MAPPY Imagery Classification Tutorial

1) Installation of Spfeas

2) Understanding the Inputs and Outputs

3) How to run Spfeas yourself

4) Stacking

5) Drawing Training Sites

6) Sample Land Cover classes

7) Classifying the image

8) Clean up

9) Reclassify

10) Accuracy Assessment

## Installation of SpFeas

To get started you will need to create an account on GitHub (<https://github.com/>). Once you log in, navigate to jgrss (Jordan Graesser’s) repository called spfeas. Once you are there, your screen should look like this:

|  |
| --- |
|  |

Click on ‘notebooks’ (highlighted with a red box)

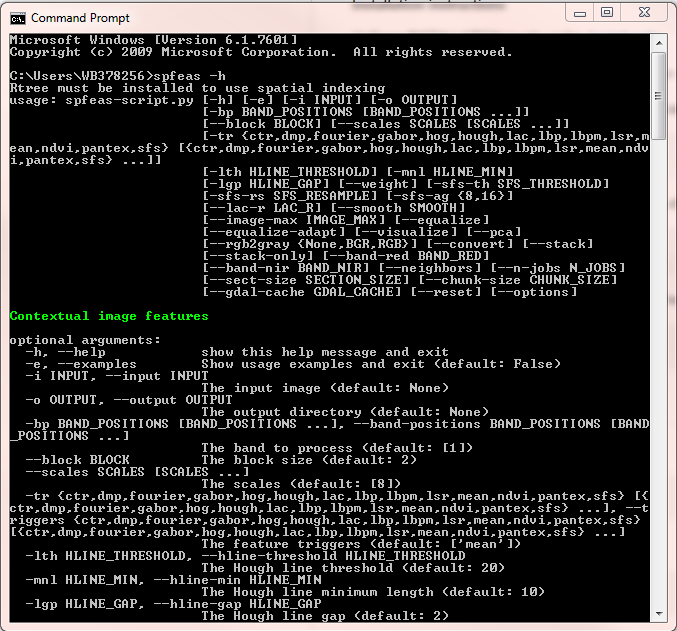
You will see two notebooks that have lots of helpful information. Start with [INSTALLATION.ipynb](https://github.com/jgrss/spfeas/blob/master/spfeas/notebooks/INSTALLATION.ipynb)

Follow the instructions in this first notebook and at the end you will have SpFeas installed.

*Note: SpFeas is a Python library for processing spatial (contextual) image features and image classification. You will be using this library to derive many different layers from the imagery. Once you have created all the layers you will be stacking them, and using them to run the classification, but for now we will focus on just get everything up and running.*

Once you have completed the installation – type in the command prompt window: spfeas –h

If the screenshot below appears – you’ve done it right!



To understand this help menu, look at the **–i** and **--input** below Contextual image features.

The **–i** is an abbreviated command you can type or if you want to spell it out, start with two dashes and type the full word like this: **--input**. Ignore the uppercase text, as that is just automatically generated and slightly confusing. After the tab space, you will see a description of the command in lower case, for input you will see “The input image”. This means that every time you type –i or --input you will need to follow it by the path and name of the input image.

Once a command has been written the computer will look for a parameter. The parameter options are listed in lower case (ignore the uppercase text). For example, look at **–tr** the optional parameters are in parenthesis, so after **–tr** you can type **mean pantex ndvi** etc. If you look at

**--block** there are no preset options, but the default is numeric (2), which means if you don’t want a block size of 2 you need to type **--block 8** or **--block 16** etc.

# Understand the Inputs and Outputs

This script (Spfeas) is going to be run on the imagery and generate a large stack of derived layers. For example, it will calculate the image’s NDVI and then it can calculate the pixel mean, or you can tell it more complex statistic. We refer to these derived layers as Spatial Features (hence the name SpFeas). Below is a brief summary of some of the spatial features:

|  |  |
| --- | --- |
| fourier | This transformation is often used to remove noise from an image, such as striping and spots. But we will use it to detect high frequency of lines or a low frequency of lines. |
| dmp  (Differential morphological profile) | This is used to measure how uniform in size the objects in the image are. It uses a multi scale technique to support the identification of objects and subobjects. |
| gabor | This filter extracts edges and orientations of edges |
| hog  (histogram of oriented gradients) | This calculates the orientation and magnitude of the shades of the image. |
| lac | This permits to distinguish spatial patterns through the analysis of their gap distribution in different scales. Low-lacunarity geometric objects are homogeneous because all gap sizes are the same, whereas high-lacunarity objects are heterogeneous. |
| lbp  (local binary pattern) | The Local Binary Pattern is a simple texture algorithm that examines pixel relationships within a radius around a center pixel. The center pixel-­‐to­‐surrounding pixels relationships are recorded as binary counts, then stored in a histogram at each scale. This LBP trigger returns the mean, variance, skew, and kurtosis of the concatenated LBP histogram. |
| mean | This is simply the local mean, at each scale, of the input image’s Digital Numbers (i.e. pixel values). |
| ndvi | NDVI is a vegetation index that measures the 'lushness' of vegetation. Its value ranges from -1 to +1 with higher values representing healthier/greener vegetation. |
| pantex | A textured derived index used to extract human settlements - The basic idea of PanTex is based on the fact that buildings cast a shadow that is producing high local contrast. Therefore, In urban areas contrast is typically high in all directions, and pantex value is high |
| sfs (Strucutral Feature Sets) | This indicator measures the directions of lines in an image. |

It’s important to keep in mind these spatial features can be derived at different scales and on different bands. The figure below shows an image with white lines depicting the ‘blocks’ and the red lines depicting the ‘scale’. In this example the blue block is being processed. If you were to run the Spfeas -tr mean you will be calculating the average number of pixels in each block (white lined) and the new resolution of the output would be the size of the blue block. Another way to think about it, is that the block size represents the output’s unit of measure

|  |
| --- |
|  |

As a result, the file names of the spatial features (the outputs) will look something like this:

|  |
| --- |
|  |

So by reading this file name, you know:

|  |
| --- |
| **Location**: Colombo, Sri Lanka **Image is panchromatic**  **spatial feature:** Pantex  **Band:** 1 **Block size** 8 (8 pixels by 8 pixels  **Scale:** 32 **statistic** is 1 (i.e mean) |

# How to run Spfeas

Create a local folder where you are going to store your images, name it “**imagery**”.

\

Then create a second folder where you are going to store your derived layers, or output layers. Name it “**spatial\_features**”

Go to GitHub.com and click on the other notebook [examples.ipynb](https://github.com/jgrss/spfeas/blob/master/spfeas/notebooks/examples.ipynb). Then open the command prompt (by clicking on your windows start and searching for command). Start to run a few of the commands that are written out on the [examples.ipynb](https://github.com/jgrss/spfeas/blob/master/spfeas/notebooks/examples.ipynb) page.

Open up notepad (or another text editor)

Because it’s difficult to edit in the command prompt, you will create your commands in notepad and then copy and paste your commands into the command prompt.

Replace the text in red with your computer’s path and image name. Throughout this tutorial, red text will represent an example and need to be replaced with your own path url or image name.

On my computer an example of running the spatial feature **mean** looks like this:

spfeas -i C:\Users\WB378256\Desktop\test\imagery\huye.tif –o C:\Users\WB378256\Desktop\test\spatial\_features -tr mean

Paste your command into the command prompter and hit return. If you are worried about memory issues add “--chunk-size 128 --sect-size 1024 --gdal-cache 128” to the end of your command

Once this has finished, you will receive a summary like this: 

Navigated to your output folder “spatial\_features” you should now see a layer that has calculated the mean of the pixels in each block at scale 2 for band 1.

On my computer it looks like this: 

In this example SpFeas picked band 1 scale 8 and block 2. If you don’t specify these parameters, SpFeas will pick the default. The larger the block size, the faster the spatial features will be calculated.

Now that you have generated your first spatial feature (mean) and understand the basic steps, go ahead and run the other examples described in [examples.ipynb](https://github.com/jgrss/spfeas/blob/master/spfeas/notebooks/examples.ipynb)

**Here is a more complex example broken down:**

**spfeas** –i C:\Desktop\Cordoba\image.tif **–o** C:\Desktop\Cordoba\features –**tr mean fourier**

--**scales** 16 32 64 **--block** 16 --**rgb**

**spfeas** (tells the computer you are creating spatial features)

**-i** (tells the computer the location and name of the input, so drag and drop the tif image into the command prompt)

**-o** (tells the computer the location where you want all of the features to be created, so copy and paste the path url of the features folder)

**-tr** (here you need to specify the triggers or features you want run, so type mean fourier etc.)

**--scales** (this will tell the size of the window around the block – within this window you calculate the features. For example if the scale is 16, that means you are calculating the fourier using 16x16 pixels and assigning that value to the center pixel (ie block).

**--block** (this is the resolution of the output and we typically do 8 or 16

**--rgb** (this makes use of all spectral bands at once, not just band 1 or band 2 etc)

**As you create more spatial features – use the notes below for reference**

* SpFeas is case sensitive.
* It will not work if there are spaces in the file path or name.
* Small dashes “-” in the file paths/names may confuse SpFeas since the parameters are delineated by dashes. Underscores work best.
* Erdas Imagine (.IMG) or GeoTIFF (.TIF) file formats are preferred.
* When running ndvi, add **--band-red 1** because the imagery is stored as RGB (red/green/blue)
* Make sure your imagery is stored in an 8-bit data range (i.e., pixel values 0-255)
* **If you are having memory problems, add (** --chunk-size 128 --sect-size 1024 --gdal-cache 128) to the end of every command.
* **If you are having problems running the scripts and commands, download the latest MpGlue tarball zipped file and upgrade it.** <https://github.com/jgrss/mpglue/releases>

1) Download the new and most updated MpGlue tarball

2) In Command prompt: cd \Downloads

3) pip install --upgrade MpGlue-0.0.5.tar.gz

**Parameters**

-i = input image with file path and extension

-o = output folder destination

* Best if separated by city and year (separate folders called Nairobi\_2010, Nairobi\_2000, Kigali\_2010, etc.)

--block = block size

* + - Output resolution of each spatial feature layer
    - May only compute one block size at a time
    - If we were aiming to get output resolutions closest to 15m, we would like to use a block size of 8 (input resolution of 2.5m X block size 8 = 20m)

--scales = scale sizes

* + - Moving window size. To compute low-level feature statistics at each pixel block (bi), we consider different neighborhood scales at and around each block, compute the statistic at each scale, and write the statistic to pixel block (bi). For instance, if you do block 8 and scale 8 you will be only using that pixel value within that block. As it goes larger, scales 16 32 64 128, it will analyze more into the neighbor pixel values. In the image below, you can see how square 3 (scale 64) is taking the value of all the pixels within it, in order to compute features (for example, mean). It will assign to the blue block b, the mean of all the pixel values within square 3. If scales 32 is entered, it will analyze the pixel values of the square 2, and so on with square 1 and scales 16.



* + - Able to compute multiple scale sizes at a time.
    - Scale sizes must increase in regular octaves (block size = 8m, then scales must be 8, 16, 32, 64, 128, etc.) --block 8 --scales 8 16 32 64 128

-tr = spatial feature to compute

* + - Able to compute multiple spatial features at a time (separate by comma)
    - Each spatial feature returns a different number of layers
      * mean = 1 layer for each block/scale
      * fourier = 2 layers for each block/scale
      * hog = 4 layers for each block/scale
      * pantex = 1 layer for each block/scale
      * lbpm = 4 layers for each block/scale
      * gabor = 16 layers for each block/scale
      * dmp= 1 layer for each block/scale
      * lac= 1 layer for each block/scale
      * lsr = 3 layers for each block/scale
      * ndvi = 1 layer for each block/scale
      * evi2 = 2 layers for each block/scale
      * grad = 2 layers for each block/scale
      * lbp = 62 layers for each block/scale
      * saliency = 2 layers for each block/scale
    - Note: sfs needs to be computed differently than all other spatial features
      * The sfs feature will not automatically use an entire scale window, so there is no need to give multiple scales.
      * Use a large scale. It is best to use the largest scale used in other features
      * Use a small block size and with resampling.
      * In the example below we are computing sfs with a block size of 2x2 at a scale of 128x128. After processing, the sfs features will be resampled to 16m x 16m cell size.
      * Example: spfeas -i /your\_image.tif -o /your\_output\_directory -tr sfs --rgb --block 2 --scales 128 --sfs-rs 16

-bd = band number to compute spatial feature on

* + - Panchromatic imagery only has one band, therefore band = 1
    - Multispectral imagery: use band number for NDVI
      * NDVI: If band combo is RGB, -bd 1
      * NDVI: If band combo is BGR, -bd 3

--rgb: only use this if you are processing Multispectral imagery, this replaces the -bd parameter and tells SpFeas to use the entire visible (RGB) spectrum.

# Stacking

Once you have all of your spatial layers created you need to bundle them together in a virtual stack. You create a stack by adding the command “--stack-only” to your command. You will also have to include the block size, scales and band position (or RGB) that were introduced when processing the spatial features.

spfeas -i C:\Users\WB378256\Desktop\test\Argentina\_Mappy\_training\imagery\huye.tif -o C:\Users\WB378256\Desktop\test\Argentina\_Mappy\_training\spatial\_features -tr mean pantex hog fourier gabor --stack-only --block *8 (or 16)* --scales *8 16 32 64 128*

In case you want to process the spatial features and stack, it will create and run the spatial features indicated and stack them all together at one step.

spfeas -i C:\Users\WB378256\Desktop\test\Argentina\_Mappy\_training\imagery\huye.tif -o C:\Users\WB378256\Desktop\test\Argentina\_Mappy\_training\spatial\_features -tr mean pantex hog fourier gabor lbpm dmp sfs lac ndvi --stack --block *8 (or 16)* --scales *8 16 32 64 128*

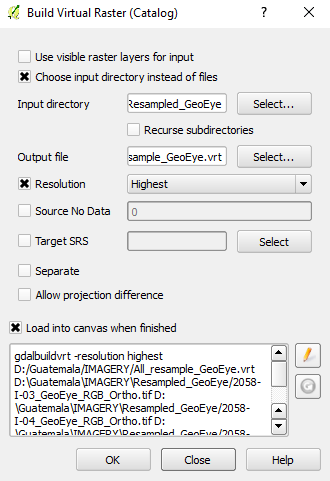
The output file in both cases is a Virtual Stack of Features in the format of atext file and a vrt (includes dmp, fourier, gabor, hog, lbpm, mean, ndvi, pantex). Both the vrt and the text file created in this step are highlighted in red.

|  |
| --- |
|  |

**Stacking Images Instead of Features**

Instead of mosaicking images, multiple images can be “stacked” next to each other in a .vrt format. Virtual rasters can save considerable disk space, and QGIS 'sees' them as a single file. This is very useful for rasters that do not need blended together to create a new image file, i.e. just connected together in a mosaic.

In QGIS select: Raster-> Miscellaneous-> Build Virtual Raster (Catalog)



# Drawing training sites

**QGIS Directions**

Open Q-GIS, and add the image you want to classify. In this part are you going to be drawing small polygons over the land classes in order to ‘teach’ the classifier. For example, if you want a water class, you are going to have to draw small and disperse polygons over the rivers, oceans or lakes that you see in your image.

1. To get started click on New Shapefile Layer 
   1. Type: Polygon
   2. Selected CRS: WGS 84 / UTM Zone [XXX]
      1. Example: Cordoba is UTM zone 20S (EPSG: 32720)
      2. Make sure your training site layers are in the same projection as your image (UTM recommended)
   3. Okay
   4. Save as: [city]\_[year]\_training\_class[class]\_[initials].shp
      1. Example: Cordoba\_2013\_training\_class01\_amcs.shp
2. Highlight training shp in table of contents
   1. Toggle Editing on 
   2. Add Feature 
      1. Right click to end polygon
3. When finished training, toggle editing off, and save edits!
4. Dissolve all training sites of one class into one polygon
   1. Vector > Geoprocessing > dissolve
   2. Save as: Cordoba \_2013\_training\_class01\_dissolve\_amcs.shp
   3. Edit ID field to say 1 for class 1:  > Update all
   4. Toggle editing off and save edits
5. Repeat steps 2-7 for all other land cover classes
   1. Class 1 - Shanty Dense or Sparse
   2. Class 2 – All other buildings
   3. Class 3 – Vegetation
   4. Class 4 – Bare soil
   5. Class 5 – Water
   6. Merge all classes together
   7. Vector > Data management > Merge shapefiles to one
   8. Output: Cordoba \_2013\_training\_merge\_amcs.shp

# Sample land cover classes

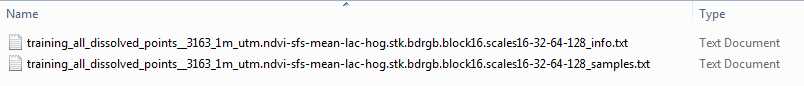
After creating the polygons shapefiles, we need to create a sample for each cell underlying a polygon features. MpGlue’s sample-raster tool will automatically convert a polygon vector file to a point file, and then sample the raster file data to a text file. Basically, it uses a polygon of land cover category to sample an image or feature stack.

In order to create the sample files, you need to print the following command:

sample-raster -s land\_cover\_samples.shp -i feature\_stack.vrt -o output\_samples\_dir

Above, -s is the input shapefile, -i is the input feature stack to sample, and -o is the output directory to save the sample text files to. There are no chunk parameters for sample-raster

The output files of this step will be two text files that will look like the screenshot below:



# Classify the image

From the sample text file, MpGlue will create and train a Random Forest (RF) classifier with 1000 (or 100, 500, etc) trees and save the model to a text file (RF\_model.txt). To learn more about this model read: <https://en.wikipedia.org/wiki/Random_forest>

This trained model is used to classify an image. The command that combines these two steps in one is as follows (note: you will have to create a name for the bolded text)

classify -i /path/feature\_stacked\_image.vrt -o /path/**output\_map.tif** -s /path/output\_samples\_dir/name\_of\_samples.txt --output-model /path/**RF\_model.txt** --classifier-info "{‘classifier’: ‘RF’,’trees’:1000}" --row-block 512 --col-block 512 --v-jobs -1

Example broken down:

**classify** (is the command)

**-i** (is the input, which in this case is the stacked features)

**-o** (is the output location and name of layer, so you paste the location of the output folder, and add the name you want the classified image to be)

**-s** (is the sample data, so drag and drop the long text file that was created during the previous sampling step, just make sure it is the txt file that ends in \_samples.txt

**--output-model** (this is the location and name you want to call the model, usually we name It RF for random forest and you can also add the number of branches, just make sure you add a name with .txt extension.

**--classifier-info** (this is defining the classifier and defining the parameters. In this case we are saying use a forest model and 100 branches, but you can change this number.

Lastly, if you are working in a 32bit environment you need to add 

This will help with memory issues

Along with the classified image, an accuracy assessment file of the Random Forest model will be created if –perc-samp is less than 1 (the default setting is 0.9). It will look like the screenshot below:

|  |
| --- |
|  |

To learn about other commands that change the classification parameters, type classify –h in the command prompt.

Here are some other options:

**--classifier-info "{‘classifier’:’AB\_EX\_RF’, ‘n\_estimators’:200, ‘trials’:50}”**

this will use an extreme booster random forest model that boosts the model 50 times

**--perc-samp-each .9**

takes 90% of the sampling from each class.

**--class-subs “{1:9, 5:.5}”**

this command will make the model use 90% of class 5’s points and 50% of class 5’s points.

**--recode-dict “{1:1, 2:1}”**

this will make the model consider the sample points in class 1 stay as class 1 and the sample points in class 2 merge with class 1. You can use this if you are having trouble distinguishing between two classes, like residential regular and residential sparse. This will recode them as the same class, without having to change the training sites attribute table.

**--classes2remove 8**

this will remove the class 8 from the classification, and not sample points from class 8 will be used in the model.

**\*Important**: This accuracy assessment does not represent the land cover accuracy of the classified raster.

# Clean-up

To improve the classified output raster, manual clean-up is necessary. Below you can find instructions to modify the land cover values of certain areas:

1. Add imagery and classified image to view
2. Create new shapefile > Polygon > UTM zone
   1. Save as: City\_YYYY\_cleanup\_class#\_yourinitials.shp
   2. Example: Conakry\_2013\_cleanup\_class1\_AMCS.shp
3. Toggle Editing on and Draw polygons around areas that should be changed to class1
4. Add field called “CID” and add the class # that it should be changed to for every polygon
5. Toggle editing off and Save Edits
6. Repeat for all classes

Tips:

* Spend about an hour zooming around to find big errors
* Things to keep an eye out for:
  + Areas of bright Soil may be misclassified as built-up >> cleanup as Other
  + Coastal sand/waves may be misclassified as built-up >> cleanup as other
  + Other residential/commercial areas may need to be cleaned up as well that the classification may have missed.

# Reclassify

Once the polygons that are going to clean up the classified image have been created, we need to recode them and reclassify them. In order to do so, there are two options available: 1) running the recode and reclassification tools, or 2) manually reclassifying the clean up areas and mosaicking the output rasters in ArcGIS.

1. **Reclassifying using MpGlue and recoding the clean up polygons.**

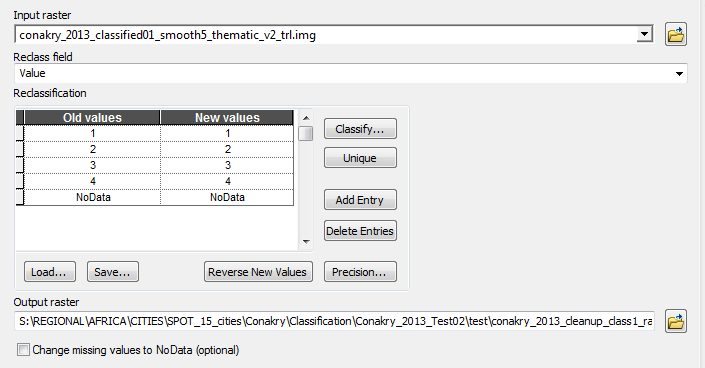
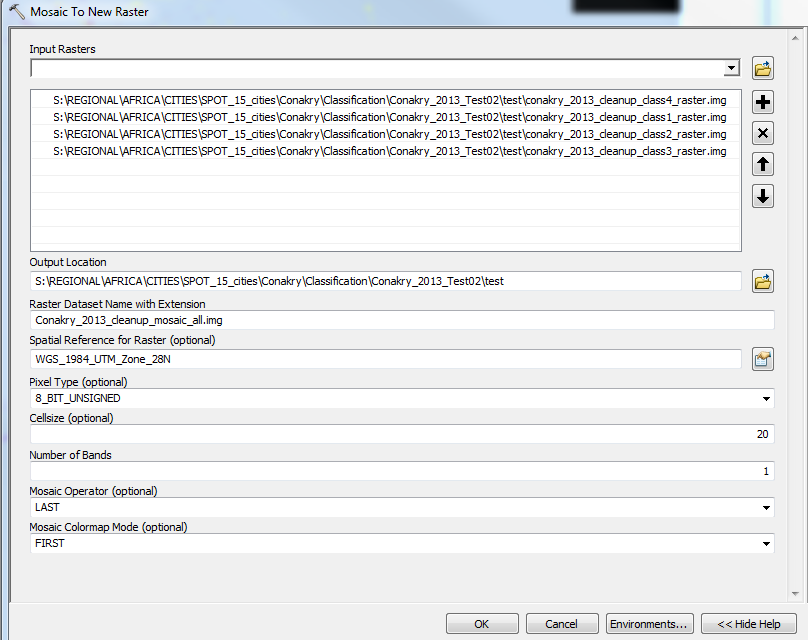
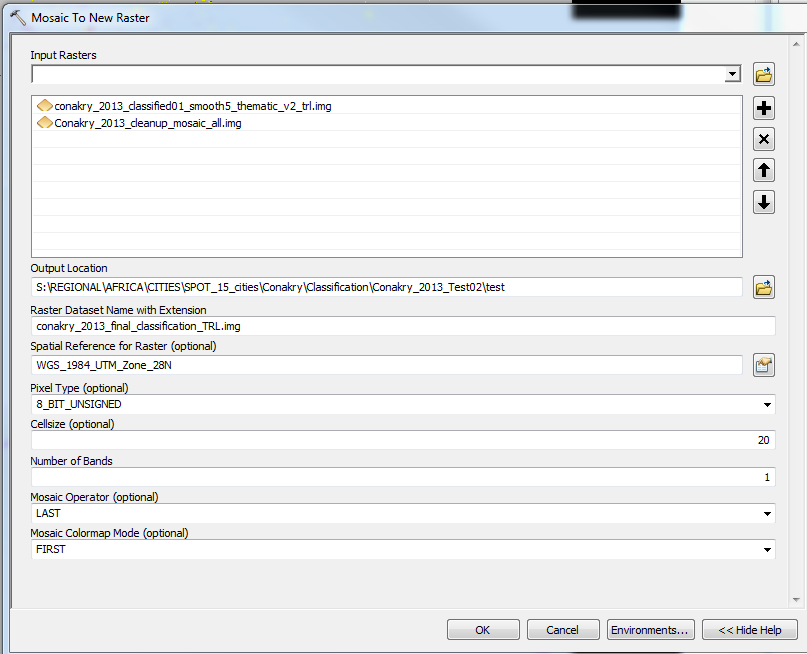
This tool assigns values to the pixels that fall within the polygons we create, reclassifying the classes into the class we specify. The command to enter is the following:

recode -p /polygon.shp -i /input\_map.tif -o /reclassified\_map.tif -c Id --rules "{1: {2:1}}"

* The -c parameter is used to tell the recode tool which polygon id column to use. The default if not added the parameter is id. If we want to take the value of a different named attribute we will need to enter -c + the name of the attribute.
* The --rules tell the recode tool where and what to reclassify.
* Walking through the recode rules: 1) The first key positions (class 1 in the command example) tell the recode tool which polygon feature Id’s to process. When we have the cleanup polygons merged in one feature, and the Id is 1 (i.e. informal), this is the number to introduce. ; 2) The second set of key positions (e.g., {2:1,3:1} )tell the recode rule which classes to reclassify. In this step, you might have to introduce all the possible values in order to change them all into 1 (informal). This is because it might not possible to know what pixel values our polygons are taking. Therefore, in the example of reclassifying polygons that should be informal and value 1, we should introduce the following: "{1: {2:1, 3:1, 4:1, 5:1}}" This command will assign all the pixels (value 2, 3, 4 and 5) within the polygons to informal and value 1.
* In feature Id 1, reclassify class 2 to 1.
* In feature Id 2, reclassify class 2 to 1.

1. **Manually reclassifying Raster for Manual Cleanup areas (ArcGIS only).**

For the manual reclassifications, see instructions as follow below:

1. Reclassify Tool
   1. Input Raster: smoothed thematic classification
   2. Reclass Field: Value
   3. Click “Environments…” Tab > Raster Analysis
      1. Cell Size: Same as classification layer (must do this to keep the rasters snapped to each other)
      2. Mask: your Cleanup shapefile (ex: Conakry\_2013\_cleanup\_class1\_yourinitials.shp)
   4. Reclassification:
      1. New Values all must be changed to the class of the cleanup polygons (see below)
   5. Output raster: conakry\_2013\_cleanup\_class1\_reclassify\_yourinitials.img
   6. 
   7. Click Okay
   8. Repeat for all 4 cleanup classes.
2. Mosaic to new Raster
   1. Input Rasters: all four cleanup rasters
   2. Output: Conakry\_2013\_cleanup\_mosaic\_all\_yourinitials.img
   3. Spatial Reference: UTM zone
   4. Cell size: 20
   5. Number of bands: 1
   6. Click Okay
   7. 
3. Mosaic the classification and the cleanup files together
   1. Input the classification first, then the cleanup mosaic image.
   2. Same parameters as above
   3. 

**\*Smooth the Classified image (Erdas).** In order to get rid of anomalous pixels, follow these steps:

1. Erdas > Raster > Thematic > Neighborhood
2. 5x5 Majority Filter: *C:\Users\wb486933\Secondary\_Cities\Mappy\_Classification\muhanga\muhanga\_classified\_smooth5.img*

**\*Make raster a thematic layer (Erdas)**

1. Erdas > Subset and Chip > thematic
   1. *C:\Users\wb486933\Ten\_Cities\Mappy\_Classification\Nairobi\_new\nairobi\_new\_mosaic\_classified\_smooth5\_thematic.img*

# Accuracy Assessment

You can perform an accuracy assessment using MpGlue or any of the following applications QGIS, ArcGIS, and Erdas Imagine. Essentially in all of these three applications you are performing the same action. You are picking 30-50 random points in each class, and then zooming to each points to see if the classification has correctly identified the pixels in the original image. We normal pick 50 points, but feel free to use more or less points. Below are the instructions for all three applications.

Regarding MpGlue, to do an accuracy assessment, use sample-raster with the assessment points and thematic map, and just add --accuracy to the command line.

**Accuracy Assessment (QGIS)**

1. Convert raster to polygon: Raster > Conversion > Polygonize
   1. Input: classification
   2. Output: Conakry\_2013\_rastertopoly.shp
2. Clip the new shapefile by the AOI file: Vector > Geoprocessing > Clip
   1. Output: Conakry\_2013\_rastertopoly\_clip.shp
3. Dissolve by Gridcode: Vector > Geoprocessing > Dissolve
   1. Input: Conakry\_2013\_rastertopoly\_clip.shp
   2. Output: Conakry\_2013\_rastertopoly\_clip\_dissolve.shp
4. Select class 1 in attribute table of dissolved shapefile, then create random points
5. Create Random Points: Conakry\_2013\_acc\_points\_class01.shp
   1. Stratified Random (50 points per class, 200 total)
6. Open attribute table of points
   1. Add field > ID\_Class >>>> this is what the classification says it is
      1. Field Calculator > ID\_CLASS = [CID] + 1
   2. Add field > ID\_Ref >>>> this will be what the original satellite imagery says it is
7. Edit ID\_Ref column
8. Open DBF in Excel
   1. Edit table and add in accuracy percent

**Accuracy Assessment (ArcGIS)**

Raster to Polygon: Nairobi\_2008\_rastertopoly.shp

Dissolve by Gridcode: Nairobi\_2008\_rastertopoly\_dissolve.shp

Clip to AOI: Nairobi\_2008\_rastertopoly\_dissolve\_clip.shp

Create Random Points: Nairobi\_2008\_acc\_points200.shp

* 1. 50 per class (200 total)

Open attribute table of points

* 1. Add field > ID\_Class >>>> this is what the classification says it is
     1. Field Calculator > ID\_CLASS = [CID] + 1
     2. The CID class it makes for you is one number off (annoying…) so double check the points are in the right classes
  2. Add field > ID\_Ref >>>> this will be what the original satellite imagery (the analyst) says it is

Edit ID\_Ref column to say whatever class you believe it’s in

Open DBF in Excel

* 1. Filter columns and count how many were correct

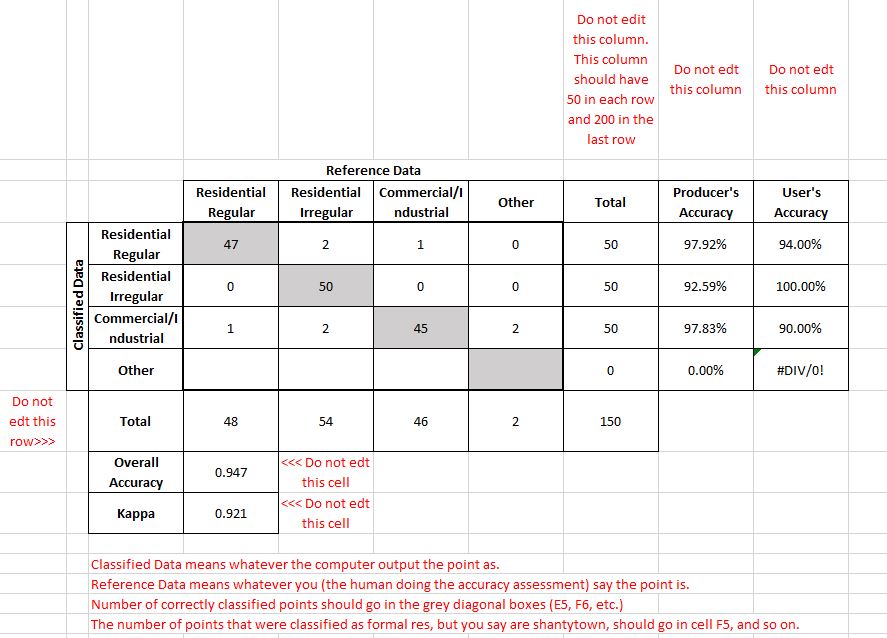
Accuracy\_Assessment\_Table\_Worksheet.xlsx

* 1. Pre-made table that you can just input your numbers into

**Accuracy Assessment (Erdas)**

1. Erdas > Supervised > Accuracy Assessment
   1. Open nairobi\_new\_smooth5\_with-12345678.img
   2. Add Random Points: 200
      1. 50 per class
      2. Select classes 1-4
   3. Save Table
2. Save Acc Report as nairobi\_new\_acc\_report.txt
3. Save table columns as nairobi\_new\_acc\_points.dat

Below you can see a screenshot of the accuracy assessment table:



We will share this excel for you to fill out.