hw7.R

joebrew

Mon Oct 20 11:22:53 2014

```r
# Attach necessary spatial packages
library(rgdal)
```

```
## Loading required package: sp
## rgdal: version: 0.9-1, (SVN revision 518)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.10.1, released 2013/08/26
## Path to GDAL shared files: /usr/share/gdal/1.10
## Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
## Path to PROJ.4 shared files: (autodetected)
```

```r
library(gstat)
library(geoR)
```

```
## Loading required package: MASS
## --------------------------------------------------------------
##  Analysis of geostatistical data
##  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
##  geoR version 1.7-4.1 (built on 2012-06-29) is now loaded
## --------------------------------------------------------------
```

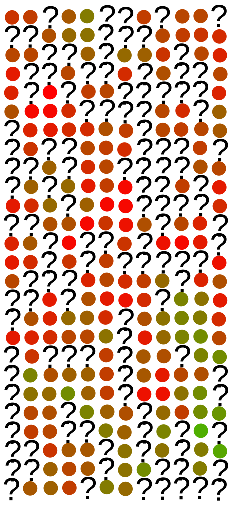
```r
library(RColorBrewer)

# Set wd
setwd("/home/joebrew/Documents/uf/phc6194/hw7")

# Read in data
s1 <- readOGR("data", "SubSample1")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "SubSample1"
## with 187 features and 6 fields
## Feature type: wkbPoint with 2 dimensions
```

```r
s2 <- readOGR("data", "SubSample2")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "SubSample2"
## with 125 features and 6 fields
## Feature type: wkbPoint with 2 dimensions
```

```r
# Plot
xcol <- colorRampPalette(c("green", "red"))(max(ceiling(s1$PM25)))
plot(s1, pch = 16, col = xcol[ceiling(s1$PM25)])
points(s2, col = "black", pch = "?")
```

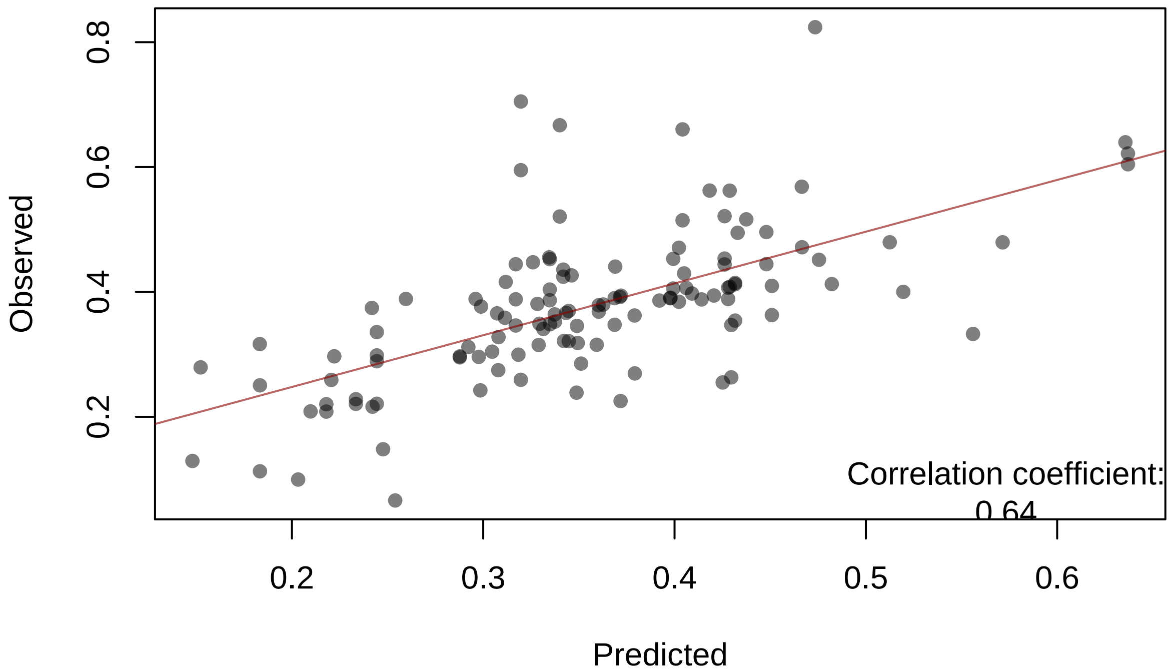


```
# Polygonal interpolation ###########

# Create thiessen / voronoi polygons
# first, function:
# (from http://stackoverflow.com/questions/9403660/how-to-create-thiessen-polygons-from-points-using-r-p
voronoipolygons <- function(x) {
  require(deldir)
  require(sp)
  if (.hasSlot(x, 'coords')) {
    crds <- x@coords
  } else crds <- x
  z <- deldir(crds[,1], crds[,2])
  w <- tile.list(z)
  polys <- vector(mode='list', length=length(w))
  for (i in seq(along=polys)) {
    pcrds <- cbind(w[[i]]$x, w[[i]]$y)
    pcrds <- rbind(pcrds, pcrds[1,])
    polys[[i]] <- Polygons(list(Polygon(pcrds)), ID=as.character(i))
  }
  SP <- SpatialPolygons(polys)
  voronoi <- SpatialPolygonsDataFrame(SP, data=data.frame(x=crds[,1],
                                                 y=crds[,2], row.names=sapply(slot(SP, 'polygon
                                                                              function(x) slot
}

# Create thissen polygon version of s1 (training data)
s1poly <- voronoipolygons(s1)
```

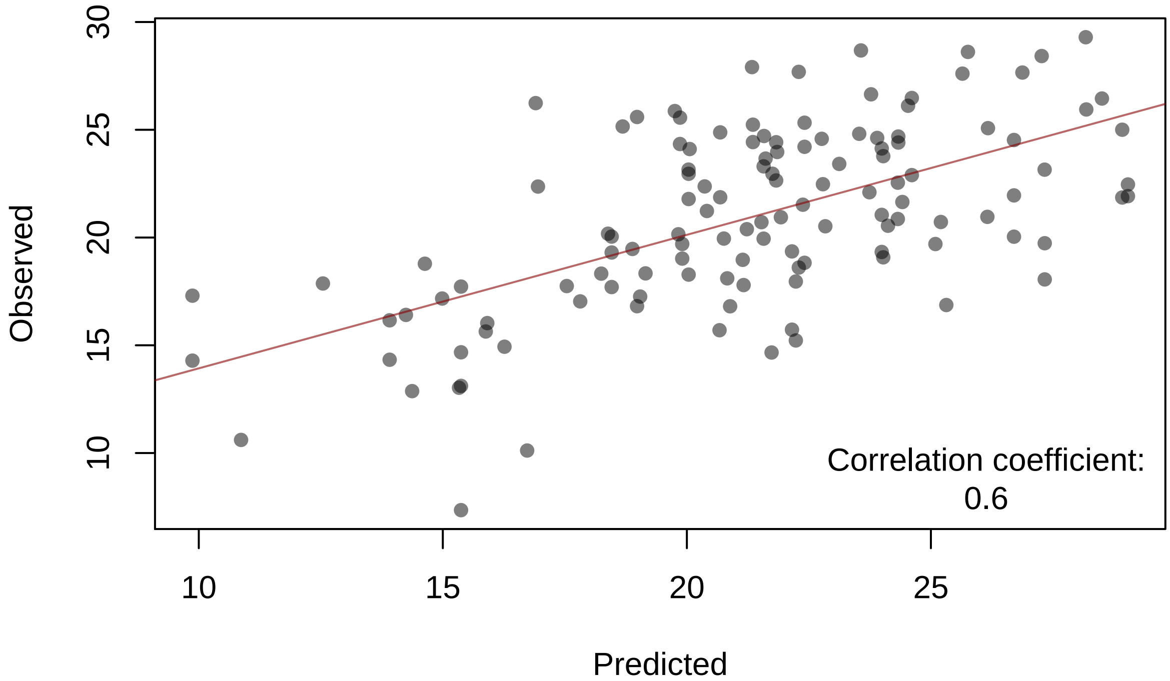
```
## Loading required package: deldir
## deldir 0.1-6

##
##      PLEASE NOTE:  The components "delsgs" and "summary" of the
##      object returned by deldir() are now DATA FRAMES rather than
##      matrices (as they were prior to release 0.0-18).
##      See help("deldir").
##
```

```
##      PLEASE NOTE: The process that deldir() uses for determining
##      duplicated points has changed from that used in version
##      0.0-9 of this package (and previously). See help("deldir").
```

```
# Bring in data
s1poly@data <- s1@data

# Define projection string for s1poly
proj4string(s1poly) <- proj4string(s1)

# Which of s1's polygons do all the s2 points fall into?
x <- over(s2, polygons(s1poly))

# Get all of s1's appropriate values for the s2 points

# Monthly aod
s2$MonthlyAOD.p <- NA
for (i in 1:nrow(s2)){
  ind <- x[i]
  s2$MonthlyAOD.p[i] <-
    s1poly$MonthlyAOD[ind]
}

# PM25
s2$PM25.p <- NA
for (i in 1:nrow(s2)){
  ind <- x[i]
  s2$PM25.p[i] <-
    s1poly$PM25[ind]
}

# Plot Correlation between monthly AOD true and predicted
JoePlot <- function(x,y){
  plot(x,y,
       xlab = "Predicted",
       ylab = "Observed",
       pch = 16,
       col = adjustcolor("black", alpha.f = 0.5))
  mylm <- lm(y ~x)
  abline(mylm, col = adjustcolor("darkred", alpha.f = 0.6))
  mycor <- cor(x,y)
  text(0.9*max(x),
       1.2*min(y),
       labels = paste0("Correlation coefficient:\n", round(mycor, digits =2)))
}

# MONTHLY AOD
JoePlot(s2$MonthlyAOD.p, s2$MonthlyAOD)
```

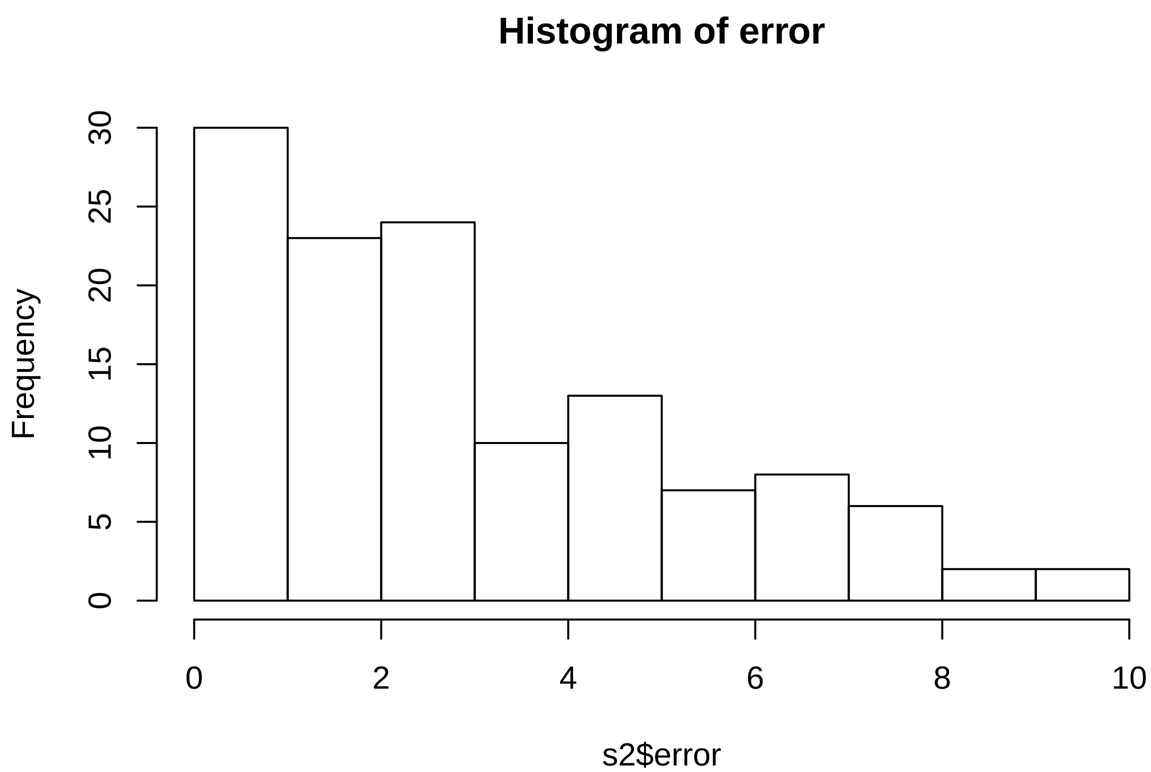



```
# PM 25
JoePlot(s2$PM25.p, s2$PM25)
```




```
# Calculate error
ErrorCalc <- function(x,y){
  z <- x-y
  z[which(z <= 0)] <- z[which(z <= 0)] * -1
  return(z)
}
s2$error <- ErrorCalc(s2$PM25.p, s2$PM25)
```

```r
# Explore error
hist(s2$error, main ="Histogram of error")
```




```r
# Fill out table
mean(s2$PM25.p)
```

```
## [1] 21.32539
```

```r
median(s2$PM25.p)
```

```
## [1] 21.58075
```

```r
min(s2$PM25.p)
```

```
## [1] 9.869708
```

```r
max(s2$PM25.p)
```

```
## [1] 29.0372
```

```r
sd(s2$PM25.p)
```

```
## [1] 4.149048
```

```r
mean(s2$error)
```

```
## [1] 2.96558
```

```r
median(s2$error)
```

```
## [1] 2.240936
```

```r
min(s2$error)
```

```
## [1] 0.075162
```

```r
max(s2$error)
```

```
## [1] 9.335048
```

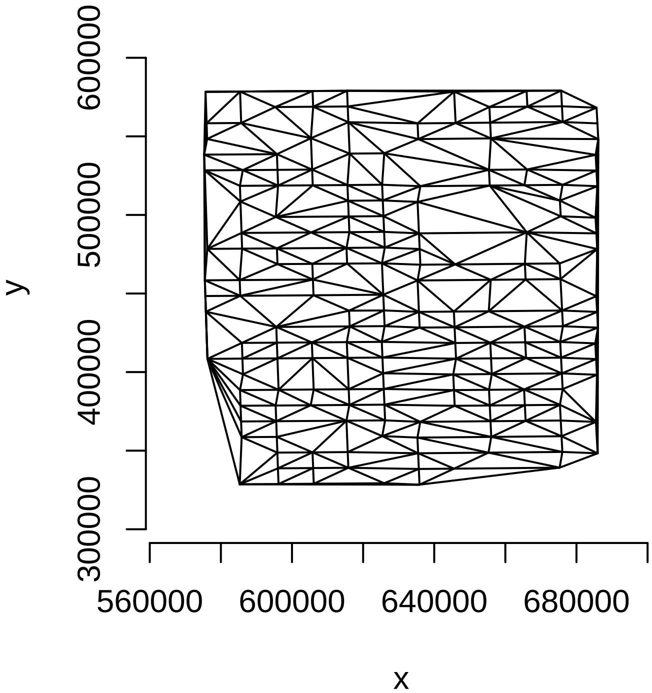
```r
sd(s2$error)
```

```
## [1] 2.296784
```

```r
mean(s2$error^2)
```

```
## [1] 14.02768
```

```r
## TRIANGULATION ####################

# Calculate triangles
library(deldir)
s1tri <- deldir(x = coordinates(s1)[,1],
                y = coordinates(s1)[,2])

# plot
plot(s1tri, wl = "triang", pch = NA)
```



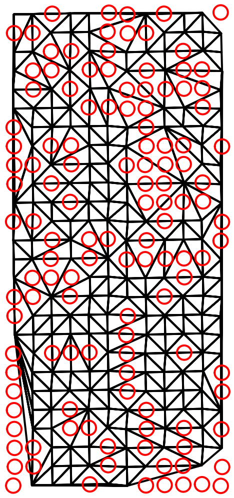

```r
# calculate tile perimeters
tilePerim(tile.list(s1tri))
```

```
## $perimeters
##   [1] 104199.96  95335.94  87963.06 100767.65 111084.28  87847.33 104439.15
##   [8]  56881.75  47193.40  56033.32  49088.38  44767.42  39612.18  74294.65
##  [15]  63160.89  60044.83  58665.71  62763.20  51817.43  47194.71  56716.21
##  [22]  52557.11  61207.26  61128.05  66066.88  69322.43  67097.81  59910.19
##  [29]  56499.76  58332.12  67508.33  68930.67  70841.94  44614.42  39485.09
##  [36]  48170.59  63259.65  59245.33  59104.44  54812.99  47635.68  47909.01
##  [43]  46326.03  50746.62  62772.76  68009.65  53397.43  43672.37  60011.18
##  [50]  81313.23  48176.76  39292.10  63536.59  58199.46  59810.35  47313.27
##  [57]  43804.54  58387.86  60669.56  72436.46  52701.07  40291.65  39740.39
##  [64]  66949.98  82043.77  60756.67  84417.55  46402.49  47583.41  44139.45
##  [71]  39689.72  53823.10  75074.27  48507.27  43493.87  47612.86  47236.42
##  [78]  40723.94  62796.70  56830.29  59186.35  62621.36  54459.56  55389.17
##  [85]  56861.28  59413.52  51085.70  61140.23  52634.91  56411.73  63638.56
##  [92]  55505.68  70909.41  74841.62  52239.18  39820.05  46529.77  51104.30
##  [99]  59475.67  52061.41  52075.50  61642.73  47172.41  39122.31  44173.44
## [106]  43768.63  48956.43  40061.04  52913.27  58276.55  39663.17  44461.91
## [113]  39807.22  46500.32  47232.65  43621.16  39664.94  39463.91  52460.79
## [120]  84710.04  39679.07  46727.86  49677.37  46727.86  50142.29  49764.49
## [127]  39680.83  43666.04  39120.31  52653.45  55428.66  53819.06  50391.53
## [134]  44126.06  43422.91  66855.19  70896.56  46907.20  49508.17  46754.45
## [141]  49193.18  49196.03  39519.88  44072.56  54679.33  72485.94  39533.73
## [148]  43891.03  40187.02  47066.33  50506.67  39728.21  39728.21  53738.50
## [155]  72798.28  46748.13  51729.71  40620.82  51429.52  46750.37  43765.42
## [162]  39931.20  74006.34  86599.56  47886.78  48255.71  49972.61  56989.44
## [169]  50956.04  53752.43  47847.99  53609.62  54144.52  67096.52  49994.06
## [176]  87740.38  53499.51  39788.98  50722.68  46071.91  85675.69 138067.45
## [183] 128748.01  81791.04  88251.35  90170.69 114170.58
```

```
##
## $totalPerim
## [1] 10672339
##
## $meanPerim
## [1] 57071.33
```

```r
# Create list of triangles
trilist <- triang.list(s1tri)


# Make list of triangles a polygon object
mypolys <- list()
for (i in 1:length(trilist)){

  x <- rbind(cbind(trilist[[i]]$x, trilist[[i]]$y),
             cbind(trilist[[i]]$x[1], trilist[[i]]$y[1]))

  x <- Polygon(x)

  mypolys[i] <- x
}

# Create a spatial polygons obejct from my polys
mytris <- SpatialPolygons(list(Polygons(mypolys, ID = 1)))

# give proj4string
proj4string(mytris) <- proj4string(s1)

# Get the mean value of the three coordinates which make up
# the vertices of each triangle
# And populate a vector
val_vector <- vector(length = length(mytris@polygons[[1]]@Polygons),
                     mode = "numeric")
for (i in 1: length(mytris@polygons[[1]]@Polygons)){

  # Extract vertices
  mymat <- mytris@polygons[[1]]@Polygons[[i]]@coords

  # Remove fourth (since it's just the loop back to close the triangle)
  mymat <- mymat[-4,]

#   #make df
#   mymat <- data.frame(mymat)

  # Get corresponding values in s1
  myvals <- s1$PM25[which(round(coordinates(s1)[,1], digits = 1) %in% round(mymat[,1], digits = 1) &
                            round(coordinates(s1)[,2], digits = 1) %in% round(mymat[,2], digits = 1))]

  #Get mean
  myval <- mean(myvals)
  val_vector[i] <- myval
}
```

```r
# Bind the triangles with the val_vector (the interpolated values) # won't work
# mytris2 <- SpatialPolygonsDataFrame(mytris@polygons[[1]]@Polygons,
#                                     data.frame("predicted" = val_vector))

# Can't quite get this to work
plot(mytris) # prediction triangles
points(s2, col = "red")
```



```r
## IDW ####################
library(spatstat)
```

```
##
## spatstat 1.38-1       (nickname: 'Le Hardi')
## For an introduction to spatstat, type 'beginner'
##
## Attaching package: 'spatstat'
##
## The following object is masked from 'package:gstat':
##
##     idw
```

```r
# # make ppp object of coordinates s1 and s2
# s1ppp <- as.ppp(coordinates(s1),
#                 c(min(coordinates(s1)[,1]),
#                   max(coordinates(s1)[,1]),
#                   min(coordinates(s1)[,2]),
#                   max(coordinates(s1)[,2])))
# s2ppp <- as.ppp(coordinates(s2),
#                 c(min(coordinates(s2)[,1]),
#                   max(coordinates(s2)[,1]),
#                   min(coordinates(s2)[,2]),
#                   max(coordinates(s2)[,2])))
# idw(x = s1ppp,
#     power = 1,
#     at = s2ppp)
```

```r
## KRIGING ############
# Define color vector (using colorbrewer)
my_colors <- colorRampPalette(c("darkgreen", "red"))(100)

#   boundary_points <- boundary@polygons[[1]]@Polygons
#   boundary_points <- boundary_points[[1]]@coords



  # Get trap locations and data values
  a <- data.frame("x" = coordinates(s1)[,1],
                  "y" = coordinates(s1)[,2],
                  "z" = s1$PM25)
  # Make into a geodata object
  b <- as.geodata(a)

  # Predict multiple points in Alachua County's boundary
  x <- seq(min(coordinates(s2)[,1]), max(coordinates(s2)[,1]), length = 100)
  y <- seq(min(coordinates(s2)[,2]), max(coordinates(s2)[,2]), length = 100)

  # Make a grid of those points
  pred.grid <- expand.grid(x,y)


  # kriging calculations
  kc <- krige.conv(geodata = b, coords = b$coords, data = b$data,
                   locations = pred.grid,
                   krige = krige.control(type.krige = "ok",
                                         cov.pars = c(10, 10000)))
```

```
## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood
```

```r
  # Plot!
  # displaying predicted values
  image(kc, loc = pred.grid,
        col = my_colors,
        xlab=NA, ylab=NA,
        xaxt = "n",
        yaxt = "n",
        xpd = NA,
        bty = "n")

  # Define percentiles for legend
  legtemp <-  round(quantile(kc$predict, probs = seq(0,1,, length = 10)))

  legend(x="topright",
         fill = my_colors[c(1,11,22,33,44,55,66,77,88,100)],
         legend = c(legtemp[1], NA, NA, legtemp[4], NA, NA, legtemp[7], NA, NA, legtemp[10]),
         border = FALSE,
         bty = "n",
         ncol = 1,
```

```
        y.intersp = 0.5,
        title = " KrigingInterpolation",
        cex = 0.75)
```