Spatial Interpolation

Hilary Dugan

3/20/2018

Spatial Interpolation

If variables have spatial autocorrelation, we can predict values where no measurements have been taken

Precaution Only interpolate where it makes sense!

https://phys.org/news/2018-01-north-american-waterways-saltier-alkaline.html

Load spatial libraries

```
library(rgdal)
library(viridisLite)
library(spdep)
library(maptools)
library(rgdal)
library(raster)
library(rgeos)
library(gstat)
library(dismo) # Species Distribution Modeling
library(mapStats)
```

```
## Warning: package 'survey' was built under R version 3.4.4
```

Preliminary data output

Today, we'll work with %forest data from the WI NLCD.

First, let's calculate %forest across WI at a 15 km resolution

```
# Load state map
usMap <- readOGR(system.file("shapes/usMap.shp", package="mapStats")[1],layer='usMap',stringsAsFactors =
stMap = usMap[usMap@data$STATE_ABBR== 'WI',] # pull out Wisconsin

# Load NLCD
nlcd = raster('/Users/hilarydugan/Downloads/nlcd_2011_landcover_2011_edition_2014_10_10/nlcd_2011_landc

# Crop nlcd to state of Wisconsin
state = spTransform(stMap,crs(nlcd))
nlcdWI = crop(nlcd,state)

# Create % forest function
perF <- function(x,...) {
  c = 100*sum(x>=40 & x <=44 ,na.rm = T)/length(x)
  return(c)
}

# Aggregate raster to 500x the resolution (~15 km)
```

```
a = aggregate(nlcdWI,500,fun=perW)
b = mask(a,state) # mask to state
# Write Raster
writeRaster(b, 'Lecture8_Interpolation/Data/perWater.grd',overwrite=T)
```

Today's data

Get state map

```
usMap <- readOGR(system.file("shapes/usMap.shp", package="mapStats")[1],layer='usMap',stringsAsFactors =
```

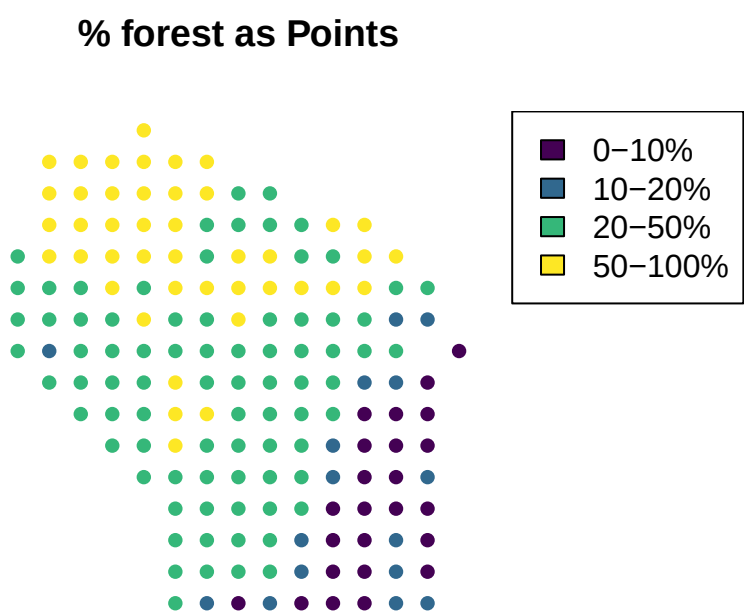
```
## OGR data source with driver: ESRI Shapefile
## Source: "/Library/Frameworks/R.framework/Versions/3.4/Resources/library/mapStats/shapes/usMap.shp", l
## with 51 features
## It has 116 fields
## Integer64 fields read as strings:  year
```

```
stMap = usMap[usMap@data$STATE_ABBR== 'WI',] # pull out Wisconsin
wi = stMap[,1] # get rid of most of the data frame
```

```
# Load raster of % forest
nlcdForest = raster('Data/perForest.grd')
names(nlcdForest)  = 'forest'

# Turn this raster into points
countyPts = SpatialPointsDataFrame(coordinates(nlcdForest),data = data.frame(forest = getValues(nlcdFor
countyPts = spTransform(countyPts,crs(wi))
countyPts = intersect(countyPts,wi)

# #Create color bins
pw = (1:4)[cut(countyPts$forest,breaks = c(-1,10,20,50,100))]
plot(countyPts,col = viridis(4)[pw],pch=16, main = '% forest as Points')
legend('topright',legend = c('0-10%','10-20%','20-50%','50-100%'),fill = viridis(4))
```



NULL model

The simplest way to predict the % forest at any locatoin would be to take the mean of all observations.

- Consider that a "Null-model" that we can compare other approaches to
- Use Root Mean Square Error (RMSE) as evaluation statistic

```
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}
null <- RMSE(mean(countyPts$forest,na.rm=T), countyPts$forest)
null
```

```
## [1] 18.40246
```

Comparing to NULL Model

Cross validate the result

Cross-validation is one of the most widely-used method for model selection, and for choosing tuning parameter values

- In k-fold cross-validation, the samples are randomly partitioned into k sets (called folds) of roughly equal size.
- A model is fit using all the samples except the first subset.
- Then, the prediction error of the fitted model is calculated using the first held-out samples.

- Repeated for each fold and the model's performance is calculated by averaging the errors across the different test sets.
- k is usually fixed at 5 or 10 .
- Cross-validation provides an estimate of the test error for each mode

Use the `dismo` package (made for species distribution modeling)

```
kf <- kfold(nrow(countyPoly), k = 5) # k-fold partitioning of a data set
kf
# Each record is randomly assigned to a group. Group numbers are between 1 and k.

rmse.model <- rep(NA, 5) # output rmse dataframe
for (k in 1:5) {
  test <- countyPoly[kf == k, ] #test data
  train <- countyPoly[kf != k, ] #training data
  gscv <- # here we will input our model based on our training data
  p <- predict(gscv, test)$var1.pred # predict values using test data
  rmse.model[k] <- RMSE(test$forest, p) # calculate RMSE with observed values
}
```

Defining Neighbors

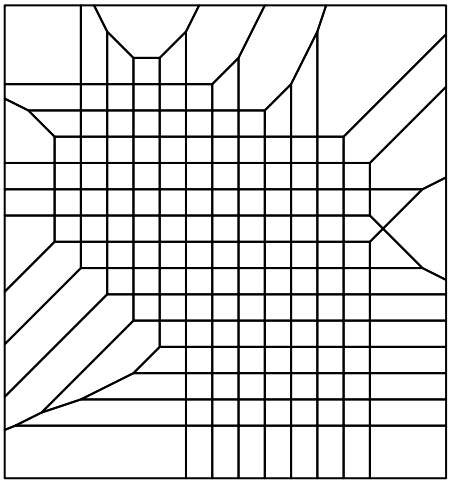
As we talked about last week, neighbors can be defined in many ways:

- As polygons that share a border with the location in question
- As being within a certain distance from the location in question
- As a specific number of neighbors that are closest to the location –> k-nearest neighbors (KNN)

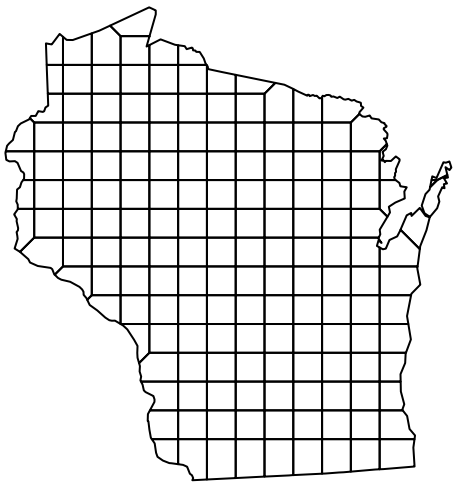
Proximity Polygons

Here we're going to use the point data

```
v <- voronoi(countyPts) #dismo package
# Create Voronoi polygons for a set of points.
# (These are also known Thiessen polygons, and Nearest Neighbor polygons;
# and the technique used is referred to as Delauny triangulation.)
plot(v)
```

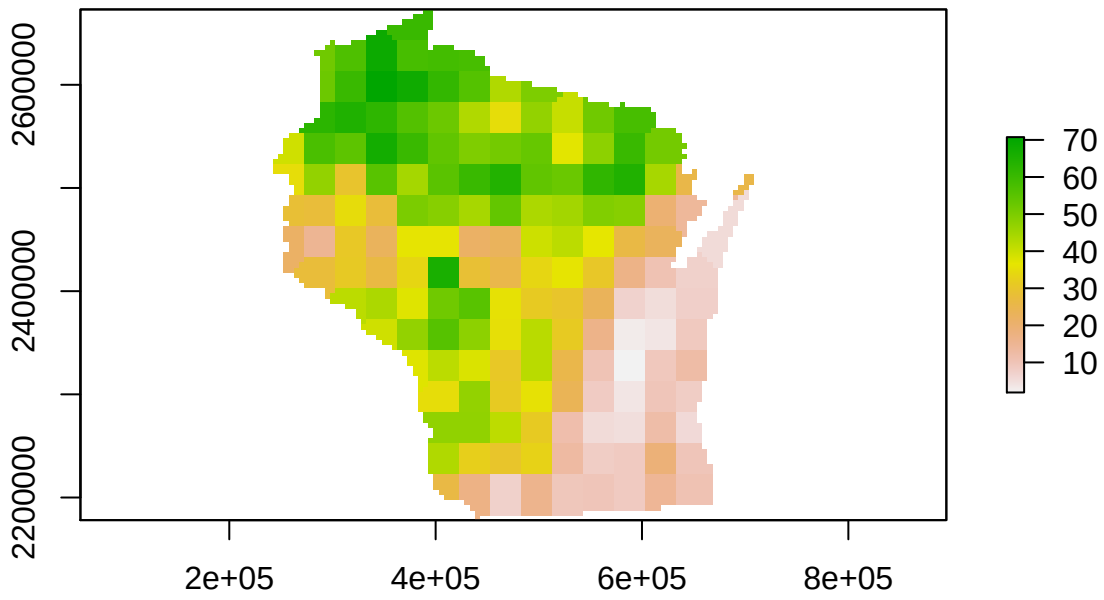


```r
v.wi <- intersect(v, wi) # crop to just wisconsin
plot(v.wi)
```



```r
# Can also 'rasterize' the results

r <- raster(wi, res=5000)
vr <- rasterize(v.wi, r, 'forest')
plot(vr)
```



Now evaluate with 5-fold cross validation.

```
kf <- kfold(nrow(countyPts))
rmse.v <- rep(NA, 5)
for (k in 1:5) {
  test <- countyPts[kf == k, ]
  train <- countyPts[kf != k, ]
  v <- voronoi(train)
  p <- raster::extract(v, test)
  p = p[!duplicated(p$point.ID),]
  rmse.v[k] <- RMSE(test$forest, p$forest)
}
rmse.v
```

```
## [1]  9.672406 12.953108 11.118239 10.830253 10.496885
```

```
mean(rmse.v) # Null model was 18
```

```
## [1] 11.01418
```

```
1 - (mean(rmse.v) / null)
```

```
## [1] 0.4014833
```

Nearest neighbour interpolation

Here we do nearest neighbour interpolation considering multiple (k) neighbours.

We can use the `gstat` package for this

- formula = defines the dependent variable as a linear model of independent variables
  - for ordinary and simple kriging use the formula z~1
  - suppose z is linearly dependent on x and y, use the formula z~x+y
- nmax = the number of nearest observations that should be used
- idp = "inverse distance power" idp to zero, such that all five neighbors are equally weighted
- idp = 2, inversely proportional to the squared distance.

```
library(gstat)
gs50 <- gstat(formula=forest~1, locations=countyPts, nmax=50, set=list(idp = 0))
gs10 <- gstat(formula=forest~1, locations=countyPts, nmax=10, set=list(idp = 0))
gs5 <- gstat(formula=forest~1, locations=countyPts, nmax=5, set=list(idp = 0))
gs2 <- gstat(formula=forest~1, locations=countyPts, nmax=5, set=list(idp = 2))

#Create raster to interpolate across
r <- raster(countyPts, res=10000)

interpforest <- function(gstatValue){
  nn <- interpolate(r, gstatValue) ## inverse distance weighted interpolation
  nnmsk <- mask(nn, wi) ## Mask to state outline
  plot(nnmsk)
}
par(mfrow=c(2,2), mar=c(1,1,1,1))
interpforest(gs50)
```

```
## [inverse distance weighted interpolation]
```

```
interpforest(gs10)
```

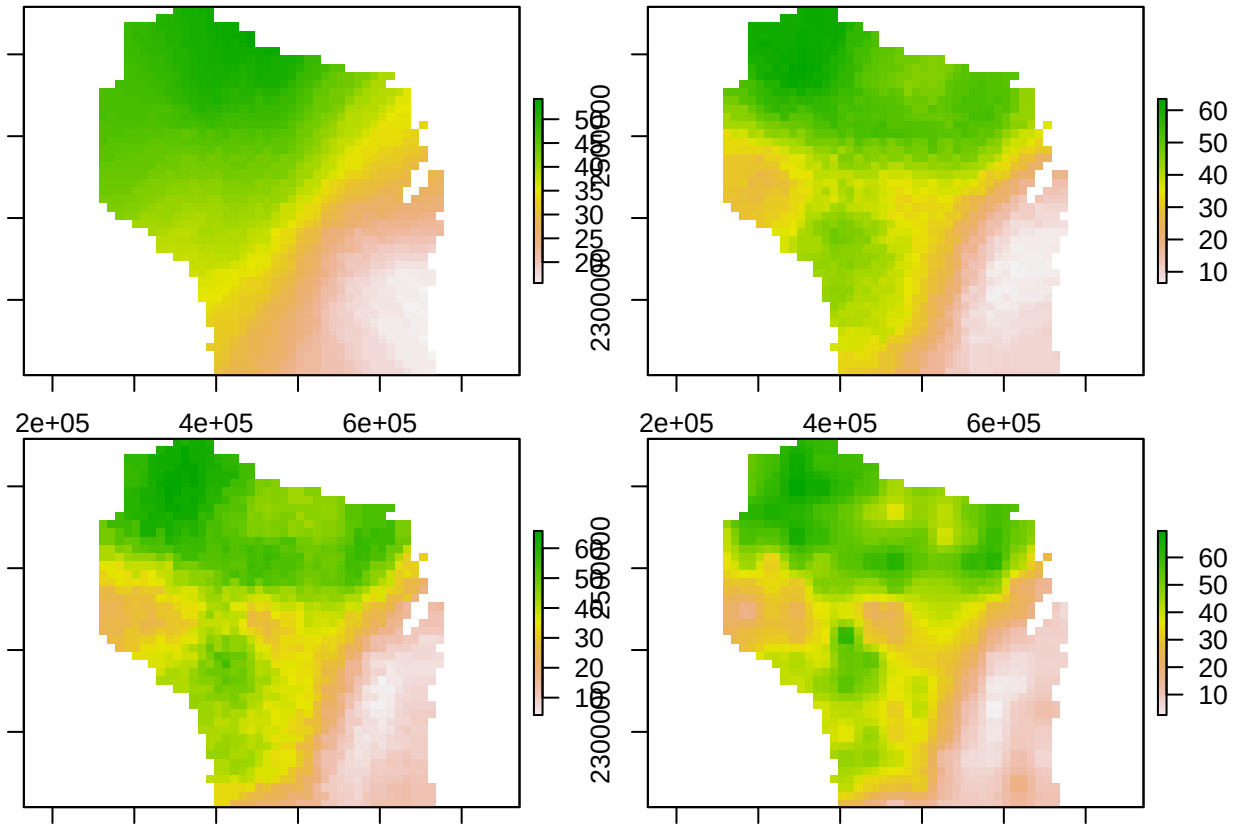
```
## [inverse distance weighted interpolation]
```

```
interpforest(gs5)
```

```
## [inverse distance weighted interpolation]
```

```
interpforest(gs2)
```

```
## [inverse distance weighted interpolation]
```



Cross validate NN-model with NULL model

```
kf <- kfold(nrow(countyPts), k = 5) # k-fold partitioning of a data set
kf
```

```
##   [1] 1 4 1 3 5 3 2 3 3 3 5 2 5 2 5 2 1 2 1 1 3 4 5 5 5 1 5 4 1 2 2 3 3 2 1
##  [36] 3 1 2 4 2 5 1 2 1 4 4 2 5 1 1 1 2 2 2 1 1 4 5 2 3 3 2 4 5 3 5 5 5 4 2
##  [71] 3 3 1 2 3 3 4 3 3 5 3 1 5 2 4 1 2 5 3 5 4 5 5 4 5 1 4 1 3 4 3 3 5 3 4
## [106] 2 1 2 1 5 2 1 3 5 4 4 2 2 1 5 4 1 2 5 4 3 4 1 3 4 2 1 1 3 1 4 5 1 5 3
## [141] 2 4 4 4 2 4 2 5 5 3 4 2 5 4 4 1 4 1 4 3 3 3 5
```

```
# Each record is randomly assigned to a group. Group numbers are between 1 and k.

rmse.NN <- rep(NA, 5) # output rmse dataframe
for (k in 1:5) {
  test <- countyPts[kf == k, ] #test data
  train <- countyPts[kf != k, ] #training data
  gscv <- gstat(formula=forest~1, locations=train, nmax=5, set=list(idp = 0)) # input our model based o
  p <- predict(gscv, test)$var1.pred # predict values using test data
  rmse.NN [k] <- RMSE(test$forest, p) # calculate RMSE with observed values
}
```

```
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
```

```
rmse.NN
```

```
## [1] 9.318011 7.557193 6.503733 7.639426 7.211349
```

```
mean(rmse.NN) # Null model was 18
```

```
## [1] 7.645942
```

```
1 - (mean(rmse.NN ) / null)
```

```
## [1] 0.5845152
```

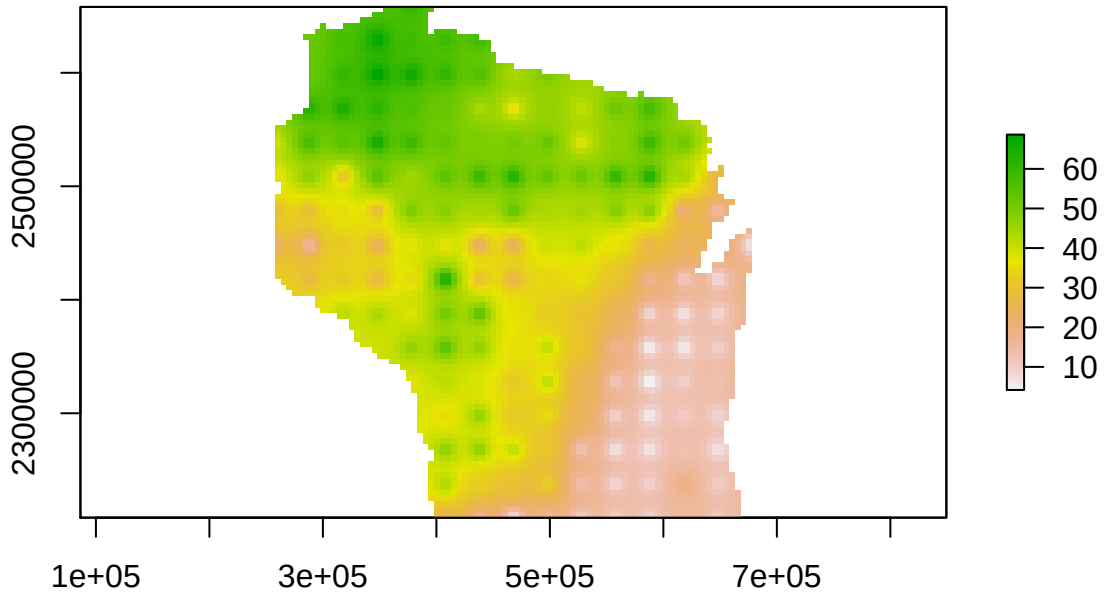
Inverse distance weighted

A more commonly used method is "inverse distance weighted" interpolation. The only difference with the nearest neighbour approach is that pointns that are further away get less weight in predicting a value a location.

```
#Create raster to interpolate across
r <- raster(countyPts, res=5000)
gs <- gstat(formula=forest~1, locations=countyPts)
idw <- interpolate(r, gs)
```

```
## [inverse distance weighted interpolation]
```

```
idwr <- mask(idw, wi)
plot(idwr)
```



Cross validate IDW-model with NULL model

```
rmse.idw <- rep(NA, 5)
for (k in 1:5) {
  test <- countyPts[kf == k, ]
  train <- countyPts[kf != k, ]
  gs <- gstat(formula=forest~1, locations=train)
  p <- predict(gs, test)
  rmse.idw[k] <- RMSE(test$forest, p$var1.pred)
}
```

```
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
```

```
rmse.idw
```

```
## [1] 11.210368  9.414813  8.892810 11.223244  8.397292
```

```
mean(rmse.idw) # Null model was 18
```

```
## [1] 9.827705
```

```
1 - (mean(rmse.idw) / null)
```

```
## [1] 0.4659569
```

Kriging

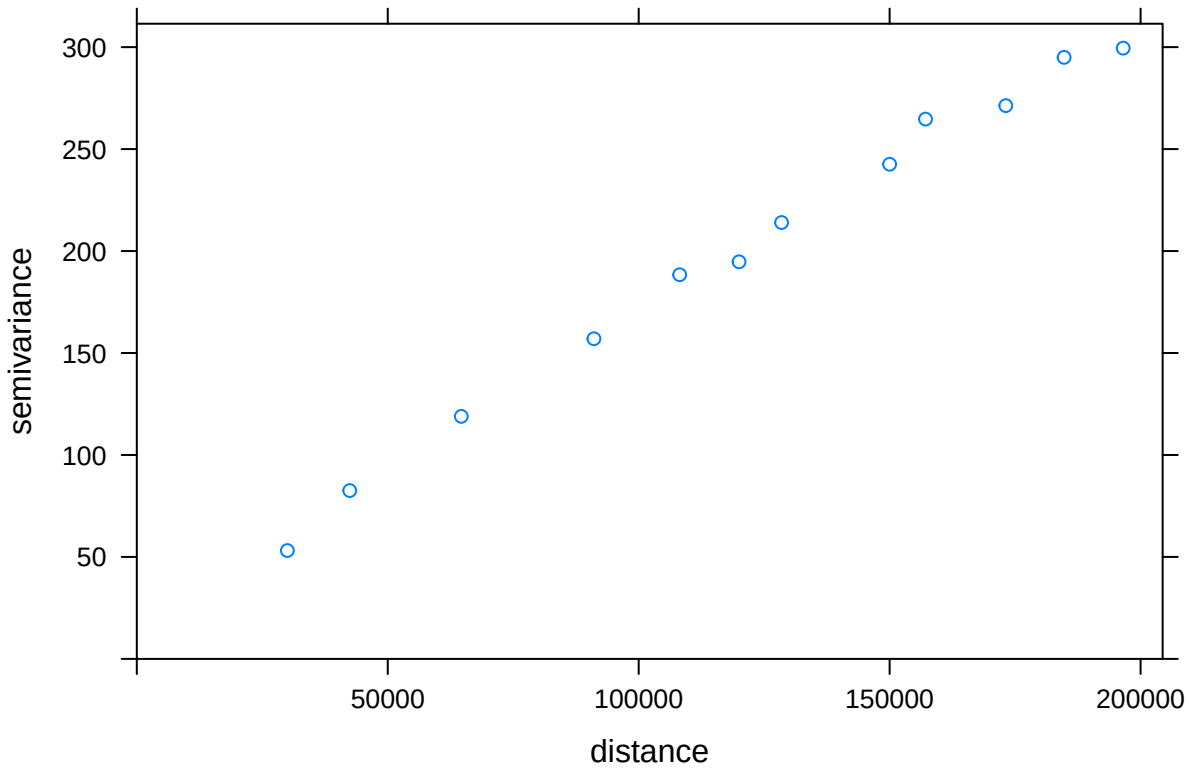
Kriging takes into account spatial autocorrelation

- IDW differs from Kriging in that no statistical models are used.
- IDW does not take into accout spatial autocorrelation
- In IDW only known z values and distance weights are used to determine unknown areas
- IDW has the advantage that it is easy to define and therefore easy to understand the results.
- Kriging is most appropriate when you know there is a spatially correlated distance or directional bias in the data.
- Kriging is a statistical method that makes use of a variograms to calculate the spatial autocorrelation between points at graduated distances
- Uses spatial autocorrelation to determine the weights that should be applied at various distances.

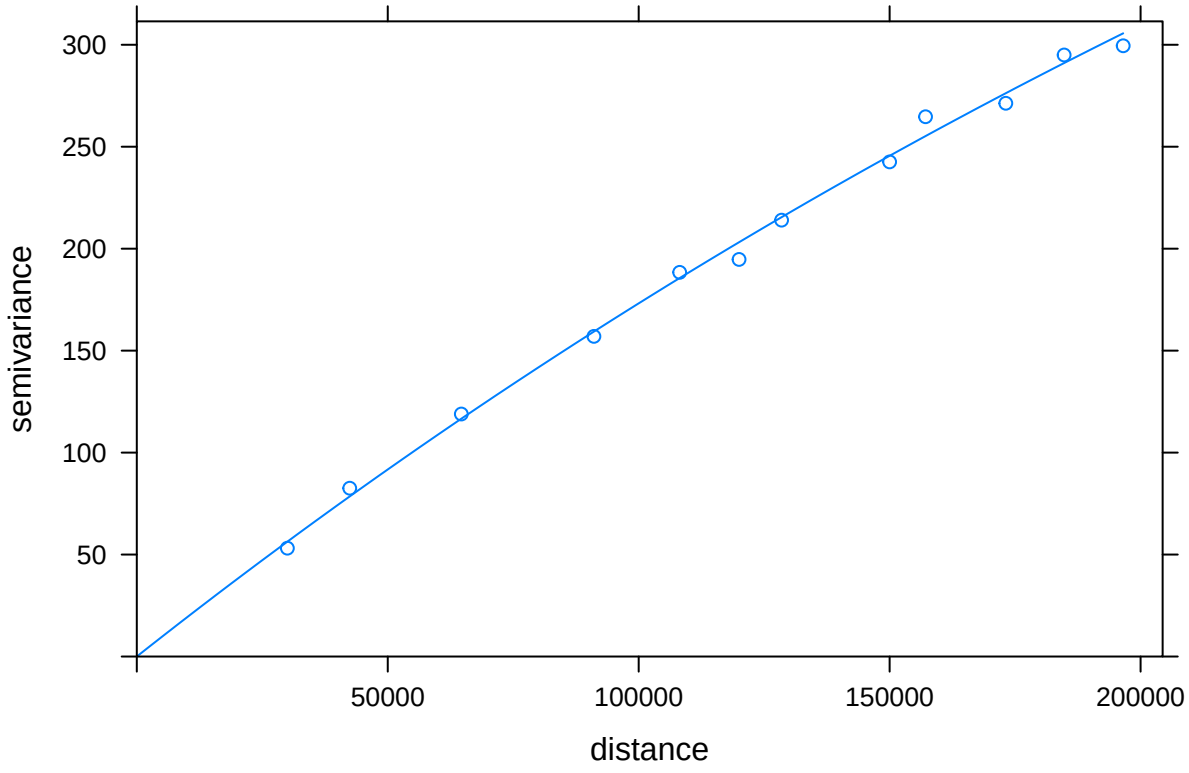
```
# Use gstat to create an emperical variogram 'v'
gs <- gstat(formula=forest~1, locations=countyPts)
v <- variogram(gs)
head(v)
```

```
##    np      dist     gamma dir.hor dir.ver   id
## 1 293  30000.00  53.12387       0       0 var1
## 2 280  42426.41  82.53162       0       0 var1
## 3 775  64660.44 118.92410       0       0 var1
## 4 931  91079.42 156.99922       0       0 var1
## 5 429 108166.54 188.40143       0       0 var1
## 6 208 120000.00 194.73028       0       0 var1
```

```
plot(v)
```




```
# Fit a exponential model variogram using default values
fve <- fit.variogram(v, vgm(NA, "Exp", NA, NA)) #Can have different forms (like 'Sph' = Spherical)
plot(v, fve)
```

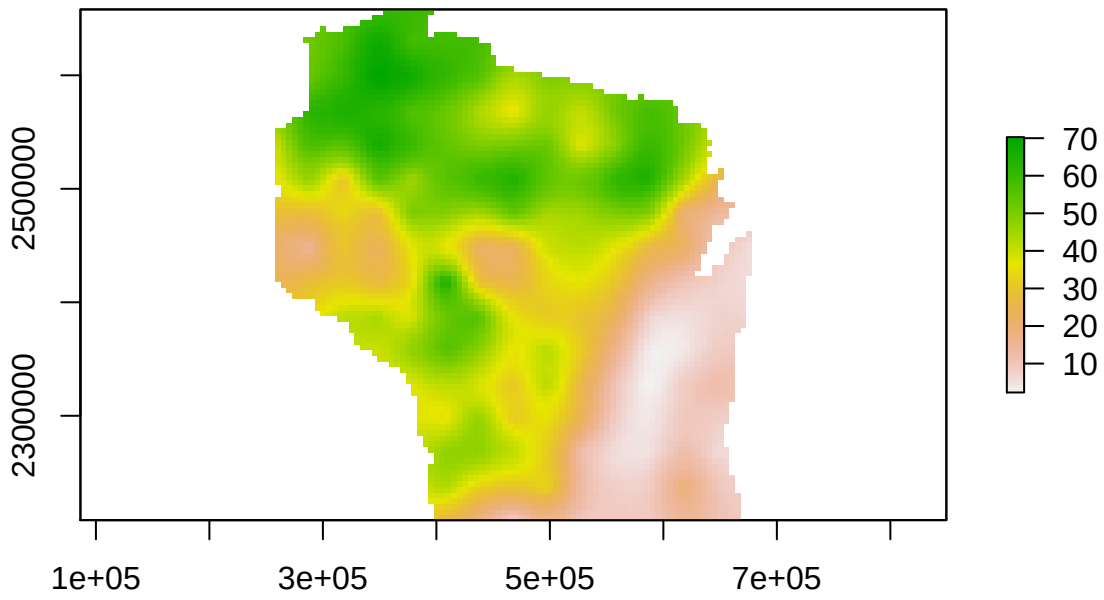


Ordinary kriging

```
#Create raster to interpolate across
r <- raster(countyPts, res=5000)
gs <- gstat(formula=forest~1, locations=countyPts, model=fve)
krig <- interpolate(r, gs)
```

```
## [using ordinary kriging]
```

```
krig <- mask(krig, wi)
plot(krig)
```



Cross validate kriging with NULL model

```
rmse.k <- rep(NA, 5)
for (k in 1:5) {
  test <- countyPts[kf == k, ]
  train <- countyPts[kf != k, ]
  gs <- gstat(formula=forest~1, locations=train, model=fve)
  p <- predict(gs, test)
  rmse.k[k] <- RMSE(test$forest, p$var1.pred)
}
```

```
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
## [using ordinary kriging]
```

```
rmse.k
```

```
## [1] 8.612280 6.930886 5.182837 6.330017 5.744478
```

```
mean(rmse.idw)
```

```
## [1] 9.827705
```

```
1 - (mean(rmse.idw) / null)
```

```
## [1] 0.4659569
```

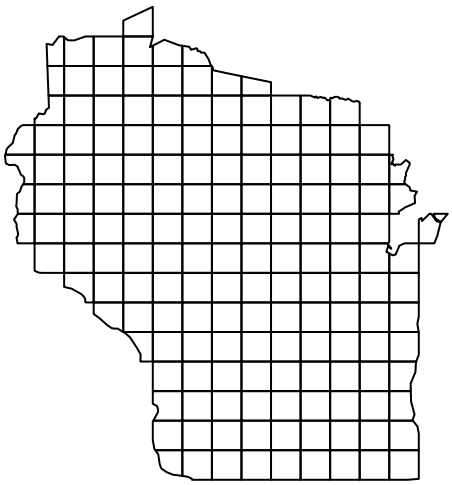
Smoothing

Smoothing is generally used to identify hot spots

Here, we create the localG statistic for polygon. This is the same as the Getis-Ord score: http://pro.arcgis.com/en/pro-app/tool-reference/spatial-statistics/h-how-hot-spot-analysis-getis-ord-gi-spatial-stati.htm

Here we want polygon data (vector instead of raster), so first transform raster to polygon

```
forestPoly = rasterToPolygons(nlcdForest, n=4, na.rm=TRUE, digits=12, dissolve=FALSE)
forestPoly = spTransform(forestPoly,crs(wi))
forestPoly = intersect(forestPoly,wi)
plot(forestPoly)
```



Next create k-nearest neighbor matrix (like last week)

```
# Get county coordinates
knn50 <- knn2nb(knearneigh(coordinates(forestPoly), k = 10))
knn50 <- include.self(knn50)

# Calculate G values
localGvalues <- localG(x = forestPoly$forest, listw = nb2listw(knn50, style = "B"), zero.policy = TRUE)
localGvalues <- round(localGvalues,3)

# Add gscores G scores
forestPoly$gvalues <- localGvalues

# Plot original polygon data
```

```
par(mar=c(1,1,1,1),mfrow=c(1,2))
pw = (1:4)[cut(forestPoly$forest,breaks = c(-1,10,20,50,100))]
plot(forestPoly,col = viridis(4)[pw],pch=16)
legend('topright',legend = c('0-10%','10-20%','20-50%','50-100%'),fill = viridis(4))

# Plot smoothed data
gw = (1:11)[cut(localGvalues,breaks = seq(-6,6))]
plot(forestPoly, col = viridis(11)[gw])
legend('topright',legend = c('less forest','more forest'), fill = viridis(7)[c(1,7)], bty='n')
```