Augmented Feature Matrix Approach

Marshall Honaker

12/3/2020

First, let's import the necessary packages:

```r
library(fields)
library(RandomFields)
library(mvtnorm)
library(ggplot2)
library(matrixcalc)
library(gstat)
library(GpGp)
```

So this is my attempt to code up Dr. Sang's suggestions from the last email we sent her. Her main suggestion was that modifying the structure of the likelihood function (like we tried to do using the spatial Mahanobis distance) in a valid way would be exceptionally difficult. Instead, a more feasible approach might be to simply augment the feature matrix by appending "a set of spatial basis functions such as splines or bisques." (Bisques seem to be a fairly recent development, so while it would be really cool to use them, for the sake of getting the assignment turned in on time I think it might be best to stick with the simpler spline functions for the time being. If we decide to try to publish this, we can explore bisque functions a little more and add in a feature to let the user choose which will be used when generating the augmented feature matrix.)

Dr. Sang's recommendations were given as follows:

> The model is Y=quantile regression (X_beta+ X_basis*beta_spline) We can then use the current framework, but now with (p+# of basis) estimating equations, and the working weight matrix is calculated using X=(covariates, spatial basis covariates).

Dr. Sang also provided some sample code that can be used to generate the desired basis:

```r
library(FRK)
n <- 500
coords <- cbind(runif(n), runif(n))
G <- auto_basis(manifold = plane(), data = coords, regular = 0, max_basis=20,
           nres = 2, type = "bisquare")
```

```
## ...Automatically choosing number of functions...
```

```
## Loading required namespace: INLA
```

```r
## this is the design matrix corresponding to 20 basis functions
X_basis <- eval_basis(G, coords)
# matrix(X_basis). you can convert sparse basis design matrix to regular matrix if needed
dim(X_basis)
```

```
## [1] 500  21
```

As can be seen above, this code generates an n x s matrix, where s is the number of total splines. I think that each column corresponds to one of the bases and that the additional column(s) are additional smoothing components, such as the truncated power basis function. (See ISLR pgs. 273-274.)

With this, we can generate the `X_basis` in Dr. Sang's suggestion. However, I'm a little uncertain what she means by `beta_spline`. I think she's referring to the coefficient estimates from the regression spline using `X_basis` as the the feature matrix (that is, the beta from the spline), but this very well could be wrong. (Discuss as needed, confirm with Dr. Sang if still not sure.)

However, in case this is in fact what she's talking about (and to see how it works if it is) let's code up a demo here.

```
response <- rnorm(n)
X <- cbind(rnorm(n), rnorm(n))
beta_spline <- as.matrix(coef(lm(response ~ as.matrix(X_basis)))[-1])
augmented_column <- as.matrix(X_basis)%*%beta_spline
dim(augmented_column)
```

```
## [1] 500   1
```

```
dim(X)
```

```
## [1] 500   2
```

```
augmented_test_mat <- cbind(X, augmented_column)
dim(augmented_test_mat)
```

```
## [1] 500   3
```

At this point, we have generated an augmented feature matrix, but is it correct? If we assume that `beta_spline` does in fact refer to the coefficients from the regression spline, then the result of multiplying `X-basis` (n x s) by `beta_spline` (s x 1) (like Sang mentions in her suggestion) results in an n x 1 column vector. While it's easy enough to append this to our existing feature matrix, does appending a single column actually help us account for the spatial variability in our data? I'm not convinced that it does. (More on this to come.)

For now though, let's type it all up and see if it works. First, let's generalize this and put it into a function that will return the augmented feature matrix. (As we have defined it so far.)

```
augmented_feature_mat <- function(y, X, locs){
  G <- auto_basis(manifold = plane(), data = locs, regular = 0, max_basis=20,
                  nres = 2, type = "bisquare")
  spatial_basis <- as.matrix(eval_basis(G, locs))
  beta_spline <- coef(lm(y ~ spatial_basis))[-1]  ## See item (1) below
  return(cbind(X, spatial_basis%*%beta_spline)[,-1]) ## See item (6) below
}
```

Now, let's copy and paste all of the functions from the `Method.rmd` file over so we can run them here and see how they interact with our newly defined augmented feature matrix.

```
get_score<- function(y, X, tau, beta){  ## See item (2) below
  n <- nrow(X)
  temp <- t(X[1,,drop=FALSE])*ifelse(y[1] - X[1,,drop=FALSE]%*%as.matrix(beta) < 0, tau  - 1, tau)[1,1]
  for(i in 2:n){
    temp <- temp + t(X[i,,drop=FALSE])*ifelse(y[i] - X[i,,drop=FALSE]%*%as.matrix(beta) < 0, tau  - 1,
  }
  return(temp)
}

weight <- function(X, tau){  ## See item (4) below
  n <- nrow(X)
  temp <- X[1,]%*%t(X[1,])
  for(i in 2:n){
```

```r
    temp <- temp + X[i,]%*%t(X[i,])
  }
  coef <- n/(tau*(1 - tau))
  return(round(coef*solve(temp), 7))
}

get_likelihood <- function(y, X, tau, beta, locs, C){ ## See item (5) below
  score <- get_score(y, X, tau, beta)
  coef <- -1/(2*length(y))
  weight_X <- weight(X, tau)
  kernel <- exp(coef*(t(score)%*%weight_X%*%score)[1,1])
  return(C*kernel)
}
```

A few comments on the methods above:

1. We need to make sure that Dr. Sang is actually talking about using the coefficients from the regression splines using the simulated bases as the feature matrix. (I think she is, because then `X_basis%*%beta_spline` would be the estimated values from that regression.)
2. The `get_score()` function outputs a p length vector whose entries correspond to the features in the feature matrix. The last element in this vector (corresponding to the additional column we get by appending `X_basis%*%beta_spline` to out original feature matrix) is much larger than the others, so we should keep an eye on this.
3. I deleted the `get_spatial_covar_mat()` function because we no longer need it.
4. The paper specifies that the weight matrix W must be positive definite. Even though we followed the definition for W given in the paper to a tee, our function doesn't return a positive definite matrix if you consider the eighth decimal place of each entry. Since the eighth decimal isn't especially significant in the grand scheme of things, I added the `round(., 7)` method to make sure that the method returns a weight matrix that satisfies the positive definiteness requirement.
5. I have no idea how to come up with good values of C in the likelihood function. I've been using $C = 1$ so far just so the code will run and play nice. I need to read through the paper some more to figure out how they did it.
6. The `augmented_feature_mat()` method was returning a feature matrix with the column of 1s for the intercept. `R` will include this automatically, so I modified `return()` statement so the output would not include this column.

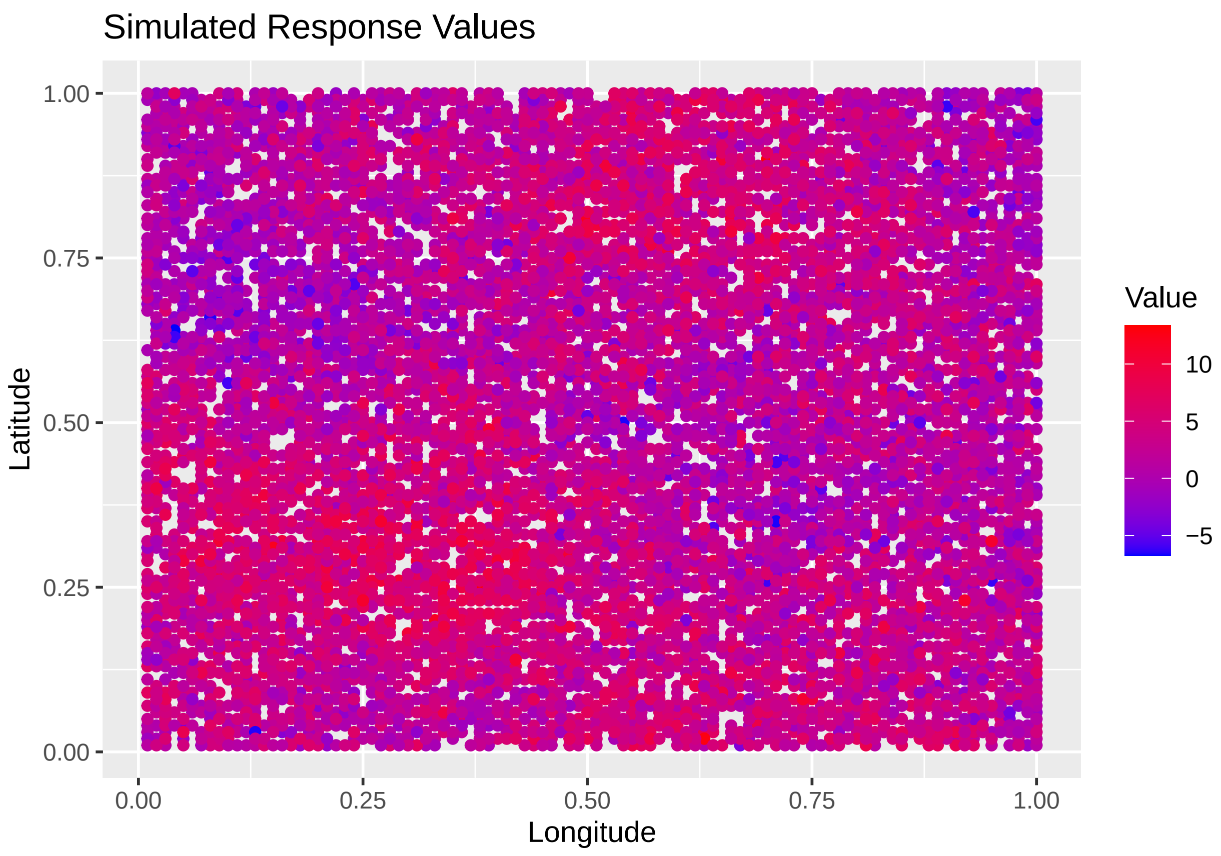
Next, let's look at the methods for the Importance Sampling algorithm for a single quantile.

```r
get_updated_params <- function(y, X, tau, beta, Sigma, locs, draw){
  n <- length(y)
  p <- ncol(X)

  w <- get_likelihood(y, X, tau, beta, locs, 1)*(1/(2*n)^p)/dmvnorm(draw, beta, Sigma)

  mu_hat <- apply(w*draw, 2, sum)/sum(w)

  S <- matrix(nrow = p, ncol = p)
  for(i in 1:p){
    for(j in 1:p){
      S[i,j] <- sum(w*draw[,i]*draw[,j])/sum(w) - mu_hat[i]*mu_hat[j]
    }
  }
  temp <- list(mu_hat, S)
  names(temp) <- c("mu", "S")
  return(temp)
}

adaIS_singleQuantile <- function(y, X, tau, C, locs, M, num_reps){
  p <- ncol(X) ## Step 1: Initialize starting values for IS algorithm
  n <- length(y) ## Step 1: Initialize starting values for IS algorithm
  beta <- coef(lm(y ~ X))[-1] ## Step 1: Initialize starting values for IS algorithm
  S <- cov(X) ## Step 1: Initialize starting values for IS algorithm

  params <- list(beta, S)  ## Put our initial parameter estimates/starting values into a list called pa
  names(params) <- c("mu", "S")

  for(i in 1:num_reps){ ## Step 4: Repeat steps 2 and 3 until we achieve the desired effective sample s
    draws <- rmvnorm(M, params$mu, params$S)  ## Step 2: Simulate M values from the proposal distributi
    params <- get_updated_params(y, X, tau, params$mu, params$S, locs, draws) ## Step 3: Update the par
  }
  return(params)
}
```

Next, let's simulate some data so we can test what we have so far.

First, let's generate and plot the simulated data. (This simulation technique was adapted from Dr. Sang's demo code. Lecture 3, Gaussian Processes; Big Data Demo)

```r
locs <- as.matrix(expand.grid((1:100)/100, (1:100)/100))
sigma_sq <- 4
phi <- .4
nu <- .5
tau_sq <- .5

w <- fast_Gp_sim(c(sigma_sq, phi, nu, 0), "matern_isotropic", locs, 30)

n <- nrow(locs)
X <- as.matrix(cbind(rep(1, n), rnorm(n), rnorm(n)))
beta <- as.matrix(c(1, 1, 2))
p <- length(beta)
y <- rnorm(n, X%*%beta + w, sqrt(tau_sq))

training_indices <- sample(1:n, round(.75*n), replace = FALSE)

y_train <- y[training_indices]
X_train <- X[training_indices,]
locs_train <- locs[training_indices,]

y_test <- y[-training_indices]
X_test <- X[-training_indices,]
locs_test <- locs[-training_indices,]

training_data <- as.data.frame(cbind(y_train, X_train[,-1], locs_train))
names(training_data) <- c("y_train", "X1_train", "X2_train", "long_train", "lat_train")

ggplot() + geom_point(data = training_data, aes(x = long_train, y = lat_train, col = y_train)) +
  scale_colour_gradient(low = "blue", high = "red") +
  labs(title = "Simulated Response Values",
       x = "Longitude",
       y = "Latitude",
       col = "Value")
```



Next, let's test out our more basic methods.

```
tau <- .5

X_augmented <- augmented_feature_mat(y_train, X_train, locs_train)
```

```
## ...Automatically choosing number of functions...
```

```
X_augmented[1:10,]
```

```
##             [,1]        [,2]       [,3]
##  [1,] -0.2951886 -1.84573528 -3.2376843
##  [2,]  0.5294462  0.75483226 -3.8447474
##  [3,]  0.4809664  1.40690509  1.3119153
##  [4,]  1.3082273 -0.33937647 -2.6898350
##  [5,] -0.5831771 -1.81335621  0.5132510
##  [6,]  1.6055861 -0.39799866 -1.9114532
##  [7,]  1.3588235 -0.06790214 -1.7802968
##  [8,]  0.8978775 -0.66906593 -0.5084916
##  [9,] -0.0170825  0.59427167  0.4433685
## [10,] -1.3567701  1.23654787 -1.1200322
```

```
beta <- coef(lm(y_train ~ X_augmented))[-1]
score <- get_score(y_train, X_augmented, tau, beta)
score
```

```
##             [,1]
## [1,]   -37.43445
## [2,]   -35.43052
## [3,] -3982.65542
```

```r
weight_mat <- weight(X_augmented, tau)
weight_mat
```

```
##             [,1]       [,2]       [,3]
## [1,]  3.9371853  0.0006042 -0.0178889
## [2,]  0.0006042  3.9650138 -0.0525617
## [3,] -0.0178889 -0.0525617  1.2781729
```

```r
is.positive.definite(weight_mat)
```

```
## [1] TRUE
```

```r
lik <- get_likelihood(y_train, X_augmented, tau, beta, locs_train, 1)
lik
```

```
## [1] 0
```

As can be seen above, the code for the more basic methods runs. However, we still run into the same issues we discussed earlier.

Now, let's move on to the Importance Sampling methods.

```r
Sigma <- cov(X_augmented)
draw <- rmvnorm(1, beta, Sigma)
pars <- get_updated_params(y_train, X_augmented, tau, beta, Sigma, locs_train, draw)
pars
```

```
## $mu
## X_augmented1 X_augmented2 X_augmented3
##          NaN          NaN          NaN
##
## $S
##      [,1] [,2] [,3]
## [1,]  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN
```

```r
## post_dist_params <- adaIS_singleQuantile(y_train, X_augmented, tau, 1, locs_train, 5000, 10)
## get_updated_params() isn't working, so we can't expect adaIS_singleQuantile() to work either
```

Because this is a Bayesian regression model, we will need to derive the posterior predictive distribution to make predictions at the test locations using the test data. Once the Importance Sampling methods work and give us proper values for the parameters of the posterior distribution, we can make the desired predictions to evaluate the predictive performance of our model.