Spatial Regression

Houjie Wang

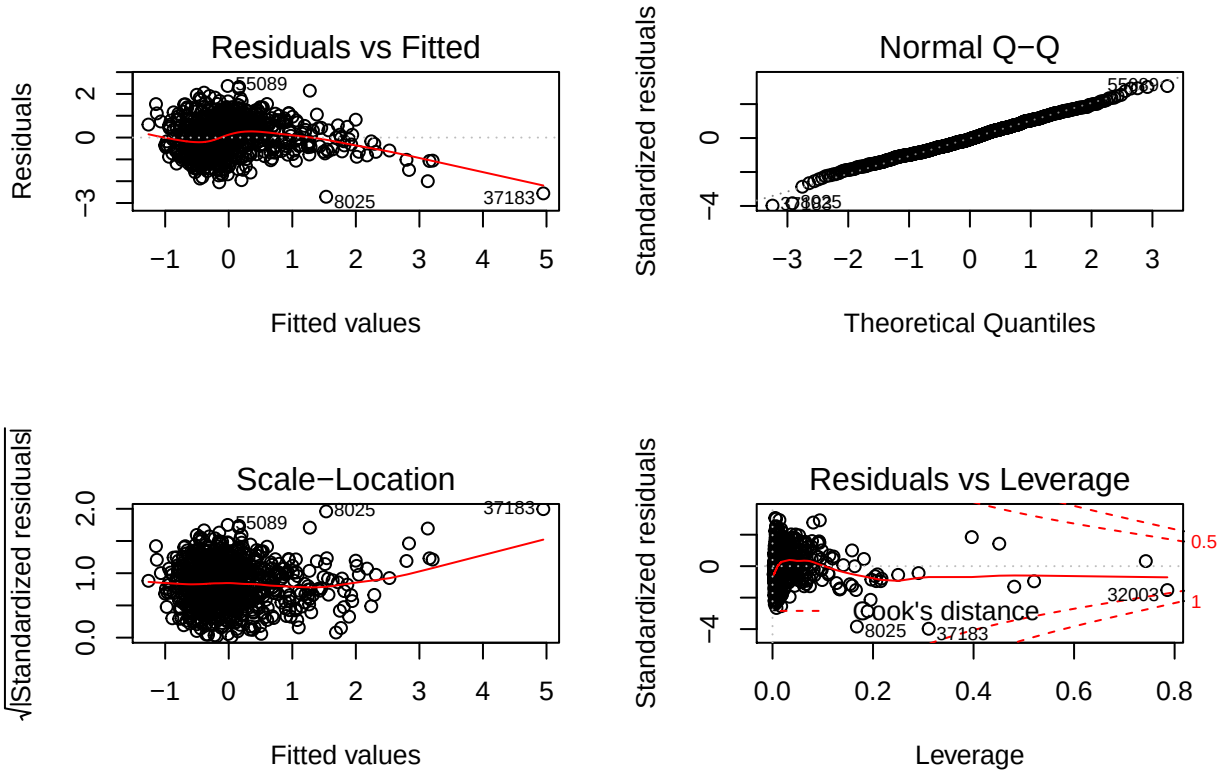
4/24/2020

```
library(igraph); library(spdep); library(spatialreg); library(car); library(glmnet); library(tseries);
load("Data.RData"); load("locs.Rdata")
rownames(Data) = locs$fips
```

1 Problem Identification

We first fit the entire data with basic linear regression, which is expected to be flawed.

```
fit = lm(Data$y~., data = Data)
# summary(fit)
par(mfrow = c(2,2))
plot(fit)[2]
```




```
## NULL
```

```
list("R_square" = 1-sum(residuals(fit)^2)/sum((Data$y-mean(Data$y))^2))
```

```
## $R_square
## [1] 0.4148382
```

```
shapiro.test(residuals(fit))
```

```
##
##  Shapiro-Wilk normality test
```

```
##
## data:  residuals(fit)
## W = 0.99717, p-value = 0.143
```

```
bgtest(fit)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
## data:  fit
## LM test = 40.233, df = 1, p-value = 2.254e-10
```

```
ncvTest(fit)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 5.739748, Df = 1, p = 0.016585
```

Based on the regression plots and hypothesis testing, the assumption of linearity, autocorrelation and constant variance are all violated. Moreover, the model is also poor in explaning the variation in data. All above facts implies that this crude regression is invalid, prompting the necessity of a new model that takes more information into accounts. Given the spatial nature of our data, we decide to employ spatial regression to resolve the autocorrelation and heteroscedasticity incurred by its spatial structure.

Spatial Regression

Before we further apply a spatial model, it's safe to apply Moran's I test (Dubé and Legros 2014) to justifiy the existence of spatial autocorrelation. Moran's I statistic of takes the form described in equation:

$$I = \frac{n}{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}} \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

, where n is the number of observations, y is the response and w_{ij} is the weight. In our scanerio, w_{ij} is a binary number from a weight matrix W, indicating if any two counties x_i and x_j share a boundary. Therefore, in matrix notation, we can write Moran's I as follows:

$$I = \frac{nY^{*'}WY^{*}}{s_o Y^{*'}Y^{*}}$$

, where W is the $n \times n$ sparse matrix and $s_o = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}$.

Note that Moran's I has asymptotic characteristic, which follows a standard normal for a large observations. ($\frac{I - E(I)}{\sqrt{Var(I)}} \sim N(0,1)$) This allows us to compute the p value easily.

Then we implement it in R:

```
# Generate the spatial structure
rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
G = graph.edgelist(as.matrix(rawData), directed=FALSE)
G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
subFIPSid = V(G)$name %in% locs$fips
countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
graph0 = permute(countyGraph, order(V(countyGraph)$name)); ### this is the dense graph to begin with.
nb.mat = as_adjacency_matrix(graph0)
nb.mat = nb.mat[as.vector(sapply(as.character(locs$fips), function(x){which(x == rownames(nb.mat))})), ]
nb.mat = nb.mat[, as.vector(sapply(as.character(locs$fips), function(x){which(x == colnames(nb.mat))}))]
nb.w = mat2listw(nb.mat, style='B')
```

```r
# Moran's I test
lm.morantest(fit, nb.w)
```

```
##
##  Global Moran I for regression residuals
##
## data:
## model: lm(formula = Data$y ~ ., data = Data)
## weights: nb.w
##
## Moran I statistic standard deviate = 21.037, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Observed Moran I      Expectation         Variance
##     0.4284145309    -0.0040465367     0.0004226136
```

The result of Moran's I test shows the presence of spatial autocorrelaiton. Thus, it would be safe and effective to analyze the data with spatial regresson models

Spatial Regression Models

Spatial Models

Thera are three basic spatial models accounting for the spatial dependence. (Le Gallo 2014) They are spatial lag model, spatial cross-regressive model and spatial error model.

Spatial Lag Model (SAR Model)

The spatial lag model takes form as follows:

$$Y = \rho WY + X\beta + \epsilon.$$

WY is the spatial lagged dependent variable; X are the explanatory variables and $\epsilon \sim N(0, \sigma_e^2)$ Similar to OLS, we are interested in estimating for the parameters ρ and β. We can also write a spatial lag model as follows:

$$Y = (I - \rho W)^{-1} X\beta + (I - \rho W)^{-1}\epsilon$$

However, since WY much correlate with the residuals (i.e. $cov(WY, \epsilon) \neq 0$), then WY is endogeneous and thus OLS would be inconsistent and biased due to the simultaneity bias)

Spatial Cross-regressive Model

Other than introducing an endogeneous variable, spatial cross-regressive intriduce a series of exogeneous spatial lag variables which are estimable direcly as OLS. It takes form as follows:

$$Y = X\beta + WX\gamma + \epsilon.$$

We can simply rewrite it as an ordinary OLS problem and easily estimate for the parameters:

$$Y = \begin{bmatrix} X & WY \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} + \epsilon = \tilde{X}\tilde{\beta} + \epsilon, \tilde{\beta} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'Y$$

Spatial Error Model

Contrary to spatial lag model, spatial error model contains a spatial autoregressive error term, which take form as follows:

$$Y = X\beta + \epsilon, \epsilon = \lambda W\epsilon + u,$$

where ϵ is a vector of spatially autocorrelated error terms and $u \sim N(0, \sigma_u^2)$. Similar to OLS, we are interested in estimating for the parameters λ and β.

Spatial Model Selection

Larange multipler test for spatial autocorrelation distincts between spatial lag model and error model including their robust forms, which prompts a proper alternative for us to resolve spatial issues.

```
lm.LMtests(fit, nb.w, test=c("LMerr","RLMerr","LMlag","RLMlag"))
```

```
## Warning in lm.LMtests(fit, nb.w, test = c("LMerr", "RLMerr", "LMlag",
## "RLMlag")): Spatial weights matrix not row standardized
```

```
##
##  Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = Data$y ~ ., data = Data)
## weights: nb.w
##
## LMerr = 426.73, df = 1, p-value < 2.2e-16
##
##
##  Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = Data$y ~ ., data = Data)
## weights: nb.w
##
## RLMerr = 8.88, df = 1, p-value = 0.002883
##
##
##  Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = Data$y ~ ., data = Data)
## weights: nb.w
##
## LMlag = 532.86, df = 1, p-value < 2.2e-16
##
##
##  Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = Data$y ~ ., data = Data)
## weights: nb.w
##
## RLMlag = 115.01, df = 1, p-value < 2.2e-16
```

Based on the results, note that both spatial error model and spatial lag model are significant in our case, (Their p-values are smaller than 0.05). So we would mainly focus on these two models.

Spatial Lag Model

The significant results will be displayed in the next section for a concise purpose

```
fit2 = lagsarlm(Data$y~., data = Data, nb.w)
```

We obtain a very significant ρ in this model (p is almost 0 by asymptotic t test), which means that the spatial coefficient ρ significantly improves the model from OLS. Moreover, we can also see an reduction in AIC compated to OLS . LM test for residual autocorrelation returns us a strong evidence that spatial autocorrelation is resolved (LM test $p-value = 0.61284$). These all indicate a stronger validity of the spatial lag model.

Spatial Error Model

```
fit3 = errorsarlm(Data$y~., data = Data, nb.w)
```

Similarly, we also obtain a strong evidence that the spatial parameter is functioning to enhance the model (p value for λ is zero) and a smaller AIC from OLS. Note that Lm test for spatial autocorrelation is not available.

Spatial Cross-regressive Model (SLX)

```
fit4 = lmSLX(Data$y~., data = Data, nb.w)
```

Summary

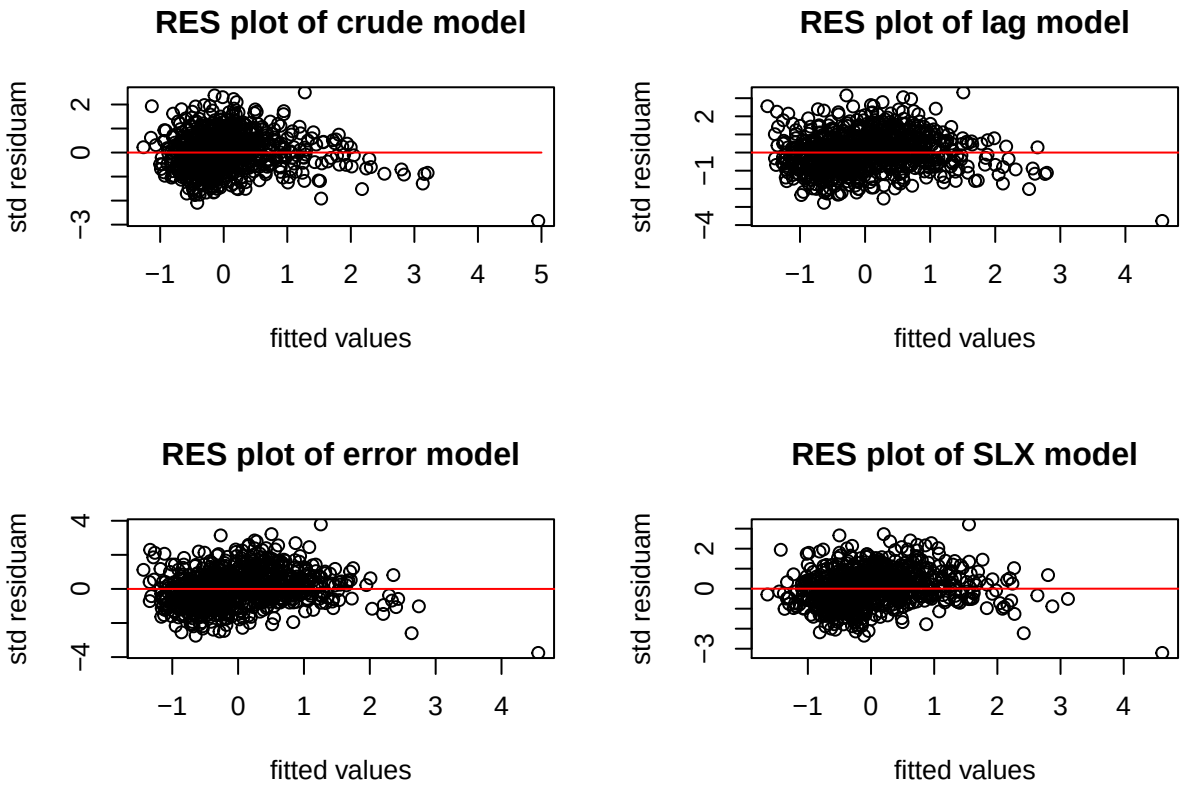
```
fit_AIC = c(AIC(fit), AIC(fit2), AIC(fit3), AIC(fit4))
R_sq = function(fit){
  return(1 - sum((residuals(fit) - mean(residuals(fit)))^2)/sum((Data$y - mean(Data$y))^2))
  }
fit_R_sq = c(R_sq(fit), R_sq(fit2), R_sq(fit3), R_sq(fit4))
fit_normality = c(shapiro.test(residuals(fit))$p.value, shapiro.test(residuals(fit2))$p.value, shapiro.
fit_auto = round(c(bgtest(fit)$p.value, 0.61284, NA, bgtest(fit4)$p.value), digits = 4)
fit_table = cbind.data.frame("AIC" = fit_AIC, "R^2" = fit_R_sq, "Normal" = fit_normality, "auto" = fit_
rownames(fit_table) = c("lm", "lag", "err", "SLX")
fit_table
```

```
##          AIC       R^2 Normal   auto
## lm  2006.312 0.4148382   TRUE 0.0000
## lag 1608.725 0.6669946   TRUE 0.6128
## err 1631.743 0.6692486   TRUE     NA
## SLX 1837.541 0.5441529   TRUE 0.0980
```

Based on the table, we conclude that under the fact that normality is preserved, by comparing the AICs and R^2, we can see all the spatial regression models are more accurate in explaning the variation within the data, especially the spatial lag model. In addition to the fitting performance, from the p values of the fourth column, we can see that lag model and SLX model to some extent resolve the presence of spatial regression ($p-value > 0.05$). Notwithstanding the lack in error models, the significace in its spatial parameter $p-value_\lambda = 0$ is enough to imply the improvement. To examine the heteroschedasticity of these models, due to a lack of a generic function for constant variance test compatiable to a variation of model object in R, we plot the fitted values verses standardized residuals.

```
par(mfrow = c(2,2))
plot(x = Data$y - residuals(fit),  y = (residuals(fit2)-mean(residuals(fit)))/sd(residuals(fit)), ylab =
lines(x = seq(-2, 5, length.out = 1000), y = rep(0, 1000), col = "red")
plot(x = Data$y - residuals(fit2),  y = (residuals(fit2)-mean(residuals(fit2)))/sd(residuals(fit2)), yla
lines(x = seq(-2, 5, length.out = 1000), y = rep(0, 1000), col = "red")
plot(x = Data$y - residuals(fit3),  y = (residuals(fit3)-mean(residuals(fit3)))/sd(residuals(fit3)), yla
```

```
lines(x = seq(-2, 5, length.out = 1000), y = rep(0, 1000), col = "red")
plot(x = Data$y - residuals(fit4),  y = (residuals(fit3)-mean(residuals(fit4)))/sd(residuals(fit4)), yla
lines(x = seq(-2, 5, length.out = 1000), y = rep(0, 1000), col = "red")
```



Although not obvious, the points in spatial model plot are more spread out and more symmetrically concentrated with respect to the central red line. Thus it's reasonable to conclude that spatial models, to some extent stablize the variance of residuals. Above all, spatial models, although not remarkably better, all out-performe the OLS in terms of the underlying assumptions of regression, significantly resolving the spatial dependence among the data.

Spatial Prediction

In this section we employ spatial lag model to make prediction and calculate for RMSE so that we are allowed to compare the lag model with other machine learning techniques.

```
rmse = c()
for (i in 1: 10){
  tst_ind = sample(1: nrow(Data), 213, replace = F)
  tst_data = Data[tst_ind,]; trn_data = Data[-tst_ind,]

  # Weight matrix for test data
  rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
  G = graph.edgelist(as.matrix(rawData), directed=FALSE)
  G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
  subFIPSid = V(G)$name %in% locs$fips[tst_ind]
  countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
  graph0 = permute(countyGraph, order(V(countyGraph)$name)); ### this is the dense graph to begin with.
  nb.mat_tst = as_adjacency_matrix(graph0)
  nb.mat_tst = nb.mat_tst[as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x == row
```

```
  nb.mat_tst = nb.mat_tst[, as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x == co
  ind_rm = -which(apply(nb.mat_tst, 1, function(x){sum(x) == 0}))
  if (length(ind_rm) != 0){
    nb.mat_tst = nb.mat_tst[ind_rm, ind_rm]
    tst_data = tst_data[ind_rm, ]
    nb.w_tst = mat2listw(nb.mat_tst)
  } else {
    nb.w_tst = mat2listw(nb.mat_tst)
  }
  rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
  G = graph.edgelist(as.matrix(rawData), directed=FALSE)
  G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
  subFIPSid = V(G)$name %in% locs$fips[-tst_ind]
  countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
  graph0 = permute(countyGraph, order(V(countyGraph)$name));
  nb.mat_trn = as_adjacency_matrix(graph0)
  nb.mat_trn = nb.mat_trn[as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x == rou
  nb.mat_trn = nb.mat_trn[, as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x == 
  ind_rm = -which(apply(nb.mat_trn, 1, function(x){sum(x) == 0}))
  if (length(ind_rm) != 0){
    nb.mat_trn = nb.mat_trn[ind_rm, ind_rm]
    trn_data = trn_data[ind_rm, ]
    nb.w_trn = mat2listw(nb.mat_trn)
  } else {
    nb.w_trn = mat2listw(nb.mat_trn)
  }
  fit = lagsarlm(trn_data$y~., data = trn_data, nb.w_trn)
  rho = as.vector(coef(fit)[1]); beta = as.vector(coef(fit)[-1])
  res = tst_data[, 1] - as.vector(solve(diag(1, nrow = nrow(nb.mat_tst)) - rho*nb.mat_tst) %*% as.matri
  rmse = c(rmse, sqrt(mean(res^2)))
}
RMSE = mean(rmse)
list("RMSE" = RMSE)
```

```
## $RMSE
## [1] 0.8084545
```

We obtain the averaged RMSE of 10 repeated 25% cross-validation for spatial lag model.

Power transformation of spatial weight matrix

Instead of using the weight matrix with binary entries indicating the proximity of two counties, we could replace the binary entries with the some power of distance so we could introduce a shrinking effect based on the distance. We evaluate this method over 30 grid points from 0.1 to 3. For each grid point of power, similar to the previous case, we do 25% cross validation 10 times and calculate the RMSE.

```
mean_rmse = c()
for (t in seq(0.1, 3, length.out = 30)){
  rmse = c()
  mse = c()
  for (i in 1: 10){
    tst_ind = sample(1: nrow(Data), 213, replace = F)
    tst_data = Data[tst_ind,]; trn_data = Data[-tst_ind,]

    # Weight matrix for test data
```

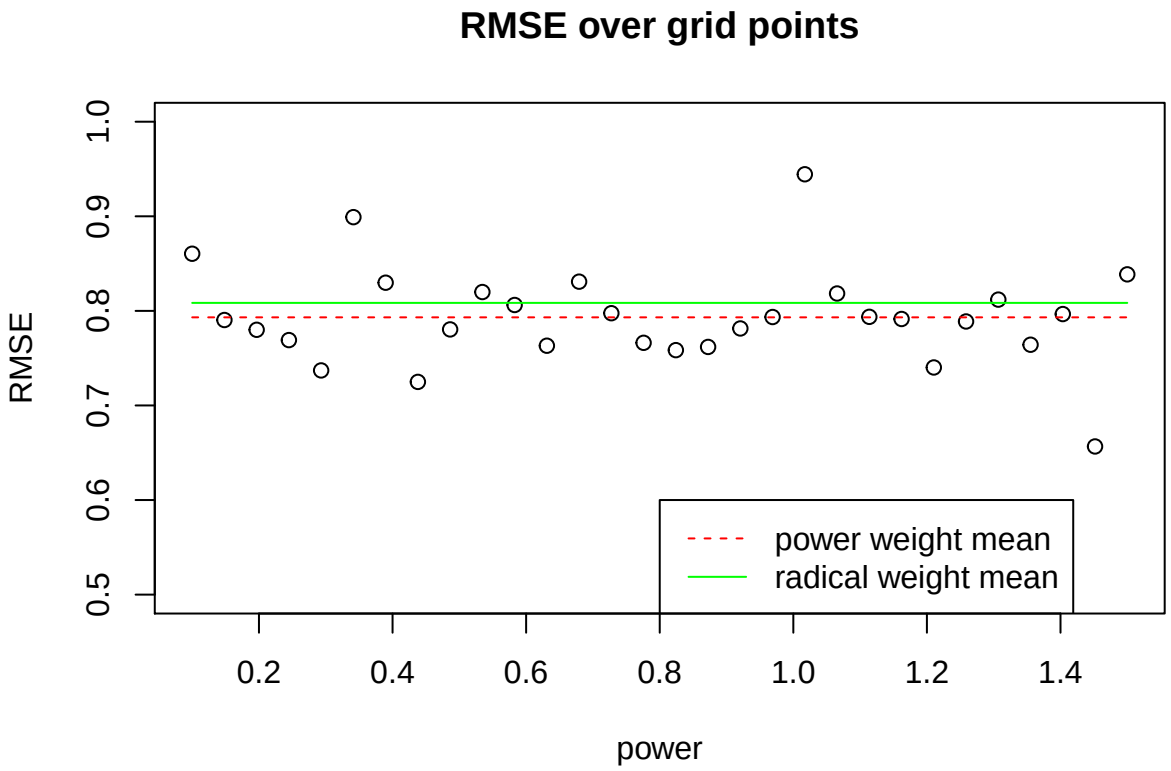
```
rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
G = graph.edgelist(as.matrix(rawData), directed=FALSE)
G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
subFIPSid = V(G)$name %in% locs$fips[tst_ind]
countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
graph0 = permute(countyGraph, order(V(countyGraph)$name));
nb.mat_tst = as_adjacency_matrix(graph0)
nb.mat_tst = nb.mat_tst[as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x == r
nb.mat_tst = nb.mat_tst[, as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x ==
ind_rm = -which(apply(nb.mat_tst, 1, function(x){sum(x) == 0}))
if (length(ind_rm) != 0){
  nb.mat_tst = nb.mat_tst[ind_rm, ind_rm]
  dist_mat = spDists(as.matrix(locs[locs$fips %in% rownames(nb.mat_tst), 1:2]))
  diag(dist_mat) = 1; dist_mat = 1/dist_mat^t
  nb.mat_tst = dist_mat
  nb.mat_tst = nb.mat_tst*dist_mat
  tst_data = tst_data[ind_rm, ]
  nb.w_tst = mat2listw(nb.mat_tst)
} else {
  dist_mat = spDists(as.matrix(locs[locs$fips %in% rownames(nb.mat_tst), 1:2]))
  diag(dist_mat) = 1; dist_mat = 1/dist_mat^t
  nb.mat_tst = dist_mat
  nb.mat_tst = nb.mat_tst*dist_mat
  nb.w_tst = mat2listw(nb.mat_tst)
}

# Weight matrix for training data
rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
G = graph.edgelist(as.matrix(rawData), directed=FALSE)
G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
subFIPSid = V(G)$name %in% locs$fips[-tst_ind]
countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
graph0 = permute(countyGraph, order(V(countyGraph)$name));
nb.mat_trn = as_adjacency_matrix(graph0)
nb.mat_trn = nb.mat_trn[as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x ==
nb.mat_trn = nb.mat_trn[, as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x =
ind_rm = -which(apply(nb.mat_trn, 1, function(x){sum(x) == 0}))
if (length(ind_rm) != 0){
  nb.mat_trn = nb.mat_trn[ind_rm, ind_rm]
  dist_mat = spDists(as.matrix(locs[locs$fips %in% rownames(nb.mat_trn), 1:2]))
  diag(dist_mat) = 1; dist_mat = 1/dist_mat^t
  nb.mat_trn = dist_mat
  nb.mat_trn = nb.mat_trn*dist_mat
  trn_data = trn_data[ind_rm, ]
  nb.w_trn = mat2listw(nb.mat_trn)
} else {
  dist_mat = spDists(as.matrix(locs[locs$fips %in% rownames(nb.mat_trn), 1:2]))
  diag(dist_mat) = 1; dist_mat = 1/dist_mat^t
  nb.mat_trn = dist_mat
  nb.mat_trn = nb.mat_trn*dist_mat
  nb.w_trn = mat2listw(nb.mat_trn)
}
fit = lagsarlm(trn_data$y~., data = trn_data, nb.w_trn)
```

```
    rho = as.vector(coef(fit)[1]); beta = as.vector(coef(fit)[-1])
    res = tst_data[, 1] - as.vector(solve(diag(1, nrow = nrow(nb.mat_tst)) - rho*nb.mat_tst) %*% as.mati
    rmse = c(mse, sqrt(mean(res^2)))
  }
  mean_rmse = c(mean_rmse, mean(rmse))
}
mean_rmse
```

```
##  [1] 0.8604175 0.7903439 0.7800088 0.7691200 0.7370431 0.8991076 0.8298040
##  [8] 0.7249047 0.7804062 0.8199128 0.8062297 0.7631555 0.8308864 0.7975365
## [15] 0.7662347 0.7584737 0.7618751 0.7814027 0.7935060 0.9444324 0.8183712
## [22] 0.7937006 0.7914283 0.7401722 0.7887962 0.8119076 0.7642080 0.7966475
## [29] 0.6566348 0.8386076
```

Here is a plot comparing the power transformation weight to radical weight.

```
plot(x = seq(0.1, 1.5, length.out = 30), y = mean_rmse, xlab = "power", ylab = "RMSE", "main" = "RMSE o
lines(x = seq(0.1, 1.5, length.out = 30), y = rep(RMSE, 30), col = "green")
lines(x = seq(0.1, 1.5, length.out = 30), y = rep(mean(mean_rmse), 30), lty = 2, col = "red")
legend(0.8,  0.6, legend = c("power weight mean","radical weight mean"), lty = c(2, 1), col = c("red",
```



In this plot, the RMSE of power weight are single points while the RMSE of radical weight is the solid green line. There appears to be a periodic fluctuating tendency of the RMSE of power transformation with respect to the radical RMSE. The dotted red line represents the mean of the RMSE of each power in the grid points. We can see that the power weight RMSE mean is close and even higher than that of radical. I think these two weights are roughly the same and for a computational ease, it's better to use radical weight.

Spatial SLX and Lasso

Note that so far, all of our analysis were made based on the data pre-selected by lasso over the entire data. Lasso selects some features so as to reduce prediciton error but does not account for spatial dependence,

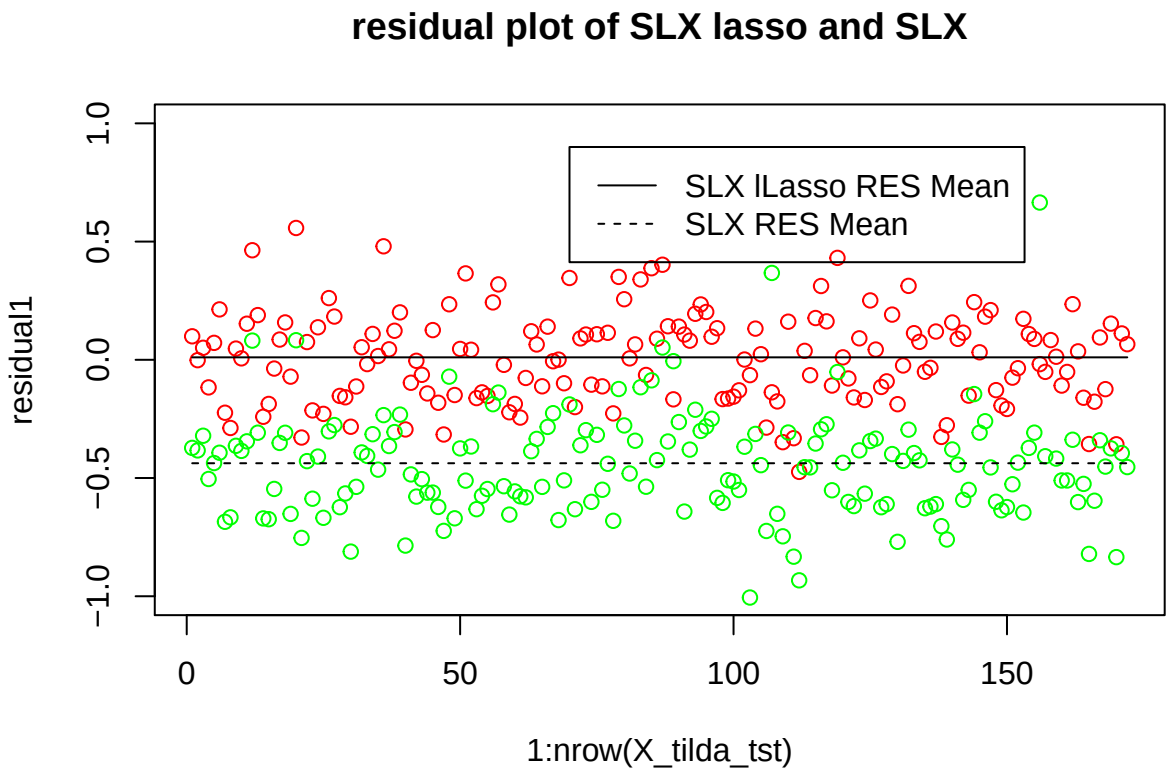
while spatial regression accounts for the latter only. We are wondering if we could perform spatial regression and model selection together and how would this method compare to direct application of spatial models. In this section, we will talk about this in a shallow level, comparing direct application of SLX model and SLX combined with lasso. The reson why we use SLX model is that SLX model could be direcly formulated into a OLS problem, which appears to be a good fit for an attempt.

```
sang_election = read.csv(file = "election_couty_cleaned.csv")
load(file = "full_data.Rdata")

############################### Data Creation ################################

full_data = full_data[-632,]
sang_election= sang_election[-623,]
ind = intersect(sang_election$fips, full_data$Id2)
y_ind = sapply(ind, function(x){which(x == sang_election$fips)})
X_ind = sapply(ind, function(x){which(x == full_data$Id2)})
y_ind = y_ind[order(as.integer(names(y_ind)))]
sang_election = sang_election[y_ind, ]
X_ind = sort(X_ind)
full_data = full_data[X_ind, ]
logOdds_dem2016 = log(sang_election$pct_dem_16/(1-sang_election$pct_dem_16))
logOdds_dem2012 = log(sang_election$pct_dem_12/(1-sang_election$pct_dem_12))
y = logOdds_dem2016 - logOdds_dem2012
Data = cbind(y, full_data[, -c(1: 4)])
X = as.matrix(Data[, -1])
locs = sang_election[, 1: 5]

############################### Cross Validation ################################
k = 10
Rmse = matrix(0, nrow = 2, ncol = k)
for (i in 1: k){
  tst_ind = sample(1: nrow(Data), 213, replace = F)
  tst_data = Data[tst_ind,]; trn_data = Data[-tst_ind,]
  # Weight matrix for test data
  rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
  G = graph.edgelist(as.matrix(rawData), directed=FALSE)
  G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
  subFIPSid = V(G)$name %in% locs$fips[tst_ind]
  countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
  graph0 = permute(countyGraph, order(V(countyGraph)$name)); ### this is the dense graph to begin with.
  nb.mat_tst = as_adjacency_matrix(graph0)
  nb.mat_tst = nb.mat_tst[as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x == row
  nb.mat_tst = nb.mat_tst[, as.vector(sapply(as.character(locs$fips[tst_ind]), function(x){which(x == c
  ind_rm = -which(apply(nb.mat_tst, 1, function(x){sum(x) == 0}))
  if (length(ind_rm) != 0){
    nb.mat_tst = nb.mat_tst[ind_rm, ind_rm]
    tst_data = tst_data[ind_rm, ]
    nb.w_tst = mat2listw(nb.mat_tst)
  } else {
    nb.w_tst = mat2listw(nb.mat_tst)
  }
  rawData = read.table("county_adjacency.munged.txt", colClasses=c("character", "character"))
  G = graph.edgelist(as.matrix(rawData), directed=FALSE)
  G = simplify(G, remove.loops=TRUE);V(G)$name=as.numeric(V(G)$name)
```

```
  subFIPSid = V(G)$name %in% locs$fips[-tst_ind]
  countyGraph = induced.subgraph(G, V(G)[subFIPSid]);
  graph0 = permute(countyGraph, order(V(countyGraph)$name));
  nb.mat_trn = as_adjacency_matrix(graph0)
  nb.mat_trn = nb.mat_trn[as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x == rou
  nb.mat_trn = nb.mat_trn[, as.vector(sapply(as.character(locs$fips[-tst_ind]), function(x){which(x == 
  ind_rm = -which(apply(nb.mat_trn, 1, function(x){sum(x) == 0}))
  if (length(ind_rm) != 0){
    nb.mat_trn = nb.mat_trn[ind_rm, ind_rm]
    trn_data = trn_data[ind_rm, ]
    nb.w_trn = mat2listw(nb.mat_trn)
  } else {
    nb.w_trn = mat2listw(nb.mat_trn)
  }

  X = as.matrix(trn_data[, -1]); y = trn_data[, 1]
  X_lag = nb.mat_trn %*% X
  colnames(X_lag) = paste("Lag", colnames(X_lag))
  X_tilda = as.matrix(cbind(X, X_lag))
  fit1 = cv.glmnet(x = X_tilda, y = y, family = "gaussian")
  X_tst = as.matrix(tst_data[, -1])
  X_lag_tst = nb.mat_tst %*% X_tst
  X_tilda_tst = as.matrix(cbind(X_tst, X_lag_tst))
  residual1 = tst_data[, 1] - predict(fit1, X_tilda_tst)
  Rmse[1, i]= sqrt(mean(residual1^2))

  fit2 = lm(y ~ X_tilda)
  X_tilda_tst = X_tilda_tst[, which(!is.na(as.vector(coef(fit2)[-1])))]
  beta = coef(fit2)[-1][which(!is.na(as.vector(coef(fit2)[-1])))]
  residual2 = tst_data[, 1] - X_tilda_tst %*% beta
  Rmse[2, i]= sqrt(mean(residual2^2))
}
rownames(Rmse) = c("SLX lasso", "SLX")
as.data.frame(Rmse)
```

```
##                  V1        V2        V3        V4        V5        V6
## SLX lasso 0.1939712 0.1877029 0.1769468 0.1857568 0.1869358 0.2044608
## SLX       0.5325610 0.5747595 0.5725571 0.4749266 0.4923919 0.5272475
##                  V7       V8       V9       V10
## SLX lasso 0.1881025 0.185314 0.192864 0.1904677
## SLX       0.5211098 0.459417 0.584695 0.5078648
```

```
plot(x = 1: nrow(X_tilda_tst), y = residual1, col = "red", ylim = c(-1, 1), main = "residual plot of SL
points(x = 1: nrow(X_tilda_tst), y = residual2, col = "green")
lines(x = 1: nrow(X_tilda_tst), y = rep(mean(residual1), nrow(X_tilda_tst)) , lty = 1)
lines(x = 1: nrow(X_tilda_tst), y = rep(mean(residual2), nrow(X_tilda_tst)), lty = 2)
legend(x = 70, y = 0.9, legend = c("SLX lLasso RES Mean","SLX RES Mean"), lty = c(1, 2))
```



Based on the comparison between the MSE of SLX lasso and SLX, we can see that applying SLX lasso largly reduce the MSE. Moreover, we can see from the plot that the residuals of SLX lasso are scatter around zero (solid line in the plot) while SLX appears to be biased. Therefore, in this case, combining SLX with lasso indeed increases the prediction accuracy. In the future,

Reference

Dubé, Jean, and Diègo Legros. 2014. *Spatial Autocorrelation*. https://doi.org/10.1002/9781119008651.ch3.

Le Gallo, Julie. 2014. *Cross-Section Spatial Regression Models*. https://doi.org/10.1007/978-3-642-23430-9_85.