Classification in QGIS/GRASS

Susan Steele-Dunne

March 2018

Before you start:

- **Read the lecture notes, and the Reader, particularly Jens Liebe's thesis!**

- Unzip the zip file.
- Create a new QGIS project file.
- Create a new mapset for your project. The data is in EPSG 32630 (WGS84 UTM 30N). You should use this as the CRS for your Location.
- Add one of the rasters to your project file and "zoom extents". Use the coordinates of the corners of your image to define the grass region.
- Set the CRS of your project to EPSG 32630 too.
- Import the three bands of data into your new GRASS mapset using **r.external -o**.

  (if you run r.external in the dialog box, show advanced options and select "over-ride projection")
- Add the GRASS raster maps to your QGIS project data frame.
- Set the GRASS region to be the extent of one of the raster maps (they are all the same, so any will do) ... see the next few slides.

We're going to use g.region to make a GRASS region that is defined by our input data. I'm going to use this as an opportunity to show you how to read the manual page.

Let's take a look at the manual page for **g.region**

I am using **GRASS 6.4.6,** so here is the manual page for that version:

Secure https://grass.osgeo.org/grass64/manuals/g.region.html

NAME

g.region - Manages the boundary definitions for the geographic region. Brief description of what the module does.

KEYWORDS

general, settings

SYNOPSIS

g.region These are flags, e.g. we typed g.region –p to print the current region
g.region help
g.region [**-dsplectwmn3bgau**] [**region**=*name*] [**rast**=*name*[,*name*,...]] [**rast3d**=*name*] [**vect**=*name*[,*name*,...]] [**3dview**=*name*] [**n**=*value*] [**s**=*value*] [**e**=*value*] [**w**=*value*] [**t**=*value*] [**b**=*value*] [**rows**=*value*] [**cols**=*value*] [**res**=*value*] [**res3**=*value*] [**nsres**=*value*] [**ewres**=*value*] [**tbres**=*value*] [**zoom**=*name*] [**align**=*name*] [**save**=*name*] [**--overwrite**] [**--verbose**] [**--quiet**]

Flags:

-d
: Set from default region

-s
: Save as default region
Only possible from the PERMANENT mapset

-p
: Print the current region

-l
: Print the current region in lat/long using the current ellipsoid/datum

-e
: Print the current region extent

We're going to use g.region to make a GRASS region that is defined by our input data.

Let's take a look at the manual page for **g.region**

I am using **GRASS 6.4.6**, so here is the manual page for that version:

Secure https://grass.osgeo.org/grass64/manuals/g.region.html

NAME

g.region - Manages the boundary definitions for the geographic region. Brief description of what the module does.

KEYWORDS

general, settings

SYNOPSIS

g.region
g.region help
g.region [-**dsplectwmn3bgau**] [**region**=*name*] [**rast**=*name*[,*name*,...]] [**rast3d**=*name*] [**vect**=*name*[,*name*,...]] [**3dview**=*name*] [**n**=*value*] [**s**=*value*] [**e**=*value*] [**w**=*value*] [**t**=*value*] [**b**=*value*] [**rows**=*value*] [**cols**=*value*] [**res**=*value*] [**res3**=*value*] [**nsres**=*value*] [**ewres**=*value*] [**tbres**=*value*] [**zoom**=*name*] [**align**=*name*] [**save**=*name*] [--**overwrite**] [--**verbose**] [--**quiet**]

Flags:

These are all parameters that you can provide to manage the boundary in different ways in this case.
If you scroll down the manual page, you will see what they all mean

-d
Set from default region

-s
Save as default region
Only possible from the PERMANENT mapset

-p
Print the current region

-l
Print the current region in lat/long using the current ellipsoid/datum

-e
Print the current region extent

Secure https://grass.osgeo.org/grass64/manuals/g.region.html

Parameters:

region=*name*
Set current region from named region

I used this last week to switch between WARegion and VoltaRegion

rast=*name[,name,...]*
Set region to match this raster map

rast3d=*name*
Set region to match this 3D raster map (both 2D and 3D values)

vect=*name[,name,...]*
Set region to match this vector map

3dview=*name*
Set region to match this 3dview file

n=*value*
Value for the northern edge (format dd:mm:ss{N|S})

I used these last week to set the extents of my VoltaRegion

s=*value*
Value for the southern edge (format dd:mm:ss{N|S})

e=*value*
Value for the eastern edge (format ddd:mm:ss{E|W})

w=*value*
Value for the western edge (format ddd:mm:ss{E|W})

t=*value*
Value for the top edge

b=*value*
Value for the bottom edge

rows=*value*
Number of rows in the new region

cols=*value*
Number of columns in the new region

https://grass.osgeo.org/grass64/manuals/g.region.html

Parameters:

region=*name*
Set current region from named region

rast=*name[,name,...]*
Set region to match this raster map

We'll use this one now to set the region to match our input raster maps

rast3d=*name*
Set region to match this 3D raster map (both 2D and 3D values)

vect=*name[,name,...]*
Set region to match this vector map

3dview=*name*
Set region to match this 3dview file

n=*value*
Value for the northern edge (format dd:mm:ss{N|S})

s=*value*
Value for the southern edge (format dd:mm:ss{N|S})

e=*value*
Value for the eastern edge (format ddd:mm:ss{E|W})

w=*value*
Value for the western edge (format ddd:mm:ss{E|W})

t=*value*
Value for the top edge

b=*value*
Value for the bottom edge

rows=*value*
Number of rows in the new region

cols=*value*
Number of columns in the new region

Note: Some parameters are different in GRASS7!!!

ALWAYS check the manual page for the version of GRASS you are using

Secure https://grass.osgeo.org/grass70/manuals/g.region.html

Parameters:

region=*name*
: Set current region from named region

raster=*name[,name,...]*
: Set region to match raster map(s)

raster_3d=*name*
: Set region to match 3D raster map(s) (both 2D and 3D values)

vector=*name[,name,...]*
: Set region to match vector map(s)

n=*value*
: Value for the northern edge

s=*value*
: Value for the southern edge

e=*value*

```
bash-3.2$ g.list type=rast
----------------------------------------------
raster files available in mapset <Mapset4>:
W13B3 W13B4 W13B5
```

g.list type=rast tells me which raster files I have in my mapset

```
----------------------------------------------
bash-3.2$ g.region rast=W13B3
```

g.region rast=W13B3 sets my region settings to match the raster W13B3

```
bash-3.2$ g.region -p
projection: 1 (UTM)
zone:       30
datum:      wgs84
ellipsoid:  wgs84
north:      1404750.75
south:      1358552.25
west:       644940.75
east:       712628.25
nsres:      28.5
ewres:      28.5
rows:       1621
cols:       2375
cells:      3849875
bash-3.2$
```

g.region –p tells me everything about by new (current) region settings

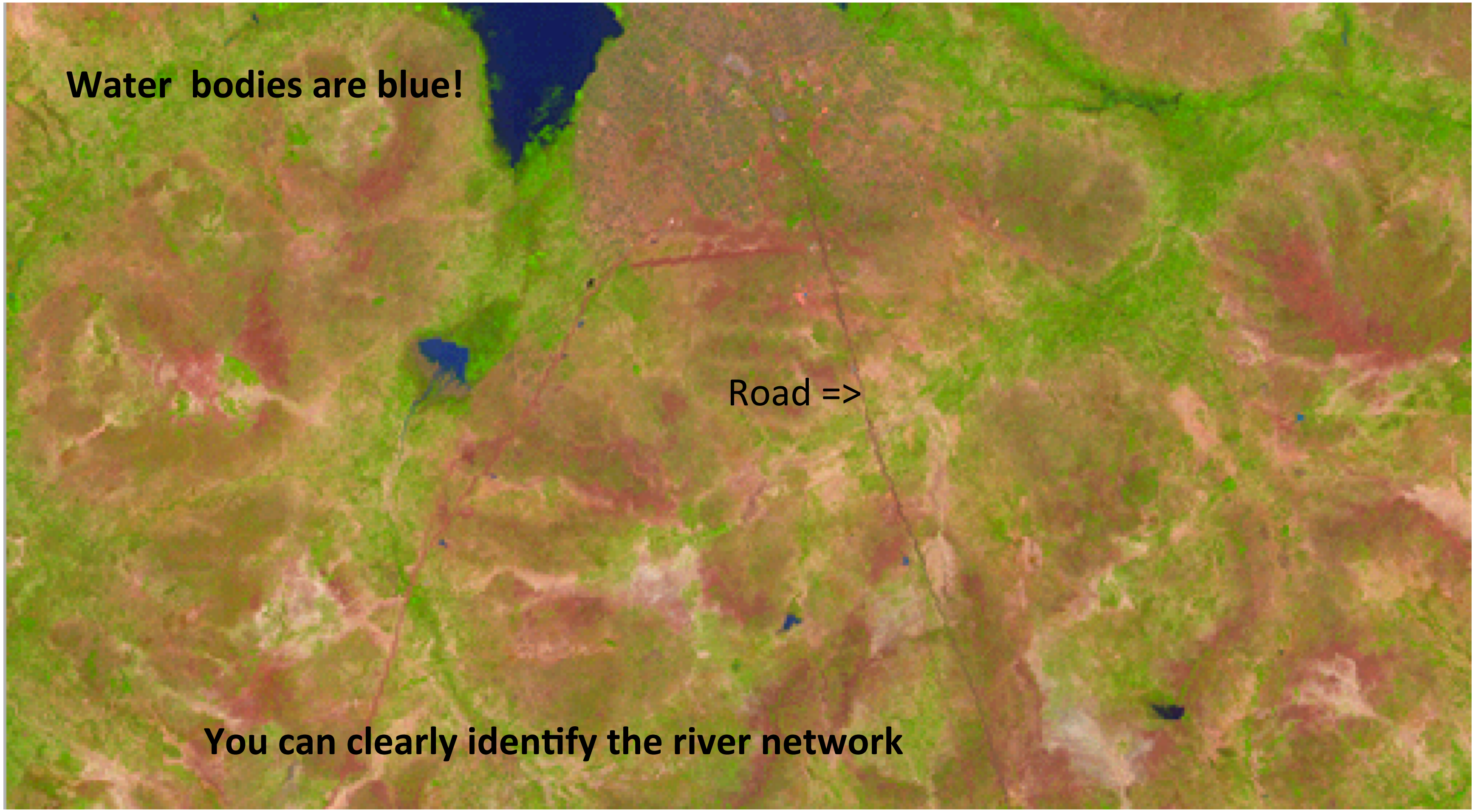
Here are the extents

The resolution is 28.5meters

And, I have the same number of rows and columns as I saw when I looked at the raster with QGIS

False color image (use **r.composite**):

Test the different RGB assignments to find one that allows you to identify the water bodies & other natural and man-made objects:



False color image (use **r.composite**):

Here, I've used Red=Band5, Green=Band4, Blue=Band3

For calculating NDVI, note that your rasters contain integer data.

You have to make your result a floating point value. See this note on the manual page for r.mapcalc which you can also look at in GRASS by clicking on the Manual tab:

Module: r.mapcalc

Options | Output | Manual

Floating point numbers are allowed in the expression. A floating point number is a number which contains a decimal point:

```
    2.3    12.0    12.    .81
```

Floating point values in the expression are handled in a special way. With arithmetic and logical operators, if either operand is float, the other is converted to float and the result of the operation is float. This means, in particular that division of integers results in a (truncated) integer, while division of floats results in an accurate floating point value. With functions of type * (see table above), the result is float if any argument is float, integer otherwise.

Note: If you calculate with integer numbers, the resulting map will be integer. If you want to get a float result, add the decimal point to integer number(s).

If you want floating point division, at least one of the arguments has to be a floating point value. Multiplying one of them by 1.0 will produce a floating-point result, as will using float():

```
    r.mapcalc "ndvi=float(lsat.4 - lsat.3) / (lsat.4 + lsat.3)"
```

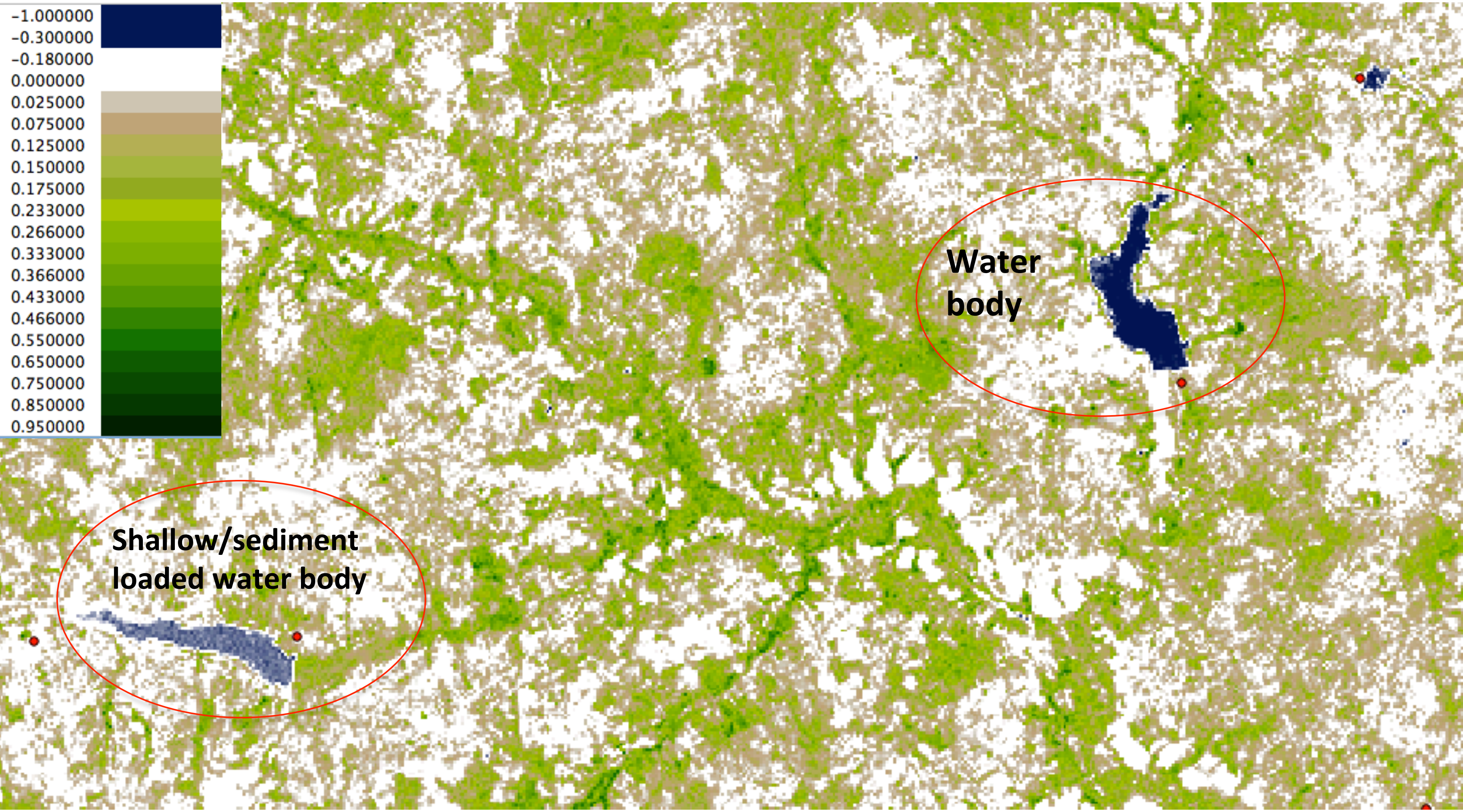
NULL support

NDVI map (NDVI calculated with **r.mapcalc**)

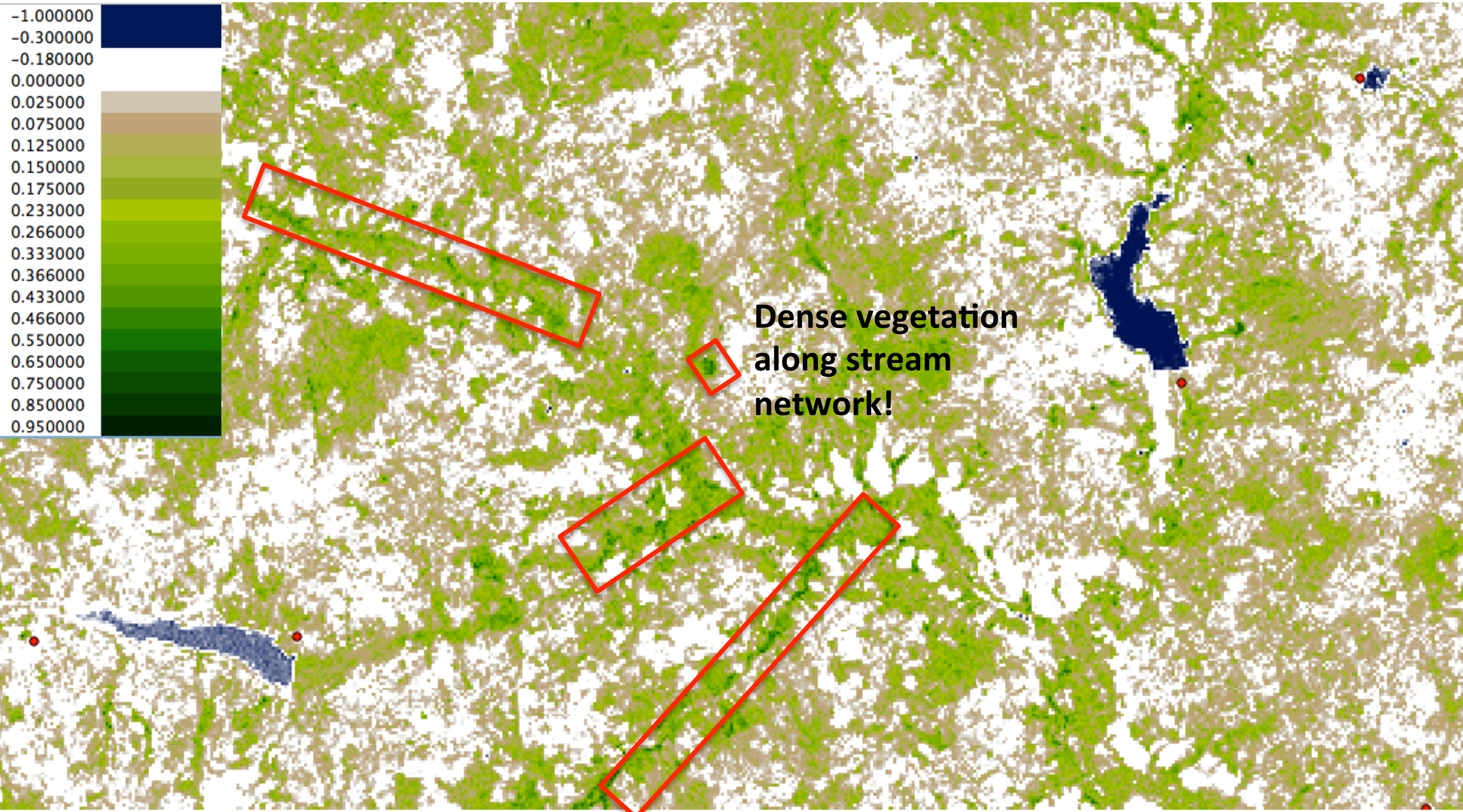
The landsat bands contain integers, so you need to include “float” to values of NDVI:
r.mapcalc “ndvi=float(W20B4-W20B3)/(W20B3+W20B4)”

Use r.colors.table to set the colormap to the “Normalized Difference Vegetation Index” colormap

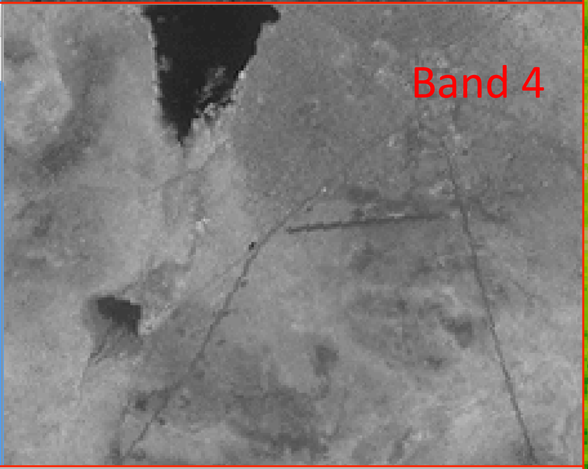
NDVI map (NDVI calculated with **r.mapcalc**)

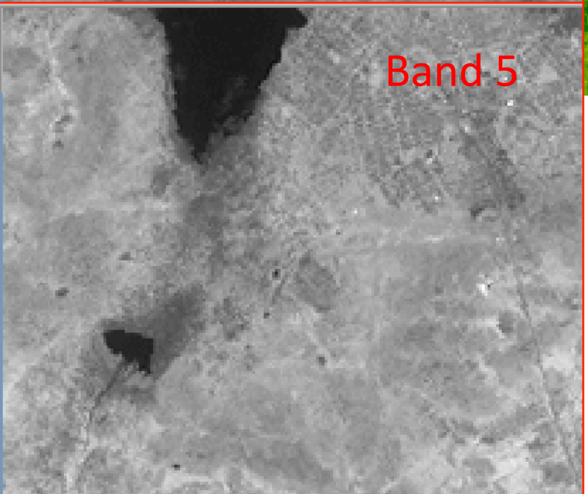


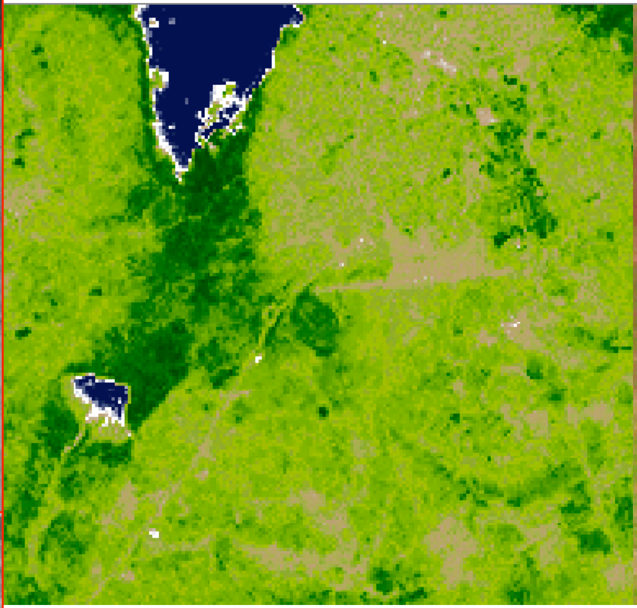
NDVI map (NDVI calculated with **r.mapcalc**)



Use NDVI, individual bands and composite to identify classes:







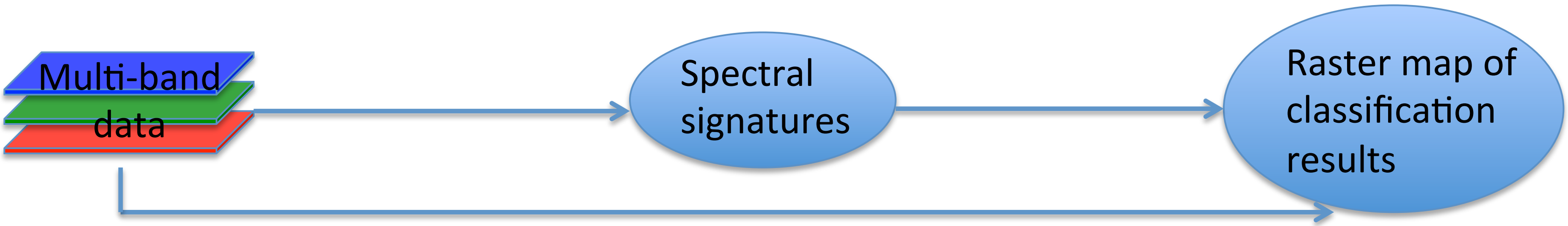
Some features are clearer looking at one of the bands than the composite or NDVI!

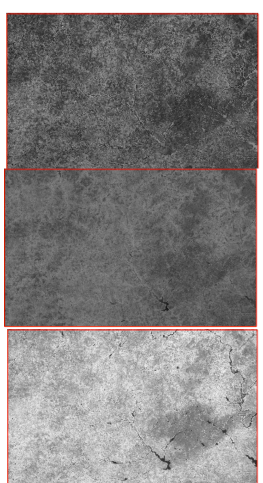
Classification using GRASS:

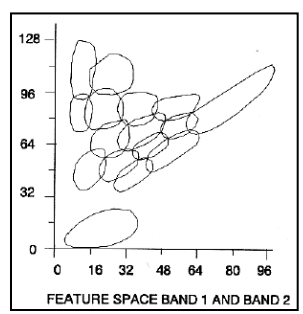
We have 3 bands of data, and we want to make a classification.

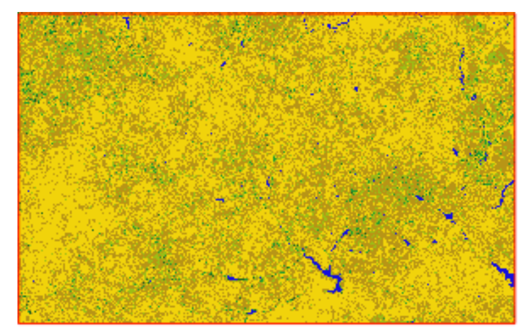
There are two main steps:

1) We need to find the spectral signatures
2) We use these spectral signature to classify each grid cell based on the values in the 3 bands





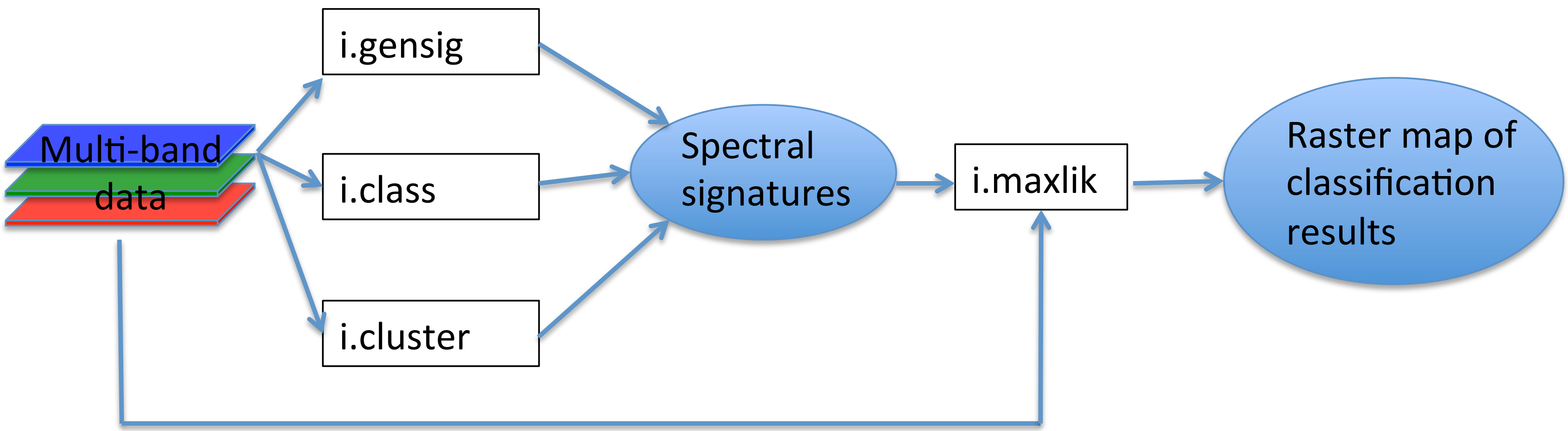


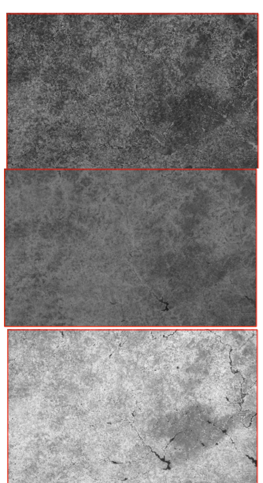


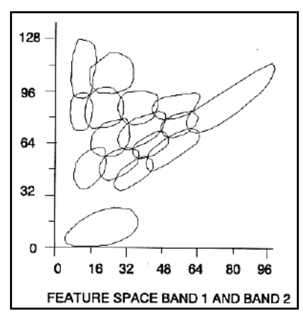
Classification using GRASS:

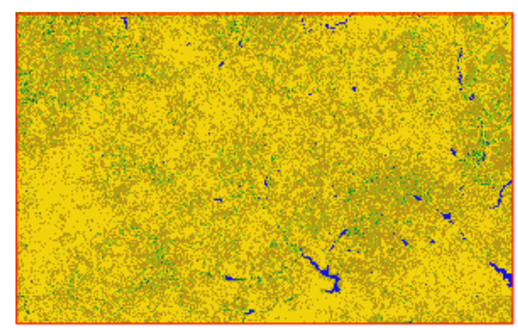
In GRASS there are three different tools to generate spectral signatures.

1) **i.cluster** is used for unsupervised classification … i.e. you let GRASS work it all out!
2) **i.class** and **i.gensig** are supervised classification, i.e. you contribute your expert knowledge to assist the classification. We want to do supervised classification



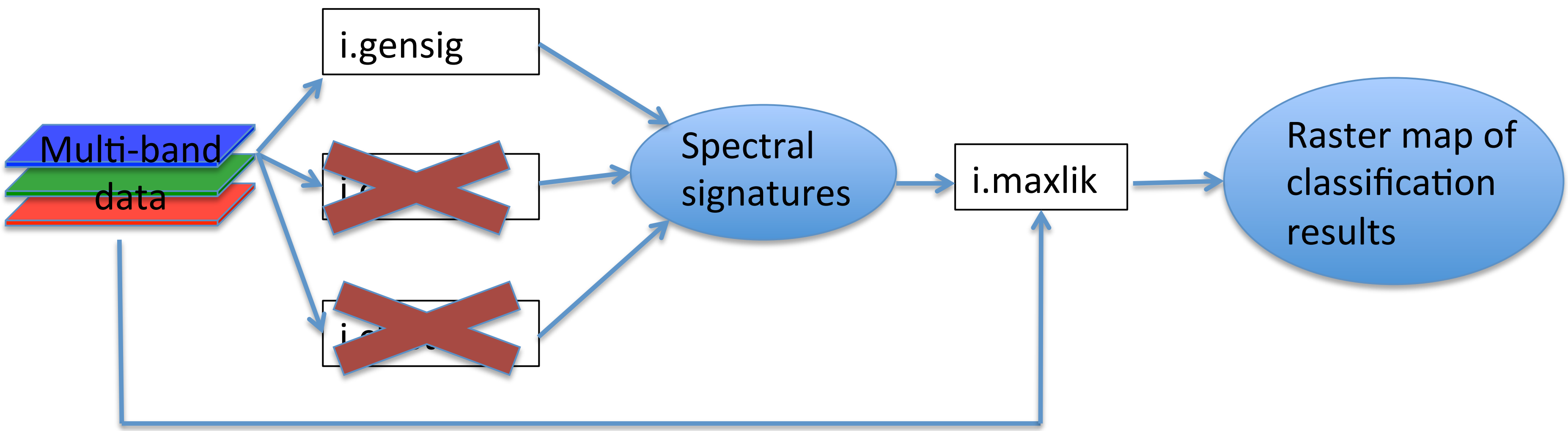


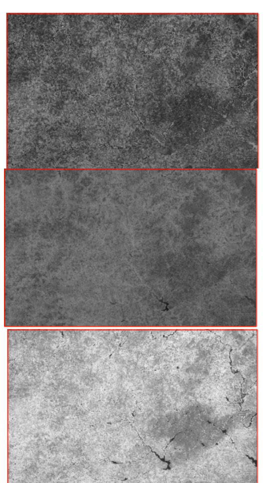


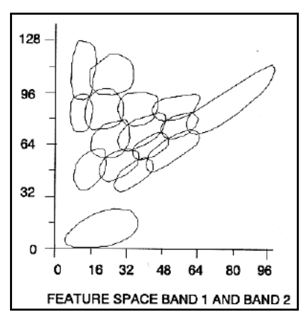


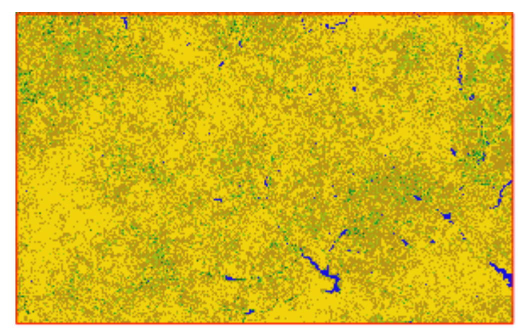
Classification using GRASS:

i.class is an interactive tool, but it's interactive when you are using GRASS as stand-alone software … not with QGIS, so we won't use it here.
i.gensig is a non-interactive for supervised classification, which is just what we need!









Training map for i.gensig

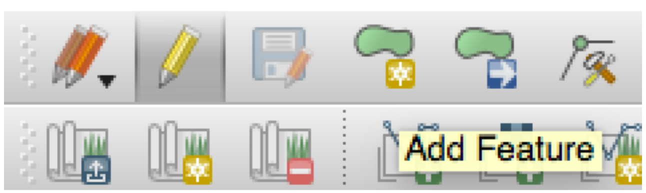
i.gensig generates the spectral signature file. You need to provide it with a raster training map.

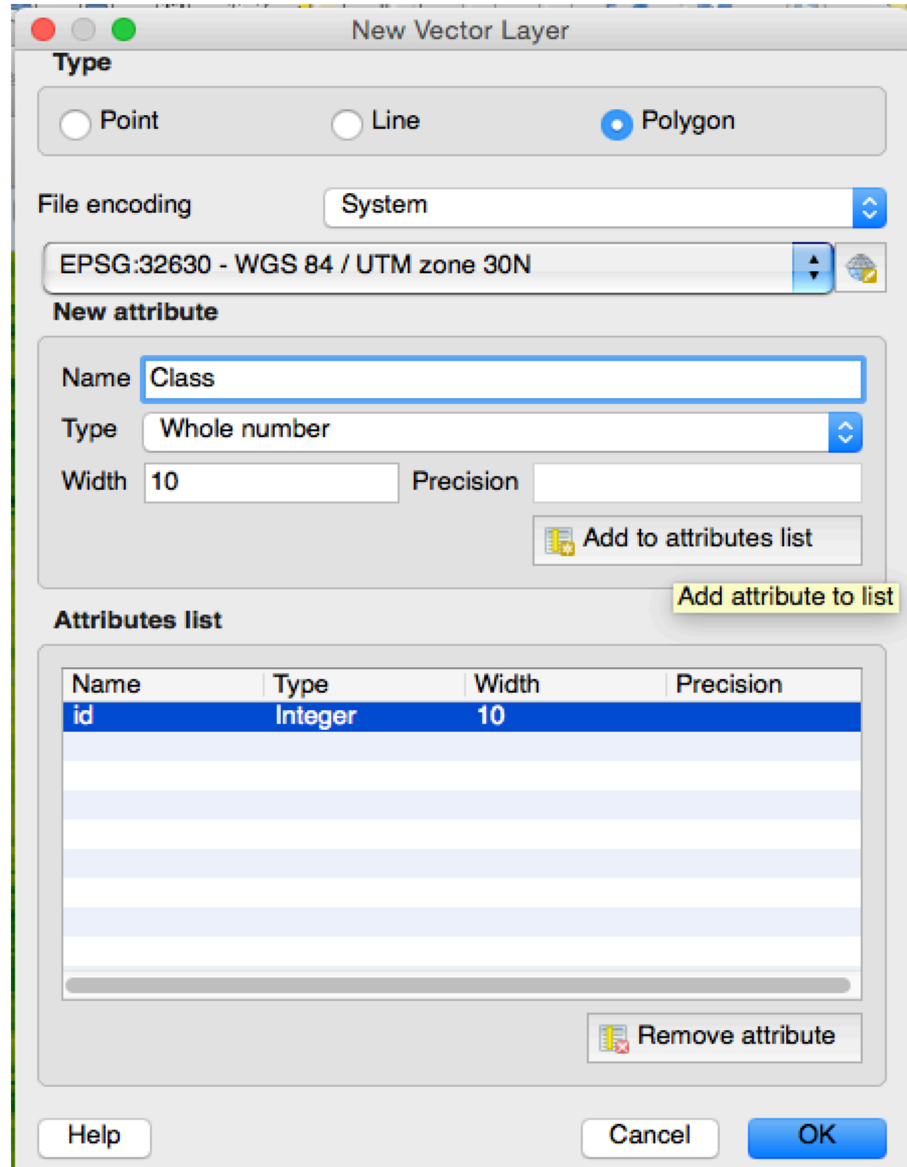
It is easiest to do this bit in QGIS:

1) Create a new polygon shapefile (Layer>Create Layer > New shapefile layer)

2) You can see that I'm adding a field called "Class" that will contain whole numbers

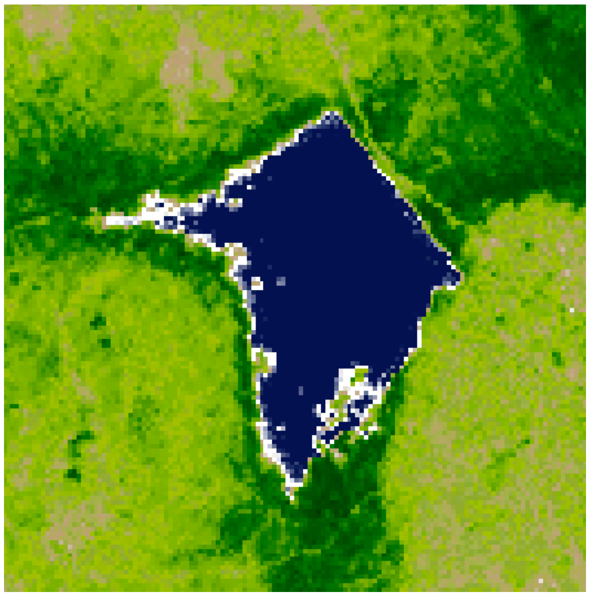
3) Once your layer is added to your project, toggle "edit on" and use "add feature" to draw some polygons where you think you have water.



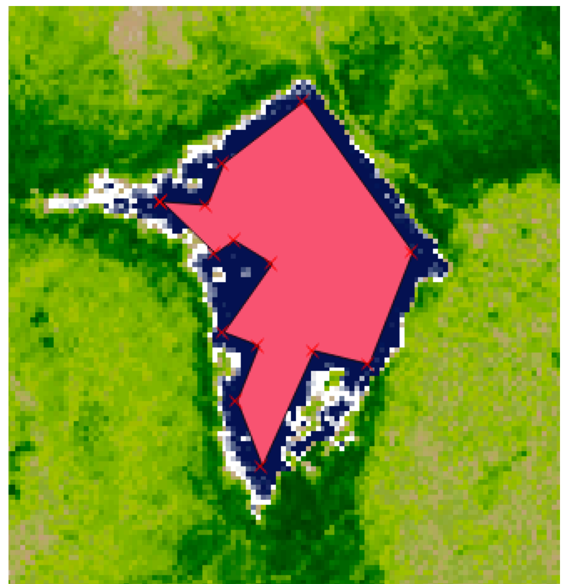


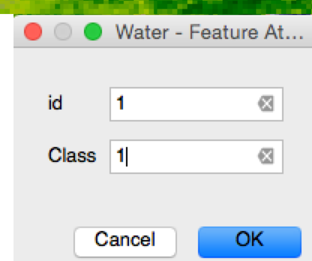
Training map for i.gensig

On my ndvi map, this looks like water:



Draw a polygon on it. When prompted, make sure to give each polygon a unique id and the class. Here, I'm calling water class 1.





In the attribute table below, you can see that each polygon has a unique id and "class" is 1 for all of them so far.

Don't forget! You can use your color composite too and your individual bands if your ndvi map just looks all green!

Training map for i.gensig

Follow the same procedure for other classes, but giving them a different value for class. Expect to have ~7-8 classes. For example, you might have:

Cover type	class
water	1
Bare soil	2
Light veg	3
Medium veg	4
Dense veg	5
urban	6
farmland	7

When you are finished, add your training shapefile to GRASS and convert it to a raster. This is the raster training map required by i.gensig.

i.group

The other thing you need to do before you run **i.gensig** is to use **i.group** to put your three bands of data together into a group.

Something like:

i.group group=landsat subgroup=landsat input=w27B3,w27B4,w27B5

i.gensig

https://grass.osgeo.org/grass64/manuals/i.gensig.html

You can only run i.gensig from the command line.
Use the manual page for help!

NAME

i.gensig - Generates statistics for i.maxlik from raster map.

KEYWORDS

imagery, classification, supervised, MLC

SYNOPSIS

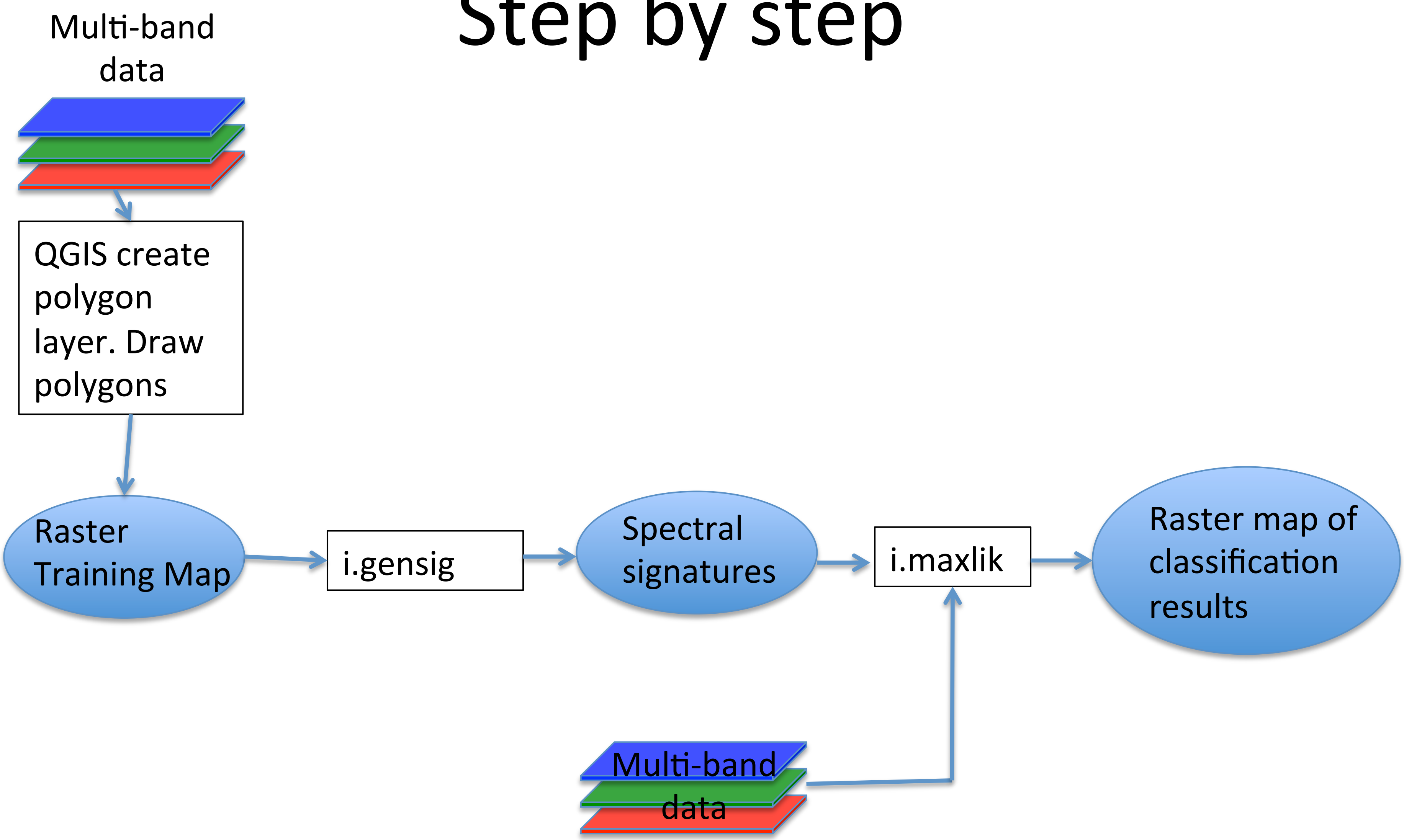
i.gensig
i.gensig help
i.gensig trainingmap=*name* **group**=*name* **subgroup**=*name* **signaturefile**=*name* [**--verbose**] [**--quiet**]

Parameters:

trainingmap=*name*
    Ground truth training map
group=*name*
    Name of input imagery group
subgroup=*name*
    Name of input imagery subgroup
signaturefile=*name*
    Name for output file containing result signatures

DESCRIPTION

Classification using GRASS: Step by step



grass.osgeo.org/grass65/manuals/i.maxlik.html

NAME

i.maxlik - Classifies the cell spectral reflectances in imagery data.
Classification is based on the spectral signature information generated by either i.cluster, i.class, or i.gensig.

KEYWORDS

imagery, classification, MLC

SYNOPSIS

i.maxlik
i.maxlik help
i.maxlik [-**q**] **group**=*name* **subgroup**=*name* **sigfile**=*name* **class**=*name* [**reject**=*name*] [--**overwrite**] [--**verbose**] [--**quiet**]

Flags:

-q
    Run quietly
--overwrite
    Allow output files to overwrite existing files
--verbose
    Verbose module output
--quiet
    Quiet module output

Parameters:

group=*name*
    Name of input imagery group
subgroup=*name*
    Name of input imagery subgroup
sigfile=*name*
    Name of file containing signatures
    Generated by either i.cluster, i.class, or i.gensig
class=*name*
    Name for raster map holding classification results
reject=*name*
    Name for raster map holding reject threshold results

You will use i.maxlik to perform the actual classification.

You should read the manual page for a full description.

DESCRIPTION

i.maxlik is a maximum-likelihood discriminant analysis classifier. It can be used to perform the second step in either an unsupervised or a supervised image classification.

After you have your (raster) classification map

- If the results from your classification are not satisfactory (e.g. you don't classify all the water bodies as small reservoirs, or you classify some soil as reservoir) then you need to add additional classes.

- When you think it looks good, check that you have identified the reservoirs. Pay particular attention to reservoirs that may have suspended sediments … they are often misclassified as soil. You may need to refine your classes and generate new spectral signature files.

- Use **r.mapcalc** to make a raster that is null everywhere and 1 in all the water grid cells.
- Use **r.clump** to group contiguous cells into an area with a unique category.
- Use **r.to.vect** to convert this raster map to a vector map.

Now you are ready to hand-check your results …

Hand-checking and final calculations

For interacting with your polygon layer, I recommend using QGIS. Only proceed to hand-checking when you know that all water bodies have been identified (look in particular at shallow water/water with sediment). It is easy to remove erroneously classed grid cells, but difficult to add them!

1) Export your vector (GRASS)map to a regular shapefile using **v.out.ogr**, and add this shapefile to your project.
2) Work with the attribute table to select and delete polygons that are small (e.g. one pixel!)
3) Use select single feature to remove huge water bodies (e.g. if you happened to get part of a large lake or a river or a burn scar).
4) If your reservoir is split in two, use "single parts to multi-parts".
5) Select features by freehand or rectangle to remove collections of spurious water bodies.
6) Use field calculator to calculate areas (and volumes).
7) Save layer as **comma separated values** to export to matlab/python for further analysis