Augmented Feature Matrix Approach

Marshall Honaker

12/3/2020

First, let's import the necessary packages:

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

Modified function to return augmented feature matrix as discussed with Dr. Sang.

```
augmented_feature_mat <- function(X, locs, knots, phi, sigma_sq){
  C_knots <- sigma_sq*exp(-rdist(knots, knots)/phi)
  half_inv_C_knots <- chol(solve(C_knots))
  basis_X <- sigma_sq*exp(-rdist(locs, knots)/phi)%*%half_inv_C_knots
  return(cbind(X, basis_X)[,-1])
}
```

Now, let's move on to the more rudimentary functions

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

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. 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 the `return()` statement so the output would not include this column.
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 columns we get by appending `X_basis` 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'm not sure 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.

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

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

  is_weights <- numeric(nrow(draws))
  for(i in 1:nrow(draws)){
    is_weights[i] <- get_likelihood(y, X, tau, draws[i,], locs, 1)*(1/(2*n)^p)/dmvnorm(draws[i,], beta,
  }

  mu_hat <- apply(w*draws, 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*draws[,i]*draws[,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

  a_1 <- coef(lm(y ~ X))[-1]
  a_0 <- quantile(y - X%*%a_1, tau)
```

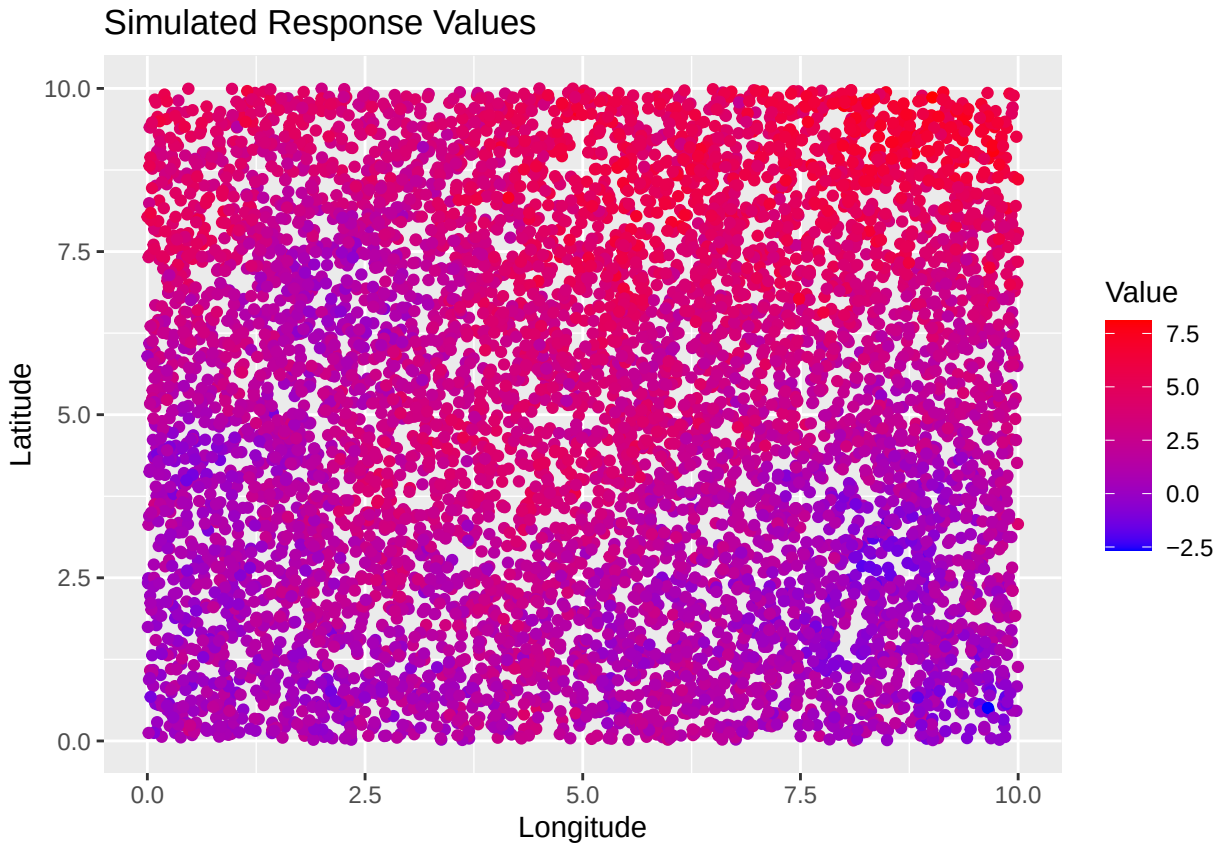
```
  a <- c(a_0, a_1)
  X <- cbind(rep(1, n), X)
  temp <- X[1,]%*%t(X[1,])
  for(i in 2:n){
    temp <- temp + X[i,]%*%t(X[i,])
  }
  S_0 <- 1*tau*(1 - tau)*solve((1/n)*temp)

  params <- list(a, S_0)
  names(params) <- c("mu", "S")
  draw <- rmvnorm(M, params$mu, params$S)

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

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

```
n <- 10000
locs <- as.matrix(cbind(runif(n, 0, 10), runif(n, 0, 10)))
sigma_sq <- 2
phi <- 5
nu <- .5
tau_sq <- .5

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

X <- as.matrix(cbind(rep(1, n), locs[,1]/10, locs[,2]/10))
beta <- as.matrix(c(1, 2, 3))
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(X_train, locs_train, expand.grid(seq(0.1,9.9,length=5),seq(0.1,9.9
## X_augmented[1:10,]

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

weight_mat <- weight(X_augmented, tau)
## weight_mat

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

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.

```
Sigma <- cov(X_augmented)
draw <- rmvnorm(1000, beta, Sigma)
```

```
## pars <- get_updated_params(y_train, X_augmented, tau, beta, Sigma, locs_train, draw)
## pars

## 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.