Spatial Leave One Out

Daivd Leslie

Wednesday, March 25, 2015

The purpose of the code is to evaluate the effectiveness of the spatial leave one out method for determining the predictive power of each model given. In order to acomplish this, the model given will be trained with data that has been reduced by **one sample**. **The sample that** is left out will then be used as a new data point to be predicted by the given model. Let's first start by generating some spatial autocorrelated data.

```r
# Import Statement(s)
library(RandomFields)
# Create Grid
n = 100

# Set distance between points (Must be derived from data)
spat_range = 5

# Generate spatially autocorralated data
mod_spat_dep = RMexp(var=1, scale=spat_range)
spat_err = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)
x1 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)
x2 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)
x3 = RFsimulate(mod_spat_dep, x=1:n, y=1:n, grid=T)

# Plot data
plot(x1)

# Plot variogram
plot(RFempiricalvariogram(data=x1))

# Convert objects to vectors
spat_err = as.vector(spat_err)
x1 = as.vector(x1)
x2 = as.vector(x2)
x3 = as.vector(x3)
y = 2*x1 + 0*x2 + x3 + spat_err

# Create coordinates
coords = cbind(rep(x=1:n, times=n), rep(x=1:n,each=n))

# Create data.frame
dataset = data.frame(y, x1, x2, x3, coords)

head(dataset)
```

```
##             y         x1          x2          x3 X1 X2
## 1 -0.86506789  0.6417892  1.05016168 -0.48605175  1  1
## 2  0.08588092  0.3626813 -0.04223851 -0.12121414  2  1
## 3 -1.10959290 -0.7009892 -0.16837090 -0.03149576  3  1
## 4 -0.06688091 -0.5118189 -0.23516754  0.73161151  4  1
## 5  0.87737735 -0.2841958  0.24529860  0.81788235  5  1
```

```
## 6  3.07059185  0.7576685 -0.29256647  0.98019678  6  1
```

Now that we have our spatially autocorrelated data, lets take a sample of our data and verify that the results generated by the model are reasonalbe estimates of the true model's coeffcients (Remebering that they should be close to the exspected values of: x1=2, x2=1, and x3=3).

```r
# Create sample population
n = 2000
samp = sample(nrow(dataset), size=n)
# Generate responce: additive model plus spatial error
mods = list()
mods$x1234_mod = glm(y ~ x1 + x2 + x3, data = dataset[samp, ])

summary(mods$x1234_mod)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x3, data = dataset[samp, ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9724  -0.6423   0.0294   0.6368   3.8706
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03634    0.02215   1.641  0.10093
## x1           1.97755    0.02213  89.351  < 2e-16 ***
## x2          -0.05488    0.02063  -2.660  0.00787 **
## x3           1.07824    0.02153  50.072  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.8829136)
##
##     Null deviance: 11098.2  on 1999  degrees of freedom
## Residual deviance:  1762.3  on 1996  degrees of freedom
## AIC: 5432.7
##
## Number of Fisher Scoring iterations: 2
```

Looking at the model summary, it appears that calculated coeffeicents are fairly close to actual coefficents. Now that we feel cofident about the sample taken, lets see how good the models generated will be at predicting new data. In order to do this, lets apply the spatial leave one out method to the data.

```r
# Function(s)
get_training_rows = function(coords, dist_thres, longlat=FALSE) {
    ## Computing the distance matrix from the data:
    if (longlat) {
        require(sp)
        dist_matrix = spDists(coords, longlat=TRUE)
    }
    else
        dist_matrix = as.matrix(dist(coords))
```

```
    ## Initializing the row indices of the dataset to be used in training
    training_rows = list()
    ## Creating the sets of training indices
    for (i in 1:nrow(dist_matrix)) {
        # Keeping only the observations far enough of the i-st observation by
        # using the threshold distance
        num_cell = which(dist_matrix[i, ] > dist_thres)
        training_rows[[i]] = num_cell
    }
    return(training_rows)
}

sloo_simple = function(model,training){
  # Creating the response variable name
  y <- as.character(x=formula(x=model)[2])
  # Initializing the 'logLik' object
  logLik <- vector(mode="numeric",length=nrow(x=model$data))
    # Calculating the SLOO logLikelihoods for each observation i
    for(i in 1:nrow(x=model$data)){
        # Extracting the i-st training set:
       training_data <- model$data[training[[i]],]
        # Calculating the model parameters from the i-st training set:
        m <- glm(formula=formula(model),data=training_data,family= model$family)
        # Predicting the i-st observation from the i-st training set:
        m.pred <- predict(object=m,newdata=model$data[i,],type="response")
        # Calculating the probability of the i-st observed value according
    # to the predicted one by the i-st training set:
    logLik[i] <- dnorm(x=model$data[i,y],mean=m.pred,
                       sd=sqrt(sum(residuals(m)^2)/nrow(training_data)),
                       log=T)
  }
    # Calculating the overall SLOO logLikelihood:
    Sum.logLik <- sum(logLik)
    return(Sum.logLik)
}

training = get_training_rows(coords[samp, ], dist_thres = spat_range)

# evalutate the four models using log likelihood
# with the SLOO and a non-spatial approach
sloo_ll = sapply(mods, function(x) sloo_simple(x, training))
ll = sapply(mods, logLik)

sloo_ll
```

```
## x1234_mod
## -2737.774
```

```
ll
```

```
## x1234_mod
## -2711.347
```

```r
sort(sloo_ll, dec=T)
```

```
## x1234_mod
## -2737.774
```

```r
sort(ll, dec=T)
```

```
## x1234_mod
## -2711.347
```

```r
# FYI the relationship between ll and AIC is given by
# AIC = 2*k - 2*ll where k is the number of parameters in the model
2* c(2, 3, 3, 4) - 2*ll
```

```
## [1] 5426.694 5428.694 5428.694 5430.694
```

```r
sapply(mods, AIC)
```

```
## x1234_mod
##  5432.694
```