GW Kappa and other 'Death to Kappa' measures

Lex Comber & Paul Harris

July 2016

1. Introduction

The final Geographically weighted extension is to calculate spatially distributed measures of a number of other statistics for reporting accuracy. These include the Kappa statistic estimate $\hat{\kappa}$, also known as k-hat and a full description of $\hat{\kappa}$ can be found in the classic Congalton (1991) paper at http://uwf.edu/zhu/evr6930/2.pdf. What $\hat{\kappa}$ seeks to do is to measure the relationship between chance agreements between *Observed* and *Predicted* classes and the expected disagreement. It uses all the elements in the correspondence matrix and not just the diagonal ones and provides a measure of the proportion of agreement after chance agreement has been removed.

However the Kappa statistic has been heavily criticized by Pontius and Millones (2011), particular because its underlying assumption of an expected random distribution of errors does not reflect the spatial auto-correlation of of many landscape processes including mapping errors. For these reasons Pontius and Millones (2011) recommend that the remote sensing community '*abandon the use of Kappa indices for purposes of accuracy assessment and map comparison*' and they suggest the use of *quantity disagreement* and *allocation disagreement* measures, described in their paper at https://www.researchgate.net/publication/233196329_Death_to_Kappa_birth_of_quantity_disagreement_and_allocation_disagreement_for_accuracy_assessment_Int_J_Remote_Sens

Both of these approaches are now considered under a Geographically Weighted framework in turn.

2. Data

The packages can be called and the data can be loaded into R. This can be done locally after you have set the working directory using the `setwd()` function. The example from my computer is below:

```r
library(GISTools)
library(spgwr)
setwd("/Users/geoaco/Desktop/my_docs_mac/leeds_work/workshop/datacode")
roilib <- readShapePoly("lib_clip.shp")
data <- read.csv("all_data_new2.csv")
```

Alternatively the code below loads into your working directory directly from a `RData` file also on the site.

```r
library(GISTools)
library(spgwr)
library(repmis)
source_data("http://github.com/lexcomber/LexTrainingR/blob/master/SpatialAccuracy.RData?raw=True")
```

```
## [1] "data"   "roilib"
```

And have a look at the what you have

```
ls()
```

```
## [1] "data"   "roilib"
```

```
head(data.frame(data))
```

```
##   PointID   East   North Urban_FS Vegetation_FS Woody_FS Grazing_FS
## 1       1 301847 3631819   0.2500         0.250    0.250       0.25
## 2       2 302491 3632155   0.0000         0.250    0.000       0.00
## 3       3 303834 3631818   0.0000         0.000    0.750       0.00
## 4       4 304480 3631008   0.2500         0.625    0.000       0.00
## 5       5 306691 3632967   0.6875         0.250    0.000       0.00
## 6       6 308175 3630784   0.0000         0.750    0.125       0.00
##   Bare_FS Boolean_FS Urban_RS Vegetation_RS Woody_RS Grazing_RS Bare_RS
## 1  0.0000          U    0.103         0.189    0.673      0.000   0.032
## 2  0.7500          B    0.256         0.036    0.387      0.000   0.321
## 3  0.2500          W    0.000         0.076    0.216      0.053   0.651
## 4  0.1250          V    0.112         0.372    0.215      0.185   0.110
## 5  0.0625          U    0.265         0.473    0.147      0.000   0.112
## 6  0.1250          V    0.000         0.365    0.312      0.143   0.175
##   Boolean_RS
## 1          V
## 2          B
## 3          W
## 4          V
## 5          V
## 6          V
```

And this can be set up for the analysis as before - this time making sure the data have a projection:

```
# the projection
lib.proj <- CRS("+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +no_defs ")
proj4string(roilib) <- lib.proj
# create the point data file
lib <- SpatialPointsDataFrame(cbind(data$East, data$North),
  data.frame(field = data$Boolean_FS, sat = data$Boolean_RS),
  proj4string = lib.proj)
# convert the data to numeric form
class.lut <- data.frame(code = unique(lib$field), num = c(1:5))
# Urban 1; Bare 2; Woodland 3;  V = 4; Grazing land = 5
# and reformat the attributes in 'lib'
index = match(lib$field, class.lut$code)
lib$field = class.lut$num[index]
index = match(lib$sat, class.lut$code)
lib$sat = class.lut$num[index]
```

3. GW crosstabs

For this we will use the `gwxtab` package which can be downloaded from `github` in the following way (**NB**: you may have to install the `devtools` package to do this):

```r
install.packages("devtools", dep = T)
library(devtools)
install_github('chrisbrunsdon/gwxtab')
```

And the the package can be loaded:

```r
library(gwxtab)
```

You should explore the documentation around `gwxtab` - Harry and I have have both worked extensively with Chris Brunsdon - he does some great work and creates great code.

4. Kappa

Calculating a normal, aspatial Kappa Estimate

The formal notation of the derivation of the Kappa statistic estimate $\hat{\kappa}$ is below:

$$\hat{\kappa} = \frac{N \sum_{i=1}^{r} x_{ii} - \sum_{i=1}^{r} (x_{i+} * x_{+i})}{N^2 - \sum_{i=1}^{r} (x_{i+} * x_{+i})} \tag{1}$$

Essentially what this does is:

1. Multiply the sum of the diagonals by the table sum.
2. Then subtract from this the sum of the row and column marginal totals products. In this case these would be $[(21x39) + (39 * 43) + (46 * 30) + (60 * 47) + (44 * 51)]$
3. Next, divide this by the sum of the table squared - minus the sum of the row and column marginal totals products as above

The top part of the equation gives a measure of chance agreement and the bottom part a measure of the expected disagreements.

We need to create the matrix and the code below creates the change / correspondence / accuracy matrix as before:

```r
tab <- table(data$Boolean_RS, data$Boolean_FS)
class.names.long <- c("Bare", "Grazing", "Urban", "Vegetation", "Woodland")
rownames(tab) <- class.names.long
colnames(tab) <- class.names.long
tab <- cbind(tab, rowSums(tab))
tab <- rbind(tab, colSums(tab))
rownames(tab)[6] <- "Total"
colnames(tab)[6] <- "Total"
tab
```

```
##            Bare Grazing Urban Vegetation Woodland Total
## Bare         18       8     7          2        4    39
## Grazing       3      23     3          8        6    43
## Urban         0       0    27          1        2    30
## Vegetation    0       4     7         31        5    47
## Woodland      0       4     2         18       27    51
## Total        21      39    46         60       44   210
```

So in this case a Kappa Estimate could be calculated from the table as follows:

```
tab.tmp <- tab[1:5, 1:5]
part.1 <- sum(diag(tab.tmp)) * sum(tab.tmp)
part.2 <- sum(colSums(tab.tmp) * rowSums(tab.tmp))
part.3 <- sum(tab.tmp)^2
k <- (part.1 - part.2) / (part.3 - part.2)
cat("Kappa estimate (k-hat): ", round(k,3))
```

```
## Kappa estimate (k-hat):  0.498
```

GW Crosstabs

Step 1: create a dummy crosstab

Once the data and libraries area loaded the first step is to create a dummy `crosstabs` object using the `new_spxt` function:

```
dummy_xtab <- new_spxt(lib,'field','sat')
```

Have a look at this `Spatial Crosstabulation` object.

```
dummy_xtab
```

```
## Spatial Crosstabulation object
## Number of locations: 210
## Dimension: 5 x 5
## Proj4 String: +proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +towgs84=0,0,0
```

```
head(coordinates(dummy_xtab))
```

```
##      coords.x1 coords.x2
## [1,]    301847   3631819
## [2,]    302491   3632155
## [3,]    303834   3631818
## [4,]    304480   3631008
## [5,]    306691   3632967
## [6,]    308175   3630784
```

The SpatialCrossTabs object has a location associated with each cross-tabulation, and a projection. The `coordinates` function gives the locations of each cross tab at each location and the square brackets `[]` reference individual cross-tabulations

```
dummy_xtab[7]
```

```
##      sat
## field 1 2 3 4 5
##     1 0 1 0 0 0
##     2 0 0 0 0 0
##     3 0 0 0 0 0
##     4 0 0 0 0 0
##     5 0 0 0 0 0
```

Step 2: create a GW crosstab - just for testing!

The next step is to create a set of dummy crosstabs at any geographical point location u_1, u_2. Functions that take a location as an argument and return the cross tabulation are called probe functions. The function `gwxtab_probe` is a tool for making probe functions. It takes a dummy cross-tabulation 'SpatialCrossTabs' object and a bandwidth to create a new function that maps a geographical location (u_1, u_2) on to a geographically weighted cross-tabulation. At this stage in the code development it defaults to bisquare kernel.

The kernel bandwidth can be fixed - e.g. 20km

```
gwxt <- gwxtab_probe(dummy_xtab,fixed(20))
# test it
round(gwxt(330749, 3627772 ), 3)
```

```
##      sat
## field     1     2     3     4     5
##     1 8.174 2.186 0.447 1.438 1.479
##     2 0.000 3.742 0.000 0.000 0.204
##     3 0.791 0.728 7.242 1.516 0.761
##     4 0.383 0.239 1.802 7.253 1.358
##     5 0.000 2.042 1.023 0.389 4.126
```

Or it can be adaptive to take the nearest n data points. First define a bandwidth - say 15% of the data points:

```
bw = 0.15
nrow(lib)
```

```
## [1] 210
```

```
bw = round(nrow(lib)*0.15, 0)
bw
```

```
## [1] 32
```

Then define the `gwxtab_probe` function:

```
gwxt_ad <- gwxtab_probe(dummy_xtab,adapt(bw))
round(gwxt_ad(330749, 3627772 ), 3)
```

```
##      sat
## field     1     2     3     4     5
##     1 1.472 0.406 0.000 0.000 0.886
##     2 0.000 0.287 0.000 0.000 0.000
##     3 0.289 0.145 2.530 0.000 0.178
##     4 0.000 0.000 0.000 2.133 0.000
##     5 0.000 0.186 0.122 0.000 0.584
```

Step 3: define a function to apply to the crosstab

The initial code below defines a function to calculate local measures of overall accuracy form each cross-tab:

```r
# overall accuracy function
ov <- function(x) data.frame(ov=sum(diag(x))/sum(x))
```

This can be tested:

```r
ov(gwxt_ad(330749, 3627772))
```

```
##          ov
## 1 0.7600426
```

And then incorporated into a `probe` function:

```r
gw_ov <- gwxtab_probe(dummy_xtab,adapt(bw),melt=ov)
# test it!
gw_ov(330749, 3627772)
```

```
##          ov
## 1 0.7600426
```

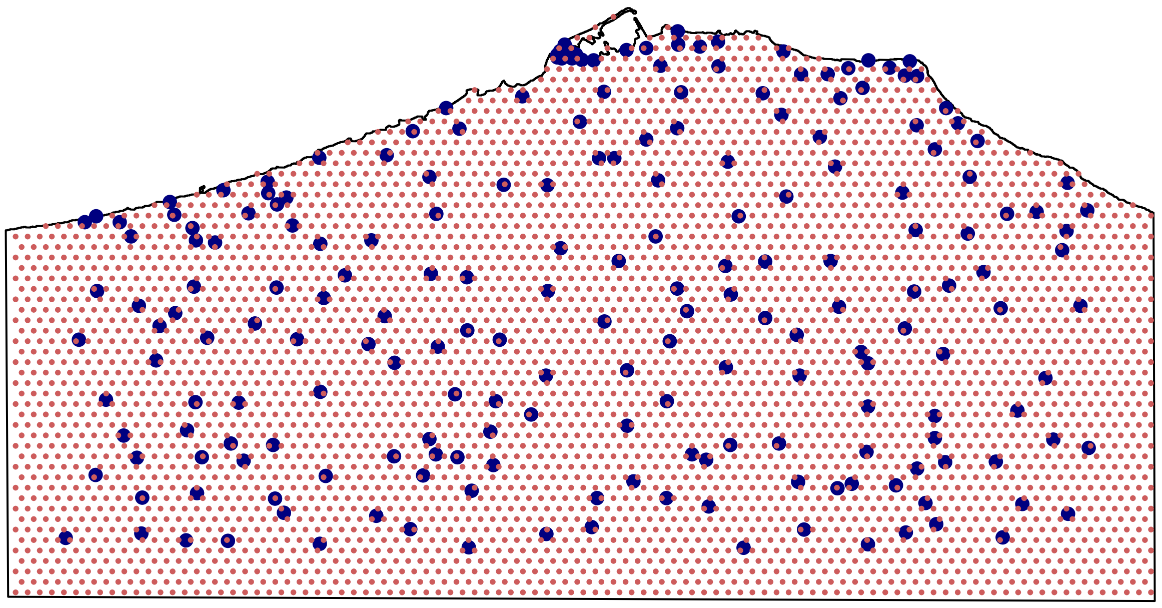
Step 4: Spatial Extension and Visualisation of local crosstab measures

The results can be visualized using the code used for the `spgwr` approaches described in earlier practicals. In this example hexbins are used.

```r
# create the hexbin objects
hg <- spsample(roilib,5000,'hexagonal',offset=c(0.5,0.5))
```

This creates around 5000 points on a hexagonal grid covering the study areas (the polygon `roilib`). These can be plotted:

```r
par(mar=c(0,0,0,0)+0.1)
plot(roilib)
plot(lib,pch=16,col='navy', add = T)
plot(hg,pch=16,col='indianred',cex=0.4,add=TRUE)
```

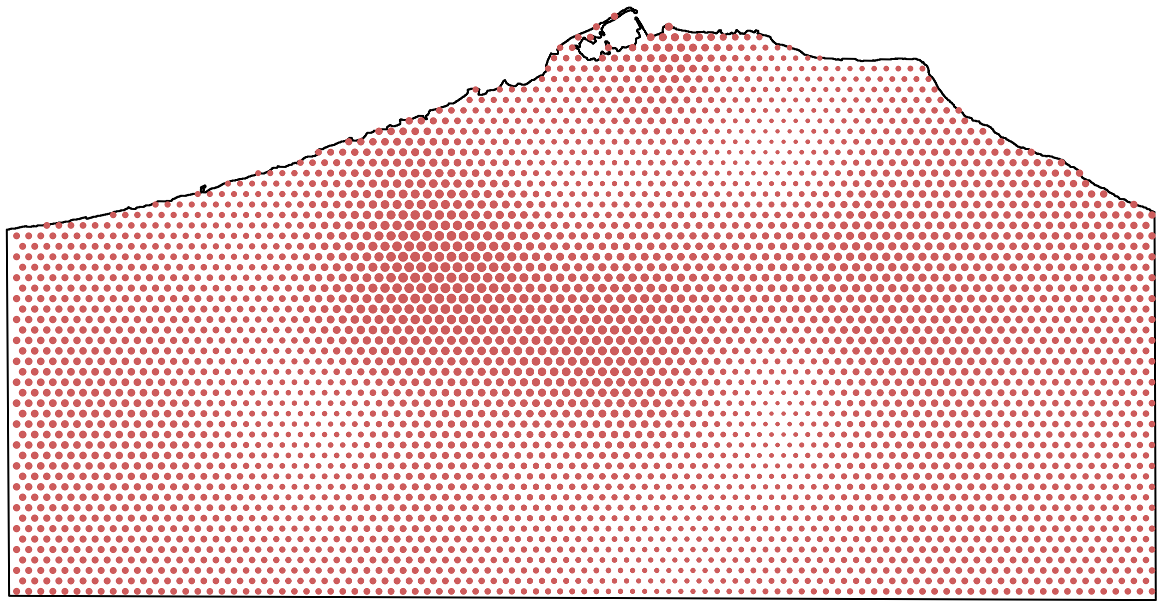


The SpatialPointsDataFrame object can finally be created - this may take a couple of seconds :

```
hg_ov <- gwxtab_sample(hg,dummy_xtab,adapt(bw),melt=ov)
```

And the spatially varying Overall Accuracy can be visualized to allow the size of each hexbin to vary in proportion to the value of the cross tab statistic - in this case overall accuracy:

```
par(mar=c(0,0,0,0)+0.1)
plot(roilib)
plot(hg_ov,pch=16,col='indianred',cex=0.8*hg_ov$ov,add=TRUE)
```



Sumamry

The steps above provide a framework for calculating any GW statistic from a correspondence matrix of cross tabulation. You should explore the parameters that are passed to the various functions here. For example, what is the effect of changing the bw variable to 15? How do the maps differ?

GW Kappa

The above code can be modified, replacing the function for overall accuracy ov, with a function for calculating local Kappa estimates. In this case we will use an adaptive bandwidth of 15 data points.

Step 1: create a dummy crosstab

This is the same as above but repeated for clarity.

```
dummy_xtab <- new_spxt(lib,'field','sat')
```

Step 2: create a GW crosstab - just for testing!

```
gwxt_ad <- gwxtab_probe(dummy_xtab,adapt(15))
round(gwxt_ad(330749, 3627772 ), 3)
```

```
##      sat
## field     1     2     3     4     5
##     1 0.896 0.138 0.000 0.000 0.807
##     2 0.000 0.000 0.000 0.000 0.000
##     3 0.040 0.000 1.500 0.000 0.000
##     4 0.000 0.000 0.000 1.320 0.000
##     5 0.000 0.000 0.000 0.000 0.338
```

Step 3: define a function to apply to the crosstab

In this case this is the Kappa estimate:

```
# Define Kappa estimate
kp <- function(x) {
    part.1 <- sum(diag(x)) * sum(x)
    part.2 <- sum(colSums(x) * rowSums(x))
    part.3 <- sum(x)^2
    k <- (part.1 - part.2) / (part.3 - part.2)
    return(data.frame(kappa = k))
}
```

Step 4: Spatial Extension and Visualisation of local crosstab measures

Create the hexbin objects:

```
hg <- spsample(roilib,5000,'hexagonal',offset=c(0.5,0.5))
```

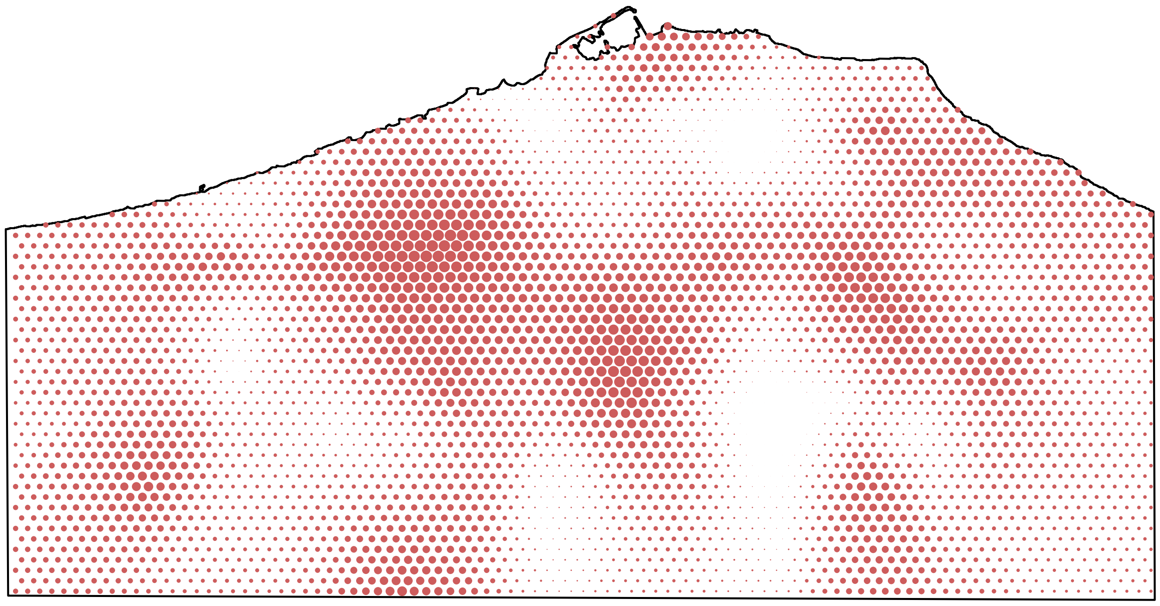
Create the `SpatialPointsDataFrame` object:

```
hg_kp <- gwxtab_sample(hg,dummy_xtab,adapt(15),melt=kp)

# now on occasion this can result in very small negative values
# and small negative anomalies can be set to the min positive value
# see http://www.cis.rit.edu/~ejipci/Reports/On_Using_and_Computing_the_Kappa_Statistic.pdf
negkap2min <- function (x) {
  index <- x < 0
  x[index] <- min(x[!index])
  return(x)}
hg_kp$kappa <- negkap2min(hg_kp$kappa)
```

Plot the hexagonal grid of points:

```
par(mar=c(0,0,0,0)+0.1)
plot(roilib)
#plot(lib,pch=16,col='navy',add=TRUE)
plot(hg_kp,pch=16,col='indianred',cex=0.8*hg_kp$kappa,add=TRUE)
```



Familiar visulisation

The code below visualizes the data in the same way as you have done in previous exercises. A slightly different grid has to be created to support he `level.plot` function and then the local kappa re-calculated over that grid:

```
# create a polygon for the study area
ext <- t(bbox(roilib))
ext <- rbind(ext, cbind(ext[1,1], ext[2,2]))
ext <- ext[c(1,3,2),]
ext <- rbind(ext, cbind(ext[3,1], ext[1,2]))
ext <- rbind(ext, ext[1,])
ext
```

```
##             x       y
## min 297345.8 3610249
##     297345.8 3644007
## max 362785.2 3644007
##     362785.2 3610249
##     297345.8 3610249
```

This can be used to make a polygon from which to create a regular sample grid:

```
poly <- Polygon(ext)
poly <- Polygons(list(poly), ID = "1")
poly <- SpatialPolygons(list(poly), integer(1))
# now use this to create a sample grid
grid <- spsample(poly,6000,'regular',offset=c(0.5,0.5))
proj4string(grid) <- lib.proj
```

Now the `SpatialPointsDataFrame` object can be created over this grid:

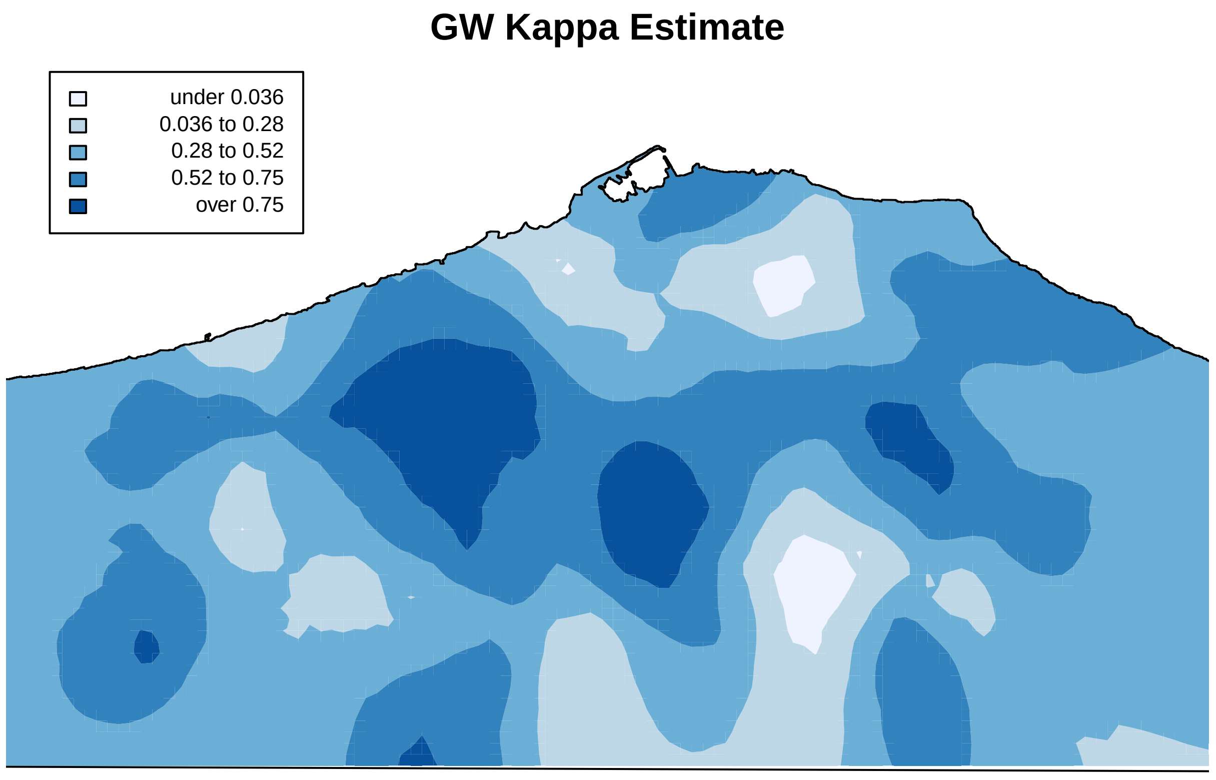
```
hg_kp <- gwxtab_sample(grid,dummy_xtab,adapt(15),melt=kp)
```

And plotted in the same way as in previous exercises:

```
shades = auto.shading(hg_kp$kappa, n=5,cols=brewer.pal(5,"Blues"),
  cutter=rangeCuts, digits = 2)
kp.spdf = SpatialPixelsDataFrame(hg_kp, data.frame(hg_kp$kappa))
par(mar=c(0,1,2,0)+0.1)
level.plot(kp.spdf,shades)
lib.masker = poly.outer(kp.spdf, roilib, extend = 100)
```

```
## Warning in RGEOSBinTopoFunc(spgeom1, spgeom2, byid, id, drop_lower_td,
## unaryUnion_if_byid_false, : spgeom1 and spgeom2 have different proj4
## strings
```

```
add.masking(lib.masker)
plot(roilib, add = T)
choro.legend(300000, 3648000, shades, cex = 0.7)
title("GW Kappa Estimate")
```



5. Quantity and Allocation Disagreement measures

Pontius and Millones (2011) define *quantity disagreement* as the amount of difference between the *Observed* reference data and the *Predicted* classified data, and compares the proportions of the classes.

They define *allocation disagreement* as the amount of difference between the *Observed* data and the *Predicted* data that are due mis-matches in the spatial allocation of classes, relative to the proportions of the classes in the *Observed* and *Predicted* data.

These can be calculated for the overall correspondence matrix using the code snippets below. First re-create the matrix

```
tab <- table(data$Boolean_RS, data$Boolean_FS) # FS columns, RS rows
class.names.long <- c("Bare", "Grazing", "Urban", "Vegetation", "Woodland")
rownames(tab) <- class.names.long
colnames(tab) <- class.names.long
tab
```

```
##
##               Bare Grazing Urban Vegetation Woodland
##   Bare          18       8     7          2        4
##   Grazing        3      23     3          8        6
##   Urban          0       0    27          1        2
##   Vegetation     0       4     7         31        5
##   Woodland       0       4     2         18       27
```

The GW approach applies these measures locally.

Quantity disagreement

The quantity disagreement is computed from the sum of the row totals (the *Predicted* data) minus the sum of the column totals (the *Observed* data) divided by 2:

```
q <- sum(abs(apply(tab, 1, sum) - apply(tab, 2, sum)))/2
q
```

```
## [1] 29
```

Now this provides a measure a quantity disagreement measure of `29` which is obviously specific to the number of pixels summarized in the matrix. To make this generic, this can be normalized by dividing by the `sum` of the matrix or by passing in a normalized matrix:

```
q <- sum(abs(apply(tab, 1, sum) - apply(tab, 2, sum)))/2
q/sum(tab)
```

```
## [1] 0.1380952
```

Or

```
q <- sum(abs(apply(tab/sum(tab), 1, sum) - apply(tab/sum(tab), 2, sum)))/2
q
```

```
## [1] 0.1380952
```

Allocation disagreement

The allocation disagreement can be computed from the overall difference between the *Observed* and *Predicted* data (ie total number of pixels minus the diagonal agreement) minus the *quantity disagrement*, `q`:

```
o <- sum(tab) - sum(diag(tab))
q <- sum(abs(apply(tab, 1, sum) - apply(tab, 2, sum)))/2
a <- o-q
a
```

```
## [1] 55
```

Again this can be normalized so that the measures are not proportional to the number of pixels in the correspondence matrix

```
o <- sum(tab) - sum(diag(tab))
q <- sum(abs(apply(tab, 1, sum) - apply(tab, 2, sum)))/2
a <- o-q
a/sum(tab)
```

```
## [1] 0.2619048
```

GW Allocation disagremment

The same steps are followed for the GW Allocation disagreement, which again requires construction of the `gwxtab` objects, etc: ### Step 1: create a dummy crosstab This is the same as above but repeated for clarity.

```
dummy_xtab <- new_spxt(lib,'field','sat')
```

Step 2: create a GW crosstab - just for testing!

```
gwxt_ad <- gwxtab_probe(dummy_xtab,adapt(15))
round(gwxt_ad(330749, 3627772 ), 3)
```

```
##      sat
## field     1     2     3     4     5
##     1 0.896 0.138 0.000 0.000 0.807
##     2 0.000 0.000 0.000 0.000 0.000
##     3 0.040 0.000 1.500 0.000 0.000
##     4 0.000 0.000 0.000 1.320 0.000
##     5 0.000 0.000 0.000 0.000 0.338
```

Step 3: define a function to apply to the crosstab

In this case this is the Quantity Disagreement :

```r
# Define function
qd <- function(x) {
    q <- sum(abs(apply(x, 1, sum)
    - apply(x, 2, sum)))/2
  # normalise
    q <- q/sum(x)
  return(data.frame(qd = q))
}
```

Step 4: Spatial Extension and Visualisation of local crosstab measures

Create the hexbin objects:

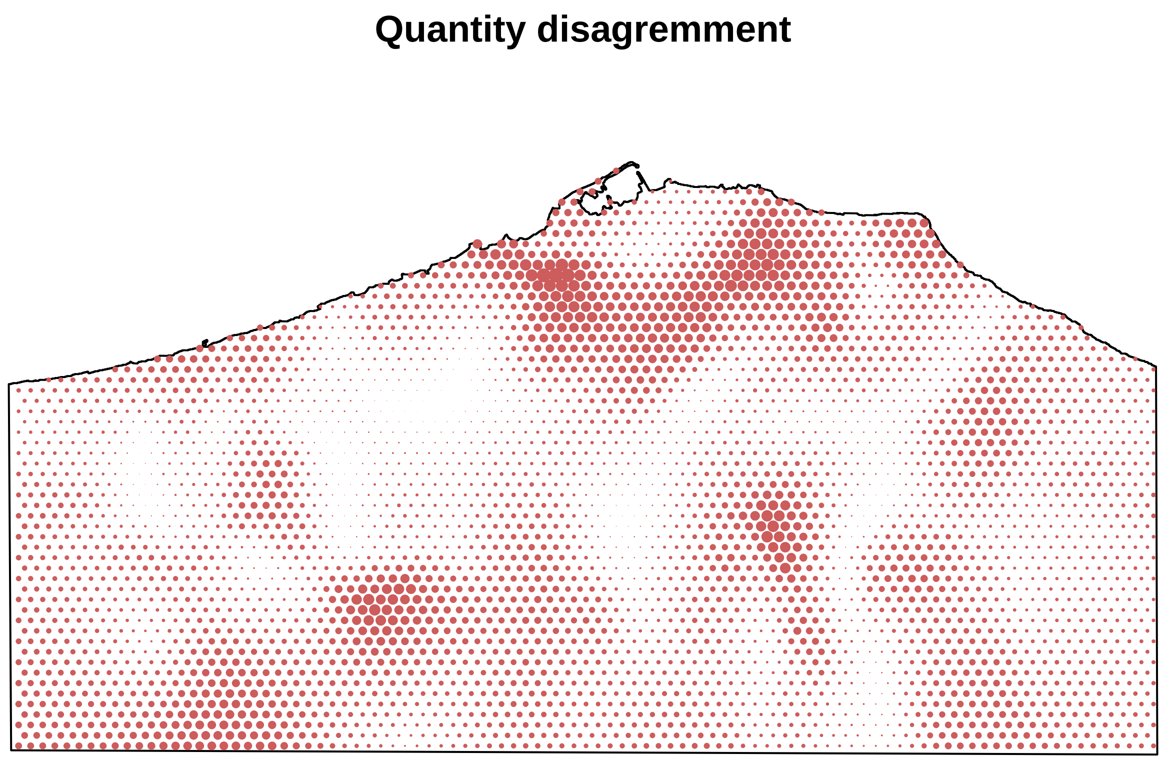
```r
hg <- spsample(roilib,5000,'hexagonal',offset=c(0.5,0.5))
```

Create the `SpatialPointsDataFrame` object:

```r
hg_qd <- gwxtab_sample(hg,dummy_xtab,adapt(15),melt=qd)
```

Plot the hexagonal grid of points:

```r
par(mar=c(0,0,1,0)+0.1)
plot(roilib)
plot(hg_qd,pch=16,col='indianred',cex=hg_qd$qd/max(hg_qd$qd),add=TRUE)
title("Quantity disagrement")
```



GW Quantity disagreemment

The GW approach applies these measures locally, which again requires construction of the `gwxtab` objects as in the Kappa example above. This is done in the following way: ### Step 1: create a dummy crosstab This is the same as above but repeated for clarity.

```
dummy_xtab <- new_spxt(lib,'field','sat')
```

Step 2: create a GW crosstab - just for testing!

```
gwxt_ad <- gwxtab_probe(dummy_xtab,adapt(15))
round(gwxt_ad(330749, 3627772 ), 3)
```

```
##      sat
## field     1     2     3     4     5
##     1 0.896 0.138 0.000 0.000 0.807
##     2 0.000 0.000 0.000 0.000 0.000
##     3 0.040 0.000 1.500 0.000 0.000
##     4 0.000 0.000 0.000 1.320 0.000
##     5 0.000 0.000 0.000 0.000 0.338
```

Step 3: define a function to apply to the crosstab

In this case this is the Allocation Disagreement :

```
# Define Kappa estimate
ad <- function(x) {
  o <- sum(x) - sum(diag(x))
  q <- sum(abs(apply(x, 1, sum) - apply(x, 2, sum)))/2
  a <- (o-q)/sum(x)
  return(data.frame(ad = a))
}
```

Step 4: Spatial Extension and Visualisation of local crosstab measures

Create the hexbin objects:

```
hg <- spsample(roilib,5000,'hexagonal',offset=c(0.5,0.5))
```

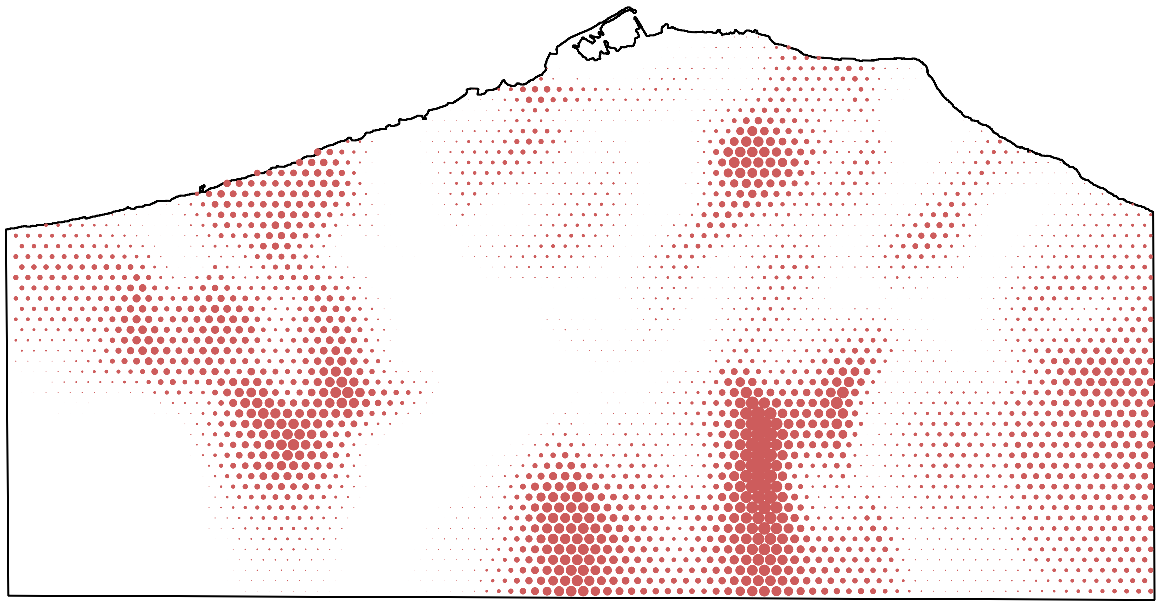
Create the `SpatialPointsDataFrame` object:

```
hg_ad <- gwxtab_sample(hg,dummy_xtab,adapt(15),melt=ad)
```

Plot the hexagonal grid of points:

```
par(mar=c(0,0,1,0)+0.1)
plot(roilib)
plot(hg_ad,pch=16,col='indianred',cex=hg_ad$ad/max(hg_ad$ad),add=TRUE)
title("Allocation disagremment")
```

Allocation disagremment



Epilogue

You may find it useful to have a look at the `diffeR` package put together by Gil Pontius and Alí Santacruz at https://cran.r-project.org/web/packages/diffeR/diffeR.pdf. This includes a number of functions for calculating quality and allocation disagreements, including per class measures which were not implemented here. You could perhaps try to develop some of these such as the `quantityDj` function into a GW framework.

In assembling these tutorials, a number of useful packages that were not previously known to us were found: `gwxtab` which really helped in the creation of the more complex (i.e. not probability based) spatially distributed measures of error. What we have done here is bolt these together with other commands from other packages - indicating that a good way of solving problems is to see who else might have done similar work.

End