HW6

Jinhong Du - 12243476

May 29, 2020

Contents

Problem A: Spatial Gaussian Processes

Consider the data from http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.0030339, available at https://github.com/stephens999/hgen48600/tree/master/data/CCR5 which you can read into R using code from https://stephens999.github.io/hgen48600/ccr5.html

These consist of latitude, longitude, and an allele frequency at each location. We will model these data as a Gaussian process. Since allele frequency lies in $[0,1]$ start by using the transformation $x = \log\left(\frac{\hat{f}}{1-\hat{f}}\right)$ (Here $\hat{f}$ is the estimated frequency in the code above.) We will let y denote locations in space (latitude, longitude) and $x(\cdot)$ denote the logit-transformed allele frequency varying as a function of space, so $x(y)$ is the logit-transformed allele frequency at location y. We will model $x(\cdot)$ as a Gaussian process, with constant mean $\mu = m$ and squared exponential covariance function of the form $a_1 e^{-\left(\frac{d}{a_2}\right)^2}$. Hence,$a = (a_1, a_2)$ and the mean m are the parameters to be estimated. Note that a_1 and a_2 must both be positive.

1 Write a function to compute the covariance matrix for $x^{\text{obs}} := (x(y_1), ..., x(y_r))$ given a value of a. Here $y_1, \ldots, y_r$ are the locations at which you have observations in the dataset. Try a few values of aa and check that the resulting covariance matrix is valid - that is, it is positive semi-definite. (The best way to check that a covariance is positive semi-definite is to attempt to perform a cholesky decomposition: if the decomposition succeeds then the matrix must be PSD).

[1]:
```
ccr5 = read.table("CCR5.freq.txt",header=TRUE)
ccr5[,1] = ifelse(ccr5[,1]>180,ccr5[,1]-360,ccr5[,1]) # changes longitudes>180␣
 ↪to negative

ccr5$count = round(ccr5$Freq* ccr5$SampleSize * 2)
ccr5$fhat = (ccr5$count+1)/(ccr5$SampleSize*2+2)
ccr5$x = log(ccr5$fhat/(1-ccr5$fhat))

library(geosphere)
comp_cov <- function(y, a){
    d <- geosphere::distm(y)/1000^2
    Sigma <- a[1] * exp(-(d/a[2])^2)
    return(Sigma)
}

chol(comp_cov(ccr5[,1:2], c(1,2)))
chol(comp_cov(ccr5[,1:2], c(0.5,1)))
chol(comp_cov(ccr5[,1:2], c(0.1,2)))
```

2 Write a function to compute the log-likelihood for the data x^{obs}, given a,ma,m. [Here we assume the mean is constant across the whole region, so mm is the same at every location]. The model here is that $x^{\text{obs}}|m, a \sim N_r(\mu, \Sigma)$ where $\Sigma = \Sigma(a)$ is the function of aa that you coded above and $\mu =$`rep(m,r)`. So your likelihood just involves computing a multivariate normal density. You can use the `R` function `mvtnorm::dmvnorm` (with `log=TRUE`).

[2]:
```
library(mvtnorm)
log_likelihood <- function(par, x, y){
    mu <- rep(par[1], length(x))
    a <- exp(par[2:3])
    Sigma <- comp_cov(y, a)
    return(mvtnorm::dmvnorm(x, mu, sigma=Sigma, log=TRUE))
}
```

3 Try using the `R` function optim (or another approach if you prefer) to optimize the likelihood numerically over a, m. (I found it seemed to work OK, in that it gave similar answers from different starting points).

[3]:
```
mle <- function(x, y, par_init){
    res <- optim(par=par_init, fn=log_likelihood,
                 method="L-BFGS-B", control=list(fnscale=-1),
                 x=x, y=y
                )
    m <- res$par[1]
    a <- exp(res$par[2:3])
    return(list(m=m,a=a))
}
res <- mle(ccr5$x, ccr5[,1:2], c(1,0,0))
res
```

$m -2.72024516675056

$a 1. 0.54951778649188 2. 0.146805066427651

4 Now we are going to try deleting each of the observed data points in turn and "impute" its value using our model. This process is sometimes known as Kriging.

i) Let $X = (X_1, \ldots, X_r)$ be r-variate normal with mean μ and variance covariance Σ. Write a function to compute the conditional expectation of X_1 given $X_2, \ldots, X_r$. [This is an application of standard results for the conditional mean of a Gaussian from, e.g. https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Conditional_distributions]

[4]:
```
comp_cond_mean <- function(x, mu, Sigma, i){
    r <- length(x)
    return(mu[i] + Sigma[i,-i] %*% solve(Sigma[-i,-i]) %*% (x[-i]-mu[-i]))
}
```

ii) Apply this function to compute $\mathrm{E}(x(y_1)|x(y_2), ..., x(y_r))$. Notice that this expectation ends up being a weighted linear combination of the other datapoints. Intuitively, if allele frequencies vary smoothly in space then this weighted linear combination should weight the nearby data points more. Does it?

[5]:
```
r <- length(ccr5$x)
comp_cond_mean(ccr5$x, rep(res$m, r), comp_cov(ccr5[,1:2], res$a), 1)
```

A matrix: 1×1 of type dbl -2.689532

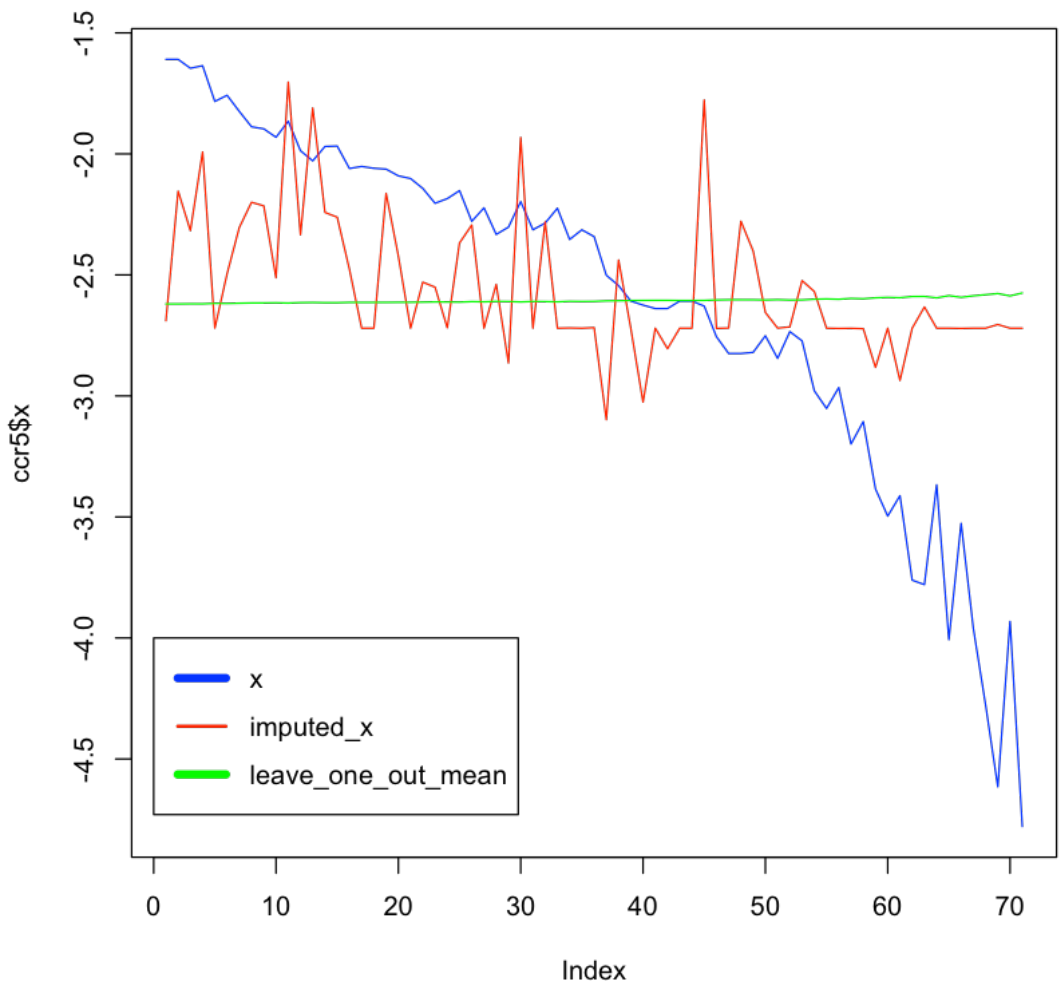
iii) Repeat this for each of the r datapoints.

[6]:
```
cond_mean <- c()
for(i in c(1:r)){
    cond_mean[i] <- comp_cond_mean(ccr5$x, rep(res$m, r), comp_cov(ccr5[,1:2],␣
 ↪res$a), i)
}
```

iv) How does the accuracy of this imputation scheme compare with just using the mean of the other datapoints to impute each datapoint?

[7]:
```
loo_mean <- c()
for(i in c(1:r)){
    loo_mean[i] <- mean(ccr5$x[-i])
}

plot(ccr5$x, type='l', col='blue')
lines(cond_mean, type='l', col='red')
lines(loo_mean, type='l', col='green')
legend(0,-4,c("x","imputed_x",'leave_one_out_mean'), lwd=c(5,2),␣
 ↪col=c("blue","red","green"), y.intersp=1.5)
```



[8]:
```
cat(mean((ccr5$x-cond_mean)^2))
cat(mean((ccr5$x-loo_mean)^2))
```

```
0.42822090.5461608
```

From the above plot and calculated MSE, the accuracy of this imputation scheme is better the just using the mean of other datapoints to impute each datapoint.