρ_u is dependent on the scaling of the input data and, moreover, the scale of the parameters ρ_u measures non-linearity along the d-th dimention. With one covariate,i.e. x_1, having a linear effect and another covariate x_2 having a nonlinear effect, it is possible that $\rho_1 > \rho_2$ even if the predictive relevance of x_1 is higher (Williams and Rasmussen 2006).



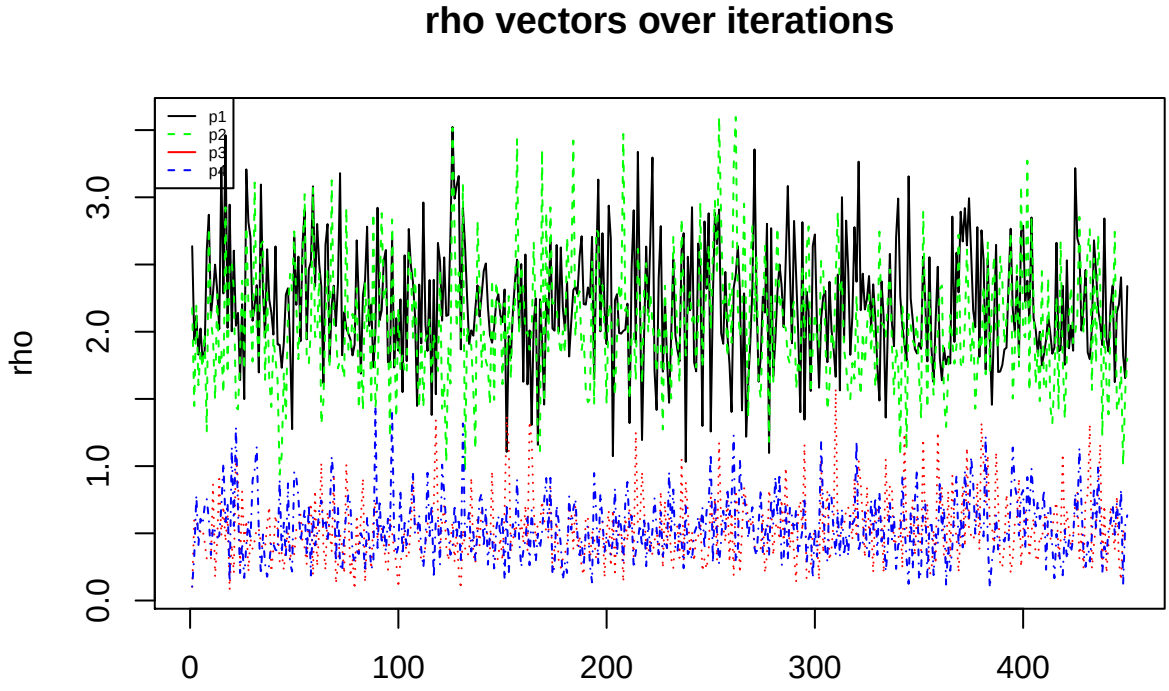
In order to apply the Gaussian process to a classification problem we need a function that allows us to express the probability that a certain observation belongs to a specific tharget, according with the corresponding covariates values. This function is called *softmax* and it has the following expression (already shown in the Introduction paragraph):

$$P\big(t^{(i)} = k\big) = \frac{exp\big(-y_k^{(i)}\big)}{\sum_{s=0}^{K-1} exp\big(-y_s^{(i)}\big)}$$

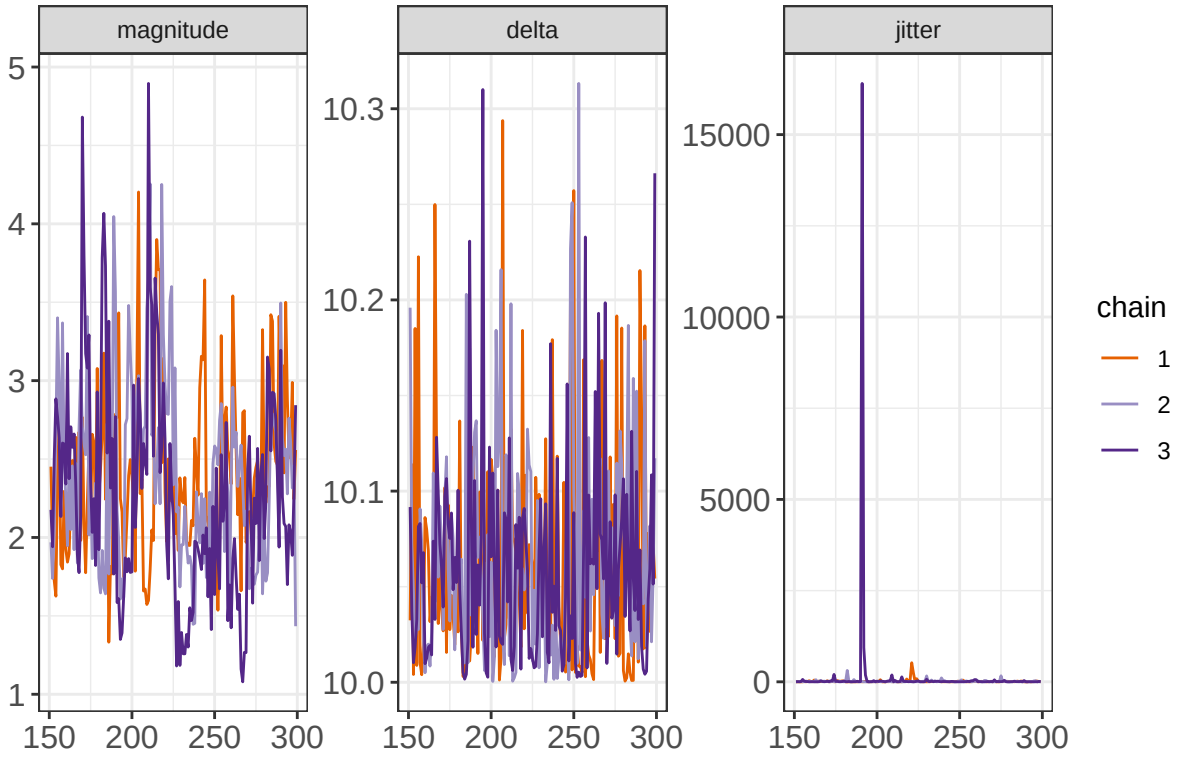
Hence the model has the following structure and prior distributions:

$$\rho \sim Gamma(5,5)$$
$$\eta \sim Cauchy(0,5)$$
$$\sigma \sim St(4,0,1)$$
$$f(x_i) \sim GP\Big(0, K\big(x \mid \rho, \eta\big)\Big)$$
$$y_i \mid f(x), \rho, \eta, \sigma \sim Mltn\big(y_i \mid softmax(f(x_i))\big)$$

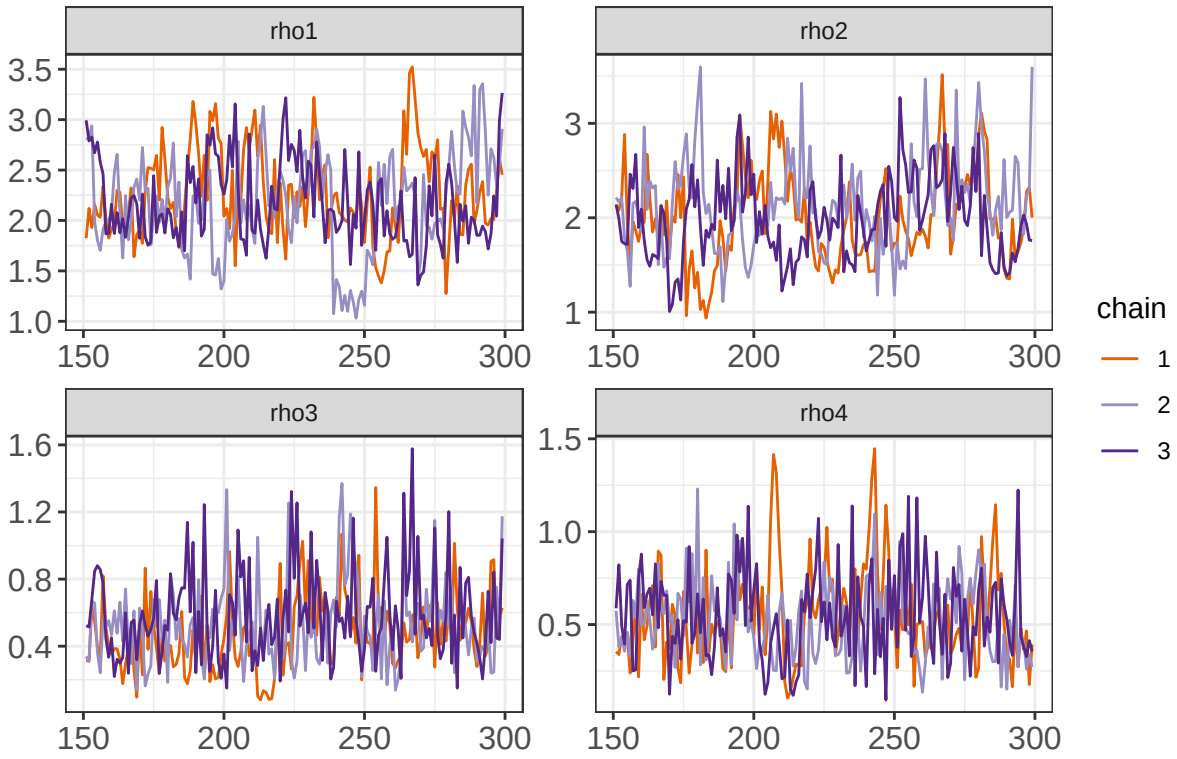
The result obtained show that the algorithm works quite good regarding the identification of irrelevant inputs: posterior estimates of the coefficients of both x_2 and x_3 are close to 0, in contrast to ρ_1 and ρ_2 whose estimates are around 2. Taking into account the accuracy of the classification, after splitting the dataset into train and test I obtain a good accuracy in the train (99%) while just the 70% on the test. This pour result could have been caused by the unbalancy of the dataset (we have 380 0s, 507 observations in which y=2 and just 112 for which y=1). Further works can be focused on the improvement of the classification accuracy.



Hyperparameters' traceplots



Exp. part hyperparameters' traceplots



Classification with Gaussian Process *(Stan code)*

```
# functions{
#
# /// Automatic Relevance Determination
#   matrix rbf(int N,int N_pred,vector x1,
#                       vector x2,vector x3,vector x4,vector x_1,
#                       vector x_2,vector x_3,vector x_4, real alpha,real rho1,
#                              real rho2,real rho3,real rho4,real delta,real jitter) {
#     int Nx = N;
#     int Ny = N_pred;
#     matrix[Nx, Ny] K;
#     real sq_alpha = square(alpha);
#     real sq_1 = square(rho1);
#     real sq_2 = square(rho2);
#     real sq_3 = square(rho3);
#     real sq_4 = square(rho4);
#     for (i in 1:(Nx-1)) {
#
#       K[i, i] = sq_alpha + delta;
#
#       for (j in (i + 1):Ny) {
#
#         K[i, j] =  sq_alpha *
#           exp(-0.5*sq_1*(x1[i] - x_1[j])^2 - 0.5*sq_2*(x2[i] - x_2[j])^2  -
#           0.5*sq_3*(x3[i] - x_3[j])^2  -  0.5*sq_4*(x4[i] - x_4[j])^2 );
#
#
#         K[j, i] = K[i, j];
#       }
#     }
#
#     K[Nx, Ny] = sq_alpha + delta;
#
#     return K + jitter;
#   }
#
#
#   vector gp_pred_rng(int N,int N_pred,
#                       vector y1, vector x1,
#                       vector x2,vector x3,vector x4,vector x_1,
#                       vector x_2,vector x_3,vector x_4,
#                       real magnitude,real rho1,
#                              real rho2,real rho3,real rho4,real delta,real jitter) {
#
#     vector[N_pred] f2;
#     {
#       matrix[N, N] C = rbf(N,N,x1,x2,x3,x4,x1,
#                       x2,x3,x4, magnitude,rho1,rho2,rho3,rho4,delta,jitter);
#
#       matrix[N, N] L_K = cholesky_decompose(C);
#       vector[N] L_K_div_y1 = mdivide_left_tri_low(L_K, y1);
#       vector[N] K_div_y1 = mdivide_right_tri_low(L_K_div_y1', L_K)';
#       matrix[N, N_pred] k_x_x_pred = rbf(N,N_pred,x1,x2,x3,x4,x_1,
```

```
#                       x_2,x_3,x_4, magnitude,rho1,rho2,rho3,rho4,delta,jitter);
#       f2 = (k_x_x_pred' * K_div_y1);
#     }
#     return f2;
#   }
# }
#
# data {
#   int<lower=1> N;
#   //for train cases
#   vector[N] x1;
#   vector[N] x2;
#   vector[N] x3;
#   vector[N] x4;
#   int<lower=1> N_pred;
#   // for test cases
#   vector[N_pred] x_1;
#   vector[N_pred] x_2;
#   vector[N_pred] x_3;
#   vector[N_pred] x_4;
#   int<lower=0,upper=2> y[N];
#   int<lower=1> S_train;
#   int<lower=1> S_test;
# }
# transformed data {
# //real delta = 1e-10;
# }
# parameters {
#   real<lower=0> magnitude;
#   real<lower=0> rho1;
#   real<lower=0> rho2;
#   real<lower=0> rho3;
#   real<lower=0> rho4;
#
#   real<lower=10> delta;
#    real<lower=0> jitter;
#
#   vector[N] f_tilde;
# }
# transformed parameters {
#   vector[N] f;
#
#   {
#     matrix[N, N] C;
#     matrix[N, N] L_cov;
#     C = rbf(N,N,x1,x2,x3,x4,x1,
#                       x2,x3,x4, magnitude,rho1,rho2,rho3,rho4,delta,jitter);
#     L_cov = cholesky_decompose(C);
#     f = L_cov * f_tilde;
#   }
# }
# model {
#   //Hyperparameters
```

```
#   magnitude ~ cauchy(0,5);
#   rho1 ~ gamma(5,5);
#   rho2 ~ gamma(5,5);
#   rho3 ~ gamma(5,5);
#   rho4 ~ gamma(5,5);
#   f_tilde ~ normal(0, 1);
#
#   delta ~ normal(10,1);
#   jitter ~ cauchy(0,5);
#   y ~ multinomial(softmax(f)); //Likelihood
# }
# generated quantities {
#   vector[N_pred] f_pred =  gp_pred_rng(N,N_pred,f,x1,x2,x3,x4,x_1,x_2,x_3,x_4,magnitude,rho1,rho2,rho
#   int y_pred[N_pred];
#   int y_pred_in[N];
#
#    y_pred_in = multinomial_rng(softmax(f),S_train);
#    y_pred = multinomial_rng(softmax(f_pred),S_test);
# }
```

References

Bernardo, J, J Berger, A Dawid, A Smith, and others. 1998. "Regression and Classification Using Gaussian Process Priors." *Bayesian Statistics* 6: 475.

Betancourt, Michael. 2017. "Robust Gaussian Processes in Stan." https://betanalpha.github.io/assets/case_studies/gp_part1/part1.html#2_gaussian_process_regression.

Documentation, Stan. 2019. "Gaussian Processes Regression." https://mc-stan.org/docs/2_19/stan-users-guide/gaussian-process-regression.html.

Hofert, Marius. 2013. "On Sampling from the Multivariate T Distribution." *The R Journal* 5 (2): 129–36.

Neal, Radford M. 1997. "Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification."

———. 2012. *Bayesian Learning for Neural Networks*. Vol. 118. Springer Science & Business Media.

Williams, Christopher KI, and Carl Edward Rasmussen. 2006. *Gaussian Processes for Machine Learning*. Vol. 2. 3. MIT press Cambridge, MA.