Final Project

이원석

December 14, 2020

1 Review of Gaussian Process models for spatial data

공간 통계에서 회귀분석은 다음과 같이 Mixed Model을 고려한다. (이 보고서에서 등장하는 모든 변수는 iid라고 생각한다.)

$$y(s_i) = x(s_i)^T\beta + w(s_i) + \epsilon(s_i) \text{ for } i = 1, \cdots, n$$

where $s_i, x(s_i), y(s_i)$ is location, $p \times 1$ covariate vector, response value. 여기서 β는 $p \times 1$ vector이며, $\epsilon(s_i)$는 random noise이다. 즉, $w(s_i)$는 spatial information의 variability를 다루고 있으며 $\epsilon(s_i)$는 location에서의 평균과 spatial variability로 설명할 수 없는 pure error를 의미한다.
모델링을 할때에 다음과 같은 가정을 이용하여 실행한다.

$$w \sim GP(0, C(\theta)),\ \epsilon \sim N(0, \tau^2 I)$$

여기서 w는 isotrophic gaussian process이다.
따라서 y는 다음과 같은 Mixed Model로 나타낼 수 있다(Latent model).

$$y|w \sim N(X\beta + w, \tau^2 I),\ w \sim N(0, C(\theta))$$

혹은 다음과 같이 위의 모델에서 w에 대한 분포를 곱한뒤 w에 대하여 적분을 하여 나타낼 수 있다(Response model).

$$y \sim N(X\beta, C(\theta) + \tau^2 I)$$

여기서는 y를 추정하기 보다는 parameter(θ)를 추정하는 bayesian 통계학을 적용한다. 빈도주의 통계학으로 y에 관한 분포를 추정하기에는 굉장히 어려운 과정이 필요하다. 하지만, bayesian 통계학에서는 통계학자의 사전지식을 이용하여 이를 반영 분포를 보다 쉽게 계산할 수 있다. 사후분포(posterior distribution;$\pi(\theta|y)$)를 계산하기위해서는 사전분포(prior distribution; $\pi(\theta)$)가 필요하다. posterior distribution은 likelihood($f(y|\theta)$)와 prior distribution의 곱으로 표현된다. (여기서의 θ는 앞의 θ와 다르게 모든 parameter를 말한다.)

$$\pi(\theta|y) \propto f(y|\theta)\pi(\theta)$$

위의 model에서는 w의 prior는 nonparametric bayesian distribution 중 하나인 gaussian process를 이용하여 posterior distribution을 계산한다. Gaussian process는 normal distribution의 parameter가 함수인것을 말한다. 다시말해, normal distribution의 평균 μ와 covariance matrix Σ가 θ에 대한 함수로써 표현된다는 의미이다.

그러나 Gaussian Process의 경우 posterior 분포를 구하기 위해서는 covariance matrix의 역행렬을 반드시 계산하여 한다. 역행렬을 계산할때에는 $O(n^3)$의 computation cost가 요구된다. 따라서 n이 커질때 computation cost는 세제곱으로 증가하여 계산 속도가 굉장히 느려지게 된다. computation cost를 줄이기 위한 연구가 계속 되고 있다.

그중에서 spatial statistics에서는 Nearest Neighbor Gaussian Processes라는 새로운 방법론이 제안 되었다 (Datta et al 2016).

2 Nearest Neighbor Gaussian Processes

Gaussian process는 location이 많아질 경우 많은 계산 시간을 요구한다. 또한 MCMC 역시도 computation cost를 많이 요구한다. 따라서 보다 효과적으로 적절한 시간내에 공간 자료를 분석하기 위해서는 computation cost를 조금 덜 요구하는 gaussian process가 필요하다.

Datta는 Nearest Neighbor Gaussian Processes라는 방법론을 제시하였다. 보통의 경우, w를 다음과 같이 가정한다.

$$w \sim N(0, C(\theta))$$

하지만, Datta는 다음과 같이 가정한다.

$$w \sim N(0, \tilde{C}(\theta))$$

$\tilde{C}(\theta)$는 원래 covariance matrix C와 비슷하지만 sparse matrix이다. 따라서 계산에 있어서 굉장히 computation cost를 획기적으로 줄일 수 있다($O(n)$).

이 행렬은 모든 location들의 거리를 구하는 것이 아니라 근접한 이웃(Nearest Neighbor)의 location만을 고려하여 계산하는 것이다. 특히나 NNGP는 Gaussian Markov Random Field(GMRF)의 특별한 경우이다. 어느 한 점이 모든 점들의 영향을 받기 보다 주변의 몇개의 점들에만 영향을 받는 것이라고 생각하는 경우이다.

3 Nearest Neighbor Gaussian Processes(NNGP) Model

3.1 Latent NNGP

Latent NNGP에서는 다음과 같이 posterior를 구한다.

$$\pi(w, \beta, \theta, \tau^2|y) \propto N(y|X\beta + w, \tau^2 I) \times N(w|0, \tilde{C}(\theta)) \times p(\beta, \theta, \tau^2)$$

w의 full conditional distribution은 다음과 같이 구할 수 있다.

$$w|\cdot \sim N(B(y - X\beta)\tau^2, B) \text{ where } B = \tilde{C}(\theta)^{-1} + I/\tau^2$$

하지만, Latent model에서는 $\tilde{C}$의 sparsity 때문에 inverse matrix를 구하는데 필요한 determinant를 계산하는데 어려움이 있다. Datta(2016)은 샘플링할 때에 $w_i|\cdot$를 하나씩 순차적으로 샘플링을 하는 것을 제안하고 있다.

3.2 Response NNGP

Latent NNGP 알고리즘에서는 $w_i|\cdot$의 방식으로 차례로 샘플링 하기 때문에 고차원에서는 매우 느리다. 따라서 w와 response를 따로 구분하지 않고 한번에 계산하는 것을 Finely (2019)가 제안하였다.

$$y \sim N(X\beta, \tilde{\Sigma}) \text{ where } \tilde{\Sigma} = \tilde{C}(\theta) + \tau^2 I$$

따라서 다음과 같이 posterior를 구할 수 있다.

$$\pi(\beta, \theta, \tau^2|y, w) \propto N(y|X\beta, \tilde{C}(\theta) + \tau^2 I) \times p(\beta, \theta, \tau^2)$$

Finely (2019)는 위의 방식으로 계산할 경우 parameter space가 $O(n)$ $O(1)$로 감소하는 것을 보였다.

4 Experiment

여기서 실험을 할때에는 다음과 같이 SAR model과 CAR model에서 적용되는 covaraince matrix를 설정하였다.

1. Latent NNGP Model

   - $$\pi(w, \beta, \theta, \tau^2|y) \propto N(y|X\beta + w, \tau^2 I) \times N(w|0, \tilde{C}(\theta)) \times p(\beta, \theta, \tau^2)$$

     where

     $$\tilde{C}^{-1} = (I - \tilde{A})^T \tilde{D}^{-1} (I - \tilde{A})$$

   - $\tilde{A}$ is a sparse and strictly lower triangular matrix with at most $M (M << N)$
   - $\tilde{D}$ is diagonal matrix

2. Response NNGP Model

   - $$\pi(\beta, \theta, \tau^2|y, w) \propto N(y|X\beta, \tilde{C}(\theta) + \tau^2 I) \times p(\beta, \theta, \tau^2)$$

     where

     $$\{\tilde{C} + \tau^2 I\}^{-1} = (I - A)^T D^{-1}(I - A)$$

   - A is a sparse and strictly lower triangular matrix with at most $M (M << N)$
   - D is diagonal matrix

3. 실험에서는 STAN을 이용하여 NNGP를 사용하였다. MCMC의 가장 기본 알고리즘은 Metropolis Hastings이다. 하지만 고차원에서 Likelihood를 계산할때에 많은 시간을 요구하며 수렴 속도도 매우 느려지게 된다. 따라서 No U Turn Sampler(NUTS)를 적용한 STAN을 이용하여 구현하였다. Metropolis Hastings 보다 빠르게 수렴하며, 계산 시간도 빠르며, 초기값의 영향을 덜 받는다.

4. 한 점의 nearest neighbor는 "spNNGP"패키지의 spConjNNGP를 이용하여 구하였다. spConjNNGP는 NNGP의 한 알고리즘으로 Gaussian univariate Bayesian conjugate spatial regression model을 구하여 nearest neighbor을 구한다. spNNGP 함수를 이용할 경우 따로 nearest neighbor를 구할 필요없이 지정해 주면 되며 Euclidean distance를 계산하여 가까운 거리를 계산한다.

5. 모두 동일한 hyperparameter를 주었다. 또한 초기값도 동일하게 주었다. 수렴시에 true값에 가까이 다가가기 때문에 어떠한 초기값을 주어도 상관없다.

6. 계산 과정에서 MCMC chain은 3개, 총 3000번의 샘플링에서 thinning을 5, warmup(burn-in)은 1500개로 설정하였다.

7. Latent NNGP의 model에서 모든 w의 element의 posterior distribution을 보지 않고 4개의 element의 posterior distribution만을 참고하겠다.

4.1 Simulation Data

4.1.1 Data 소개

simulation을 통하여 300개의 데이터를 생성하였다.

$$y(s) = \beta_0 + x(s)\beta_1 + w(s) + \epsilon(w), \text{ where } \epsilon(s) \sim N(0, \tau^2 I)$$

w의 공분산 행렬은 다음과 같이 설정하였다.

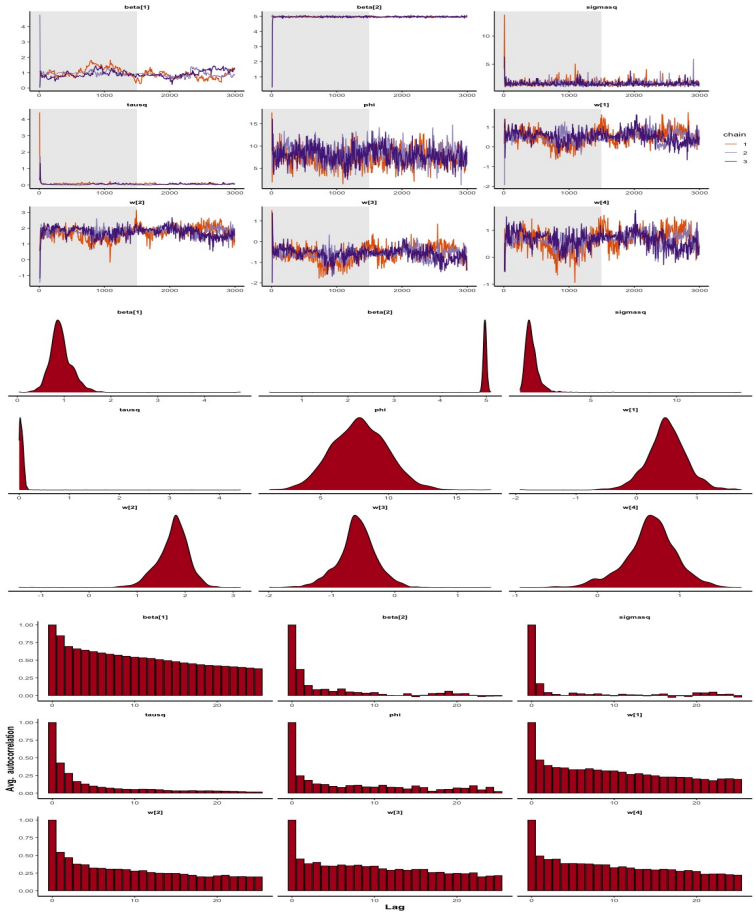
$$C(\theta) = \sigma^2 \exp(-\phi \|s_i - s_j\|)$$

마지막으로 x는 $N(0, 1)$에서 생성하였다.

4.1.2 posterior 분석

회색부분으로 색칠되어 있는 부분은 burnin으로 취급한다. parameter들의 trace plot과 summary statistic은 다음과 같다.

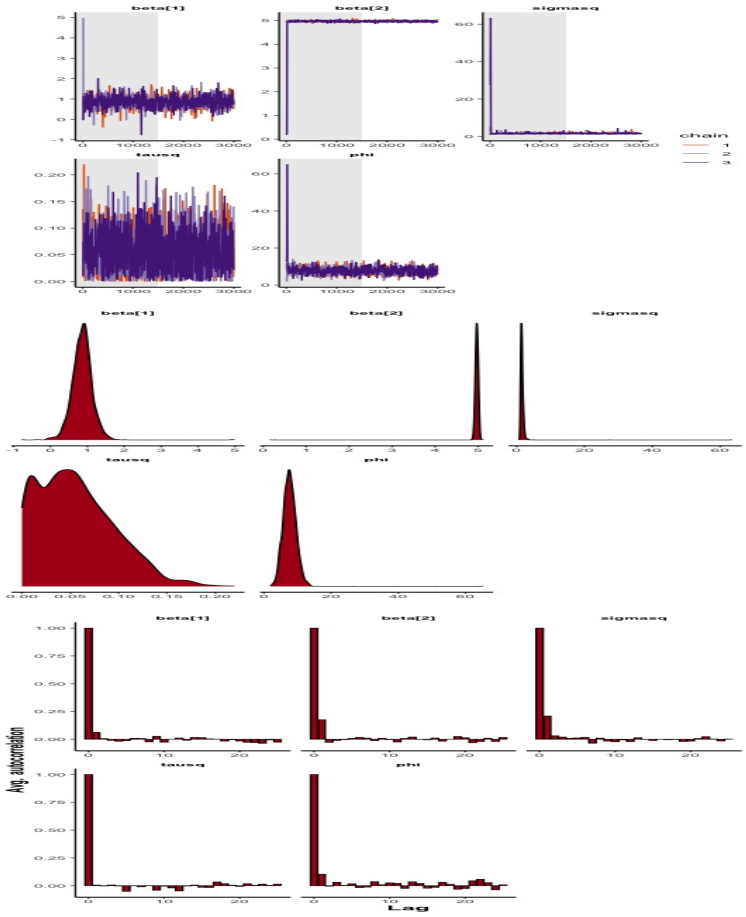
1. Latent NNGP



-

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	0.8746	0.2199	0.7329	0.8531	0.9958	17
beta[2]	4.9691	0.0338	4.9470	4.9683	4.9910	290
sigmasq	1.5543	0.4093	1.3016	1.4765	1.7205	642
• tausq	0.0499	0.0351	0.0211	0.0447	0.0726	12
phi	7.8851	2.0081	6.4607	7.7770	9.2545	100
w[1]	0.5493	0.3248	0.3553	0.5389	0.7419	47
w[2]	1.8084	0.3106	1.6312	1.8353	2.0049	35
w[3]	-0.5493	0.2983	-0.7129	-0.5451	-0.3686	42
w[4]	0.7047	0.2999	0.5392	0.7150	0.8867	35

2. Reponse NNGP



-

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	0.8487	0.2657	0.6805	0.8566	1.0096	866
beta[2]	4.9696	0.0326	4.9486	4.9700	4.9915	912
sigmasq	1.5512	0.3696	1.3087	4.9700	1.7151	603
tausq	0.0564	0.0391	0.0258	0.0528	0.0820	853
phi	1.5512	1.8319	6.2814	7.5925	8.7555	708

3. $\beta, \sigma^2, \tau^2, \phi$의 posterior distribution은 모두 비슷하다. trace plot, autocorrelation plot, 그리고 ESS를 보았을때 Response NNGP의 모델이 훨씬 더 좋다는 것을 알 수 있다. w의 uncertainty를 미리 계산하여 적용한것이 가장 주요한 차이점이라고 보인다. simulation data에서는 τ^2를 제외한 parameter들은 잘 수렴하였다. Response NNGP를 적용한다면 훨씬 더 효과적인 계산을 통하여 데이터를 분석할 수 있다. 도출한 posterior distribution을 통하여 새로운 데이터가 주어질때의 predictive distribution을 계산하여 y를 추정할 수 있다.

4.2 LDNSuicides 데이터

4.2.1 Data 소개

LDNSuicides.csv는 Blangiardo et al. (2013)에서 제안한 데이터이다. 1989년 1993년동안 런던 자치구의 자살 건수를 포함하고 있다. 포함하고 있는 데이터는 다음과 같다.

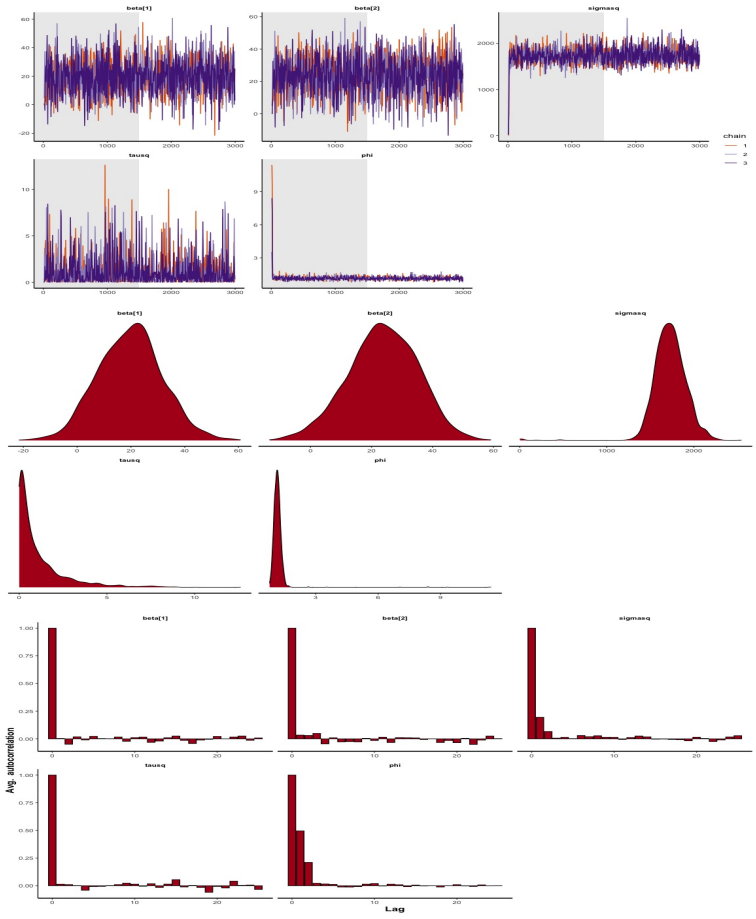
- ID : IDs of the boroughs in London
- NAME : the names of the boroughs in London
- y : the numbers of suicides
- E : the expected numbers of suicides
- x_1: the values of an index of social deprivation (lack of social connections)
- x_2: the values of an index of social fragmentation (lack of sense of community)

모델링에서 y값은 y로 두었고, x는 (x_1, x_2)로 두었다.

4.2.2 posterior 분석

회색부분으로 색칠되어 있는 부분은 burnin으로 표시하였다. parameter들의 trace plot과 summary statistic은 다음과 같다.

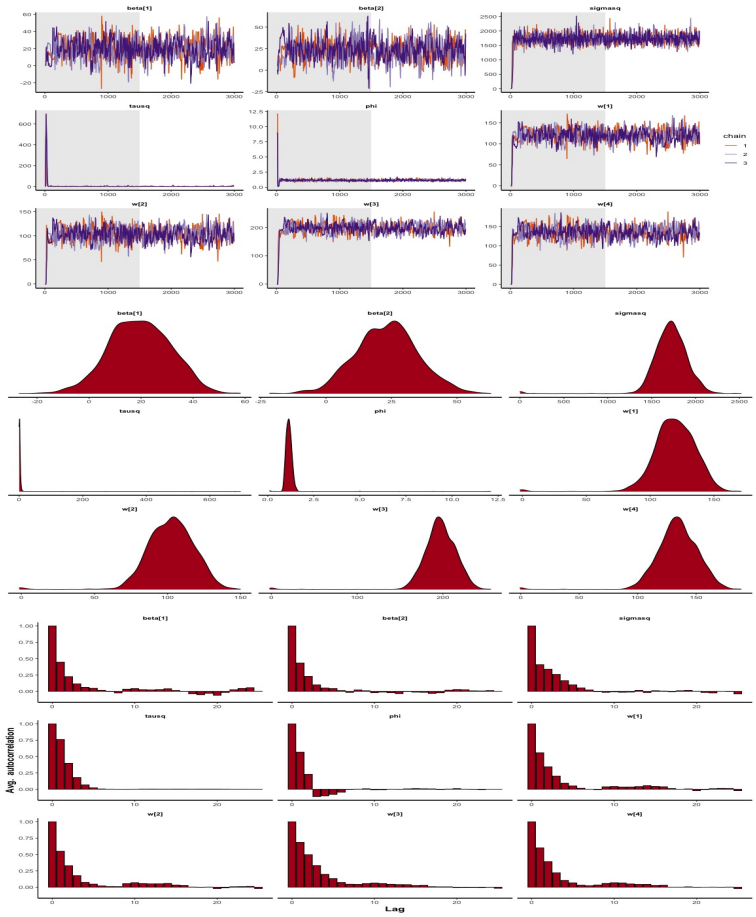
1. Latent NNGP



-

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	19.364	11.951	10.877	19.216	28.206	367
beta[2]	22.681	11.887	14.768	22.797	30.300	366
sigmasq	1729.9	166.37	1614.1	1722.7	1834.0	964
tausq	1.7826	2.0056	0.4189	1.0995	2.3975	161
phi	1.1593	0.1477	1.0499	1.1485	1.2557	790
w[1]	119.54	15.036	109.04	119.23	130.54	373
w[2]	102.20	14.574	91.884	102.51	112.41	353
w[3]	196.70	16.247	185.55	196.30	208.18	325
w[4]	133.72	15.493	123.15	133.16	144.46	335

2. Reponse NNGP



-

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	19.988	11.853	12.2478	20.6217	27.227	902
beta[2]	24.011	11.550	16.3987	24.4820	32.216	709
sigmasq	1739.9	169.57	1615.63	1733.53	1850.8	871
tausq	1.0077	1.3849	0.1116	0.4719	1.3266	859
phi	1.1581	0.1485	1.0551	1.1538	1.2509	941

3. $\beta, \sigma^2, \tau^2, \phi$의 posterior distribution은 모두 비슷하다. trace plot, autocorrelation plot, 그리고 ESS를 보았을때 Response NNGP의 모델이 훨씬 더 좋다는 것을 알 수 있다. w의 uncertainty를 미리 계산하여

적용한것이 가장 주요한 차이점이라고 보인다. LDNSuicides data에서는 모든 parameter들은 잘 수렴하였다. Response NNGP를 적용한다면 훨씬 더 효과적인 계산을 통하여 데이터를 분석할 수 있다. 도출한 posterior distribution을 통하여 새로운 데이터가 주어질때의 predictive distribution을 계산하여 y를 추정할 수 있다.

4.3 meuse 데이터

4.3.1 Data 소개

meuse데이터는 gstat 패키지에 포함되어 있는 데이터이다. 이는 네달란드의 슈타인 마을 근처의 meuse 강 옆 들판의 중금속 농도를 측정한 데이터를 포함하고 있다. 구성요소는 다음과 같다.

- x: Easting (in meters).
- y: Northing (in meters).
- cadmium: Topsoil cadmium concentration (in ppm).
- copper: Topsoil copper concentration (in ppm).
- zinc: Topsoil zinc concentration (in ppm).
- lead: Topsoil lead concentration (in ppm).
- etc

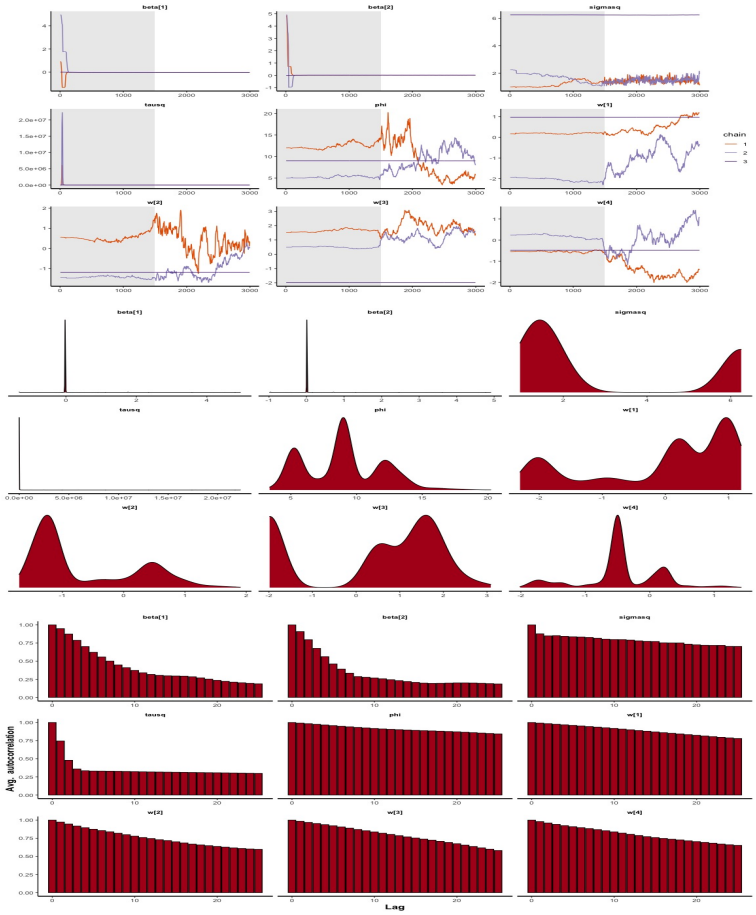
모델링에서 y값은copper로 두었고, x는 (x, y)로 두었다.

4.3.2 posterior 분석

회색부분으로 색칠되어 있는 부분은 burnin으로 표시하였다. parameter들의 trace plot과 summary statistic은 다음과 같다.

1. Latent NNGP

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	-0.0218	0.0024	-0.0228	-0.0219	-0.0210	324
beta[2]	0.0119	0.0013	0.0115	0.0120	0.0125	324
sigmasq	3.0623	2.2540	1.4050	1.5813	6.2383	1
tausq	165.11	60.522	149.06	191.18	201.83	2
phi	9.0714	2.8782	7.3017	8.9927	10.189	8
w[1]	0.1746	0.9042	-0.5538	0.4128	0.9706	1
w[2]	-0.6157	0.8283	-1.1901	-1.1688	0.0087	2
w[3]	0.3982	1.7395	-1.9814	1.2282	1.6982	1
w[4]	-0.6125	0.7513	-1.2848	-0.4794	-0.4722	2

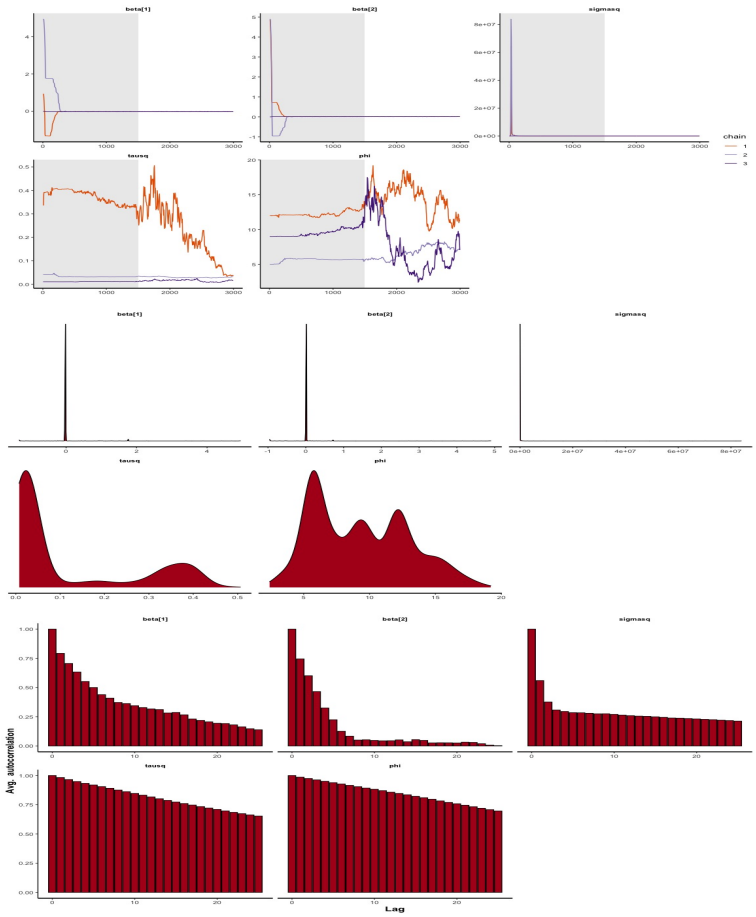


2. Reponse NNGP

Summary Statistics						
parameter	mean	sd	25%	50%	75%	ESS
beta[1]	-0.0216	0.0042	-0.0244	-0.0214	-0.0189	390
beta[2]	0.0118	0.0023	0.0103	0.0117	0.0134	390
sigmasq	429.91	41.901	399.60	429.33	458.28	64
tausq	0.0885	0.1161	0.0178	0.0293	0.1183	2
phi	9.4997	4.3065	5.8894	7.8507	13.906	2

-

3. $\beta, \sigma^2, \tau^2, \phi$의 posterior distribution은 모두 비슷하다. trace plot, autocorrelation plot, 그리고 ESS를 보았을때 Response NNGP의 모델이 훨씬 더 좋다는 것을 알 수 있다. w의 uncertainty를 미리 계산하여 적용한것이 가장 주요한 차이점이라고 보인다. meuse data에서는 모든 parameter들이 잘 수렴하지 않았다. thinnig에도 불구하고 autocorrelation이 너무 굉장히 높게 나왔으며 posterior distribuion이 multi modal인 경우도 있다. 이 데이테어서는 Response NNGP를 적용한다면 훨씬 더 효과적인 계산을 할 수 있지만 parameter들의 posterior distribution을 효과적으로 구하지는 못하였다. 다른 MCMC 알고리즘을 통하여 실험이 필요할 것으로 보인다.



5 Summary and Conclusions

이번 보고서를 통해서 Nearest Neighbor Gaussian Processes(NNGP)를 살펴보았다. 공간 정보를 담고 있는 w에 gaussian process라는 nonparametric bayesian prior를 주어 데이터를 분석한다. 하지만 $O(n^3)$라는 computation cost가 필요하다. 이는 고차원 데이터 혹은 데이터셋의 크기가 굉장히 커지는 경우에서는 굉장히 높은 computation cost를 요구하게 되는데 이를 해결하려는 시도 중 하나가 NNGP이다. 모든 점 사이의 거리를 구하는 것 보다 몇개의 인접한 점들 사이의 거리를 구하여 gaussian process의 covariance matrix를 sparse matrix로 만들어 계산이 보다 더 편리하고 빠르도록 한다. NNGP에서는 Latent model과 response model로 나뉘어진다. Latent model은 w를 parameter로 취급하는 것이고 Response model은 Latent model의 w를 적분하여 계산 모델을 이용한다. 대부분의 데이터셋에서 Latent model의 ESS가 Response model의 ESS보다 낮았다. trace plot에서는 각 parameter의 수렴상태를 볼 수 있었는데 여기에서도 reponse model이 훨씬 잘 수렴한다는 것을 볼 수 있었다. 나아가 Latent model이 계산해야되는 parameter의 수가 많기 때문에 계산시간이 Response model보다 상대적으로 많은 시간이 필요하다. 따라서 NNGP를 이용할때에는 response model을 적용하는 것이 더 효과적이라는 결론을 얻을 수 있었다. meuse 데이터에서 볼 수 있듯이 y의 attribute가 다양할때에도 full gaussian process를 이용하여 분석하는 것보다 sparse한 covariance를 가지는 gaussian process를 생각해볼 수 있다.

6 References

1. Datta A, Banerjee S, Finley AO, Gelfand AE(2016). *Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets.* Journal of the American Statistical Association

2. Hoffman, Gelman(2011). *The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.* Journal of Machine Learning Research

3. Finley, Datta, Banerjee(2020). *spNNGP R package for Nearest Neighbor Gaussian Process models.* ArXiv

final_project

이원석

2020 12 13

주제

Finley, Datta, Banerjee(2020).spNNGP R package for Nearest Neighbor Gaussian Process models.ArXiv 를 STAN으로 구현해보고 데이터를 분석하는 것입니다. Code RSTAN community를 참고하였으며 보고서의 주목적은 데이터를 분석하는 것에 있습니다. Code는 다음과 같습니다. 기존의 NNGP를 제안한 논문도 참고하였습니다.(Datta A, Banerjee S, Finley AO, Gelfand AE(2016).Hierarchical Nearest-Neighbor Gaussian ProcessModels for Large Geostatistical Datasets.Journal of the American Statistical Association)

코드

```
writeLines(readLines('NNMatrix.R'))
```

```
## library(spNNGP)
##
## i_dist <- function(i, neighbor_index, s){
##   dist(s[c(i, neighbor_index[[i - 1]]), ])
## }
##
## get_NN_distM <- function (ind, ind_distM_d, M) {
##   if (ind < M ){l = ind } else {l = M};
##   M_i <- rep(0, M * (M - 1) / 2);
##   if (l == 1) {}
##   else{
##     M_i[1: (l * (l - 1) / 2)] <-
##       c(ind_distM_d[[ind]])[(l + 1): (l * (l + 1) / 2)]
##   }
##   return(M_i)
## }
##
## get_NN_dist <- function (ind, ind_distM_d, M) {
##   if (ind < M ){l = ind } else {l = M};
##   D_i <- rep(0, M);
##   D_i[1:l] <- c(ind_distM_d[[ind]])[1:l]
##   return(D_i)
## }
##
## get_NN_ind <- function (ind, ind_distM_i, M) {
##   if (ind < M ){l = ind } else {l = M};
##   D_i <- rep(0, M);
##   D_i[1:l] <- c(ind_distM_i[[ind]])[1:l]
##   return(D_i)
## }
##
## NNMatrix <- function(coords, n.neighbors, n.omp.threads = 2,
##                      search.type = "cb", ord = order(coords[, 1])){
##
##   N <- nrow(coords)
##   m.c <- spConjNNGP(rep(0, N) ~ 1, coords = coords,
##                     n.neighbors = n.neighbors,
##                     theta.alpha = c("phi" = 5, "alpha" = 0.5),
##                     sigma.sq.IG = c(2, 1),
##                     cov.model = "exponential",
##                     n.omp.threads = n.omp.threads,
##                     search.type = search.type,
##                     ord = ord,
##                     return.neighbor.info = T, fit.rep = F,
##                     verbose = F)
##   M = n.neighbors
##   NN_ind <- t(sapply(1: (N - 1), get_NN_ind, m.c$neighbor.info$n.indx[-1], M))
##   neighbor_dist <- sapply(2:N, i_dist, m.c$neighbor.info$n.indx[-1],
##                           m.c$coords[m.c$neighbor.info$ord, ])
##   NN_distM <- t(sapply(1: (N - 1), get_NN_distM, neighbor_dist, M))
##   NN_dist <- t(sapply(1: (N - 1), get_NN_dist, neighbor_dist, M))
##
##   return(list(ord = m.c$neighbor.info$ord
```

```
##   return(list(ord = m.c$neighbor.info$ord,
##               coords.ord = m.c$coords[m.c$neighbor.info$ord, ],
##               NN_ind = NN_ind, NN_distM = NN_distM, NN_dist = NN_dist))
## }
##
##
## Check_Neighbors <- function(coords, n.neighbors, NN.matrix, ind){
##
##   plot(coords)
##   points(coords[1:ind, , drop = FALSE], col = "grey", pch = 19)
##
##   if (ind < n.neighbors) {dim = ind} else {dim = n.neighbors}
##   for (j in 1:dim){
##     points(coords[NN.matrix$NN_ind[ind - 1, j], , drop = FALSE],
##            col = "orange",  pch = 19)
##   }
##   points(coords[ind, , drop = FALSE], col = "blue", pch = 19)
##   legend("topright", inset=.05,
##          c("obs", paste0( ind, "th obs"),
##            paste0("neighbors of ", ind,"th obs"),
##            paste0("obs indexed <", ind)), pch = c(1, 19, 19, 19),
##          col=c("black", "blue", "orange", "grey"), horiz = F)
## }
```

```
writeLines(readLines('nngp_response.stan'))
```

```
##   /* Response NNGP model */
##
##   functions{
##       real nngp_lpdf(vector Y, vector X_beta, real sigmasq, real tausq,
##                      real phi, matrix NN_dist, matrix NN_distM, int[,] NN_ind,
##                      int N, int M){
##
##           vector[N] V;
##           vector[N] YXb = Y - X_beta;
##           vector[N] U = YXb;
##           real kappa_p_1 = tausq / sigmasq + 1;
##           int dim;
##           int h;
##
##           for (i in 2:N) {
##               matrix[ i < (M + 1) ? (i - 1) : M, i < (M + 1) ? (i - 1): M]
##               iNNdistM;
##               matrix[ i < (M + 1) ? (i - 1) : M, i < (M + 1) ? (i - 1): M]
##               iNNCholL;
##               vector[ i < (M + 1) ? (i - 1) : M] iNNcorr;
##               vector[ i < (M + 1) ? (i - 1) : M] v;
##               row_vector[i < (M + 1) ? (i - 1) : M] v2;
##               dim = (i < (M + 1))? (i - 1) : M;
##
##               if(dim == 1){iNNdistM[1, 1] = kappa_p_1;}
##               else{
##                   h = 0;
##                   for (j in 1:(dim - 1)){
##                       for (k in (j + 1):dim){
##                           h = h + 1;
##                           iNNdistM[j, k] = exp(- phi * NN_distM[(i - 1), h]);
##                           iNNdistM[k, j] = iNNdistM[j, k];
##                       }
##                   }
##                   for(j in 1:dim){
##                       iNNdistM[j, j] = kappa_p_1;
##                   }
##               }
##
##               iNNCholL = cholesky_decompose(iNNdistM);
##               iNNcorr = to_vector(exp(- phi * NN_dist[(i - 1), 1: dim]));
##
##              v = mdivide_left_tri_low(iNNCholL, iNNcorr);
##
##              V[i] = kappa_p_1 - dot_self(v);
##
```

```
##              v2 = mdivide_right_tri_low(v', iNNCholL);
##
##              U[i] = U[i] - v2 * YXb[NN_ind[(i - 1), 1:dim]];
##            }
##            V[1] = kappa_p_1;
##            return - 0.5 * ( 1 / sigmasq * dot_product(U, (U ./ V)) +
##                            sum(log(V)) + N * log(sigmasq));
##        }
##    }
##
##    data {
##        int<lower=1> N;
##        int<lower=1> M;
##        int<lower=1> P;
##        vector[N] Y;
##        matrix[N, P + 1] X;
##        int NN_ind[N - 1, M];
##        matrix[N - 1, M] NN_dist;
##        matrix[N - 1, (M * (M - 1) / 2)] NN_distM;
##        vector[P + 1] uB;
##        matrix[P + 1, P + 1] VB;
##        real ss;
##        real st;
##        real ap;
##        real bp;
##    }
##
##    transformed data {
##        cholesky_factor_cov[P + 1] L_VB;
##        L_VB = cholesky_decompose(VB);
##    }
##
##    parameters{
##        vector[P + 1] beta;
##        real<lower = 0> sigma;
##        real<lower = 0> tau;
##        real<lower = 0> phi;
##    }
##
##    transformed parameters {
##        real sigmasq = square(sigma);
##        real tausq = square(tau);
##    }
##
##    model{
##        beta ~ multi_normal_cholesky(uB, L_VB);
##        phi ~ gamma(ap, bp);
##        sigma ~ normal(0, ss);
##        tau ~ normal(0, st);
##        Y ~ nngp(X * beta, sigmasq, tausq, phi, NN_dist, NN_distM, NN_ind, N, M);
##    }
```

```
writeLines(readLines('nngp_latent.stan'))
```

```
##    /* Latent NNGP model*/
##
##    functions{
##        real nngp_w_lpdf(vector w, real sigmasq, real phi, matrix NN_dist,
##                         matrix NN_distM, int[,] NN_ind, int N, int M){
##
##            vector[N] V;
##            vector[N] I_Aw = w;
##            int dim;
##            int h;
##
##            for (i in 2:N) {
##
##                matrix[ i < (M + 1)? (i - 1) : M, i < (M + 1)? (i - 1): M]
##                iNNdistM;
##                matrix[ i < (M + 1)? (i - 1) : M, i < (M + 1)? (i - 1): M]
##                iNNCholL;
```

```
##               vector[ i < (M + 1)? (i - 1) : M] iNNcorr;
##               vector[ i < (M + 1)? (i - 1) : M] v;
##               row_vector[i < (M + 1)? (i - 1) : M] v2;
##
##               dim = (i < (M + 1))? (i - 1) : M;
##
##               if(dim == 1){iNNdistM[1, 1] = 1;}
##               else{
##                   h = 0;
##                   for (j in 1:(dim - 1)){
##                       for (k in (j + 1):dim){
##                           h = h + 1;
##                           iNNdistM[j, k] = exp(- phi * NN_distM[(i - 1), h]);
##                           iNNdistM[k, j] = iNNdistM[j, k];
##                       }
##                   }
##                   for(j in 1:dim){
##                       iNNdistM[j, j] = 1;
##                   }
##               }
##
##               iNNCholL = cholesky_decompose(iNNdistM);
##               iNNcorr = to_vector(exp(- phi * NN_dist[(i - 1), 1:dim]));
##
##               v = mdivide_left_tri_low(iNNCholL, iNNcorr);
##
##               V[i] = 1 - dot_self(v);
##
##               v2 = mdivide_right_tri_low(v', iNNCholL);
##
##               I_Aw[i] = I_Aw[i] - v2 * w[NN_ind[(i - 1), 1:dim]];
##
##           }
##           V[1] = 1;
##           return - 0.5 * ( 1 / sigmasq * dot_product(I_Aw, (I_Aw ./ V)) +
##                           sum(log(V)) + N * log(sigmasq));
##       }
##   }
##
##
##   data {
##       int<lower=1> N;
##       int<lower=1> M;
##       int<lower=1> P;
##       vector[N] Y;
##       matrix[N, P + 1] X;
##       int NN_ind[N - 1, M];
##       matrix[N - 1, M] NN_dist;
##       matrix[N - 1, (M * (M - 1) / 2)] NN_distM;
##       vector[P + 1] uB;
##       matrix[P + 1, P + 1] VB;
##       real ss;
##       real st;
##       real ap;
##       real bp;
##   }
##
##   transformed data {
##       cholesky_factor_cov[P + 1] L_VB;
##       L_VB = cholesky_decompose(VB);
##   }
##
##   parameters{
##       vector[P + 1] beta;
##       real<lower = 0> sigma;
##       real<lower = 0> tau;
##       real<lower = 0> phi;
##       vector[N] w;
##   }
##
##   transformed parameters {
##       real sigmasq = square(sigma);
##       real tausq = square(tau);
```

```
##   }
##
##   model{
##       beta ~ multi_normal_cholesky(uB, L_VB);
##       phi ~ gamma(ap, bp);
##       sigma ~ normal(0, ss);
##       tau ~ normal(0, st);
##       w ~ nngp_w(sigmasq, phi, NN_dist, NN_distM, NN_ind, N, M);
##       Y ~ normal(X * beta + w, tau);
##   }
```

```
writeLines(readLines('mycode.R'))
```

```
## rm(list = ls())
## library(rstan)
## library(spNNGP)
## library(spatstat)
## library(fields)
## library(gstat)
## library(spdep)
## set.seed(1234)
## source("NNmatrix.R")
##
## rmvn <- function(N, mu = 0, V = matrix(1)){
##   P <- length(mu)
##   D <- chol(V)
##   t(matrix(rnorm(N * P), ncol = P) %*% D + rep(mu, rep(N, P)))
## }
##
## N <- 300
## coords <- cbind(runif(N), runif(N))
## X <- as.matrix(cbind(1, rnorm(N)))
## B <- as.matrix(c(1, 5))
## sigma.sq <- 2
## tau.sq <- 0.1
## phi <- 3 / 0.5
##
## D <- as.matrix(dist(coords))
## R <- exp(- phi * D)
## w <- rmvn(1, rep(0, N), sigma.sq * R)
## Y <- rnorm(N, X %*% B + w, sqrt(tau.sq))
##
## #---------------------- Build neighbor index by NNMatrix ----------------------#
## M = 3                 # Number of Nearest Neighbors
## NN.matrix <- NNMatrix(coords = coords, n.neighbors = M, n.omp.threads = 2)
##
## #-------------------------- Set parameters of priors --------------------------#
## P = 1                  # number of regression coefficients
## uB = rep(0, P + 1)     # mean vector in the Gaussian prior of beta
## VB = diag(P + 1)*1000  # covariance matrix in the Gaussian prior of beta
## ss = 3 * sqrt(2)       # scale parameter in the normal prior of sigma
## st = 3 * sqrt(0.1)     # scale parameter in the normal prior of tau
## ap = 3; bp = 0.5       # shape and rate parameters in the Gamma prior of phi
##
## #------------------------------ response NNGP -------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N, M = M, P = P,
##              Y = Y[NN.matrix$ord], X = X[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12),
##                list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5),
##                list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi")
## samples <- stan(
##   file = "nngp_response.stan",
##   data = data,
##   init = myinits,
```

```
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples)
## stan_trace(samples, inc_warmup = T)
## stan_dens(samples, inc_warmup = T)
## stan_ac(samples, inc_warmup = T)
##
##
## #------------------------------ latent NNGP ---------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N, M = M, P = P,
##              Y = Y[NN.matrix$ord], X = X[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12,
##                     w_b1 = rep(0, N)),
##                list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5,
##                     w_b1 = rep(0.1, N)),
##                list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9 ,
##                     w_b1 = rep(0, N)))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi", "w")
## samples_w <- stan(
##   file = "nngp_latent.stan",
##   data = data,
##   init = myinits,
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples_w,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                            "w[2]", "w[3]", "w[4]"))
## stan_trace(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                               "w[2]", "w[3]", "w[4]"))
## stan_dens(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                              "w[2]", "w[3]", "w[4]"))
## stan_ac(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                            "w[2]", "w[3]", "w[4]"))
##
##
##
## #------------------------------ data2 ---------------------------------#
##
## data1 <- read.csv("LDNSuicides.csv")
## head(data1)
## Y1 <- data1$y
## X1 <- cbind(data1$x1,data1$x2)
## N1 <- length(Y1)
##
## #---------------------- Build neighbor index by NNMatrix ----------------------#
## M = 3                 # Number of Nearest Neighbors
## NN.matrix <- NNMatrix(coords = X1, n.neighbors = M, n.omp.threads = 2)
##
##
## #------------------------------ response NNGP ---------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N1, M = M, P = P,
##              Y = Y1[NN.matrix$ord], X = X1[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12),
```

```
##               list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5),
##               list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi")
## samples1 <- stan(
##   file = "nngp_response.stan",
##   data = data,
##   init = myinits,
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples1)
## stan_trace(samples1, inc_warmup = T)
## stan_dens(samples1, inc_warmup = T)
## stan_ac(samples1, inc_warmup = T)
##
##
##
##
## #------------------------------ latent NNGP ---------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N1, M = M, P = P,
##              Y = Y1[NN.matrix$ord], X = X1[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12,
##                     w_b1 = rep(0, N)),
##                list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5,
##                     w_b1 = rep(0.1, N)),
##                list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9 ,
##                     w_b1 = rep(0, N)))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi", "w")
## samples_w1 <- stan(
##   file = "nngp_latent.stan",
##   data = data,
##   init = myinits,
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples_w1,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                             "w[2]", "w[3]", "w[4]"))
## stan_trace(samples_w1, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                                "w[2]", "w[3]", "w[4]"))
## stan_dens(samples_w1, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                               "w[2]", "w[3]", "w[4]"))
## stan_ac(samples_w1, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                             "w[2]", "w[3]", "w[4]"))
##
##
##
## #------------------------------ data3 ---------------------------------#
##
##
## data(meuse)
## summary(meuse)
##
##
## X3 <- cbind(meuse$x,meuse$y)
## Y3 <- meuse$copper
## N3 <- length(Y3)
##
## #---------------------- Build neighbor index by NNMatrix ----------------------#
```

```
## #                        Build neighbor index by NNMatrix                          #
## M = 3                 # Number of Nearest Neighbors
## NN.matrix <- NNMatrix(coords = X3, n.neighbors = M, n.omp.threads = 2)
##
## #------------------------------ response NNGP ---------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N3, M = M, P = P,
##              Y = Y3[NN.matrix$ord], X = X3[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12),
##                list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5),
##                list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi")
## samples <- stan(
##   file = "nngp_response.stan",
##   data = data,
##   init = myinits,
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples)
## stan_trace(samples, inc_warmup = T)
## stan_dens(samples, inc_warmup = T)
## stan_ac(samples, inc_warmup = T)
##
##
## #------------------------------ latent NNGP ---------------------------------#
## options(mc.cores = parallel::detectCores())
## data <- list(N = N3, M = M, P = P,
##              Y = Y3[NN.matrix$ord], X = X3[NN.matrix$ord, ],
##              NN_ind = NN.matrix$NN_ind, NN_dist = NN.matrix$NN_dist,
##              NN_distM = NN.matrix$NN_distM,
##              uB = uB, VB = VB, ss = ss, st = st, ap = ap, bp = bp)
##
## myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5, phi = 12,
##                     w_b1 = rep(0, N)),
##                list(beta = c(5, 5), sigma = 1.5, tau = 0.2, phi = 5,
##                     w_b1 = rep(0.1, N)),
##                list(beta = c(0, 0), sigma = 2.5, tau = 0.1, phi = 9 ,
##                     w_b1 = rep(0, N)))
##
## parameters <- c("beta", "sigmasq", "tausq", "phi", "w")
## samples_w <- stan(
##   file = "nngp_latent.stan",
##   data = data,
##   init = myinits,
##   pars = parameters,
##   iter = 3000,
##   chains = 3,
##   thin = 5,
##   seed = 123
## )
##
## summary(samples_w,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                            "w[2]", "w[3]", "w[4]"))
## stan_trace(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                               "w[2]", "w[3]", "w[4]"))
## stan_dens(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                              "w[2]", "w[3]", "w[4]"))
## stan_ac(samples_w, inc_warmup = T,pars = c("beta", "sigmasq", "tausq", "phi", "w[1]",
##                                            "w[2]", "w[3]", "w[4]"))
```