The Remote Sensing and GIS Software Library

Pete Bunting

January 2017
1 Introduction
   • The Software System
   • Course Overview

2 Workbook
   • Getting Started
     • Part 1
     • Part 2
     • Part 3
   • Pre-Processing
   • Classification
     • Pixel Base Classification
     • Object Base Classification

3 Conclusions
Paradigm

- Shifting from small datasets to large.
  - For years scientists have demonstrated techniques on single images / study sites.
  - We now have large spatial and time series datasets available to us - opportunity to demonstrate techniques on larger datasets.
- From single core to multicore
  - How many Universities and institutions have invested in HPC equipment?
  - How can we effectively utilise this capability for Remote Sensing?
- Software licensed per core?
  - Does open source provide a better more flexible solution?
The Remote Sensing and GIS Software Library

- Started by Pete Bunting in April 2008 as a means of grouping together code developed as part of his research and to provide a better platform for development of new functions.
- Released under an open source (GPL) license in November 2009.
- Contains over 300 functions for processing vector and raster data.
- The user interfaces is a Python bindings, which means commands can be combined into scripts.
  - Great for batch processing large datasets.
- Paper published in Computers & Geosciences [Bunting et al., 2014]
RSGISLib is used in combination with a number of other open source software. The main ones, which we’ll be using during this workshop, are:

- GDAL
- The KEA image format
- RIOS
- TuiView
- Scikit-learn

With Python used to join together the different packages and produce processing chains.
Geospatial Data Abstract Library (GDAL)

- Provides a library for reading and writing image and vector formats
  - Supports a very wide variety of formats
- Includes a set of useful tools, such as
  - Translation between file formats (e.g., GeoTIFF to ENVI)
  - Reprojecting data
  - Polygonise / Rasterise
- Widely used in many software packages
  - Good, broad and activity community
- http://www.gdal.org
KEA image file format

- HDF5 based image file format
- GDAL driver
  - Therefore the format can be used in any GDAL compatible software (e.g., ArcMap)
- Support for large raster attribute tables
  - Used for classification
- zlib based compression
  - Small file sizes
  - 10 m SPOT mosaic of New Zealand 5GB per island (Each approx. 65000, 84000 pixels)
- Development funded and supported by Landcare Research, New Zealand.
- [Bunting and Gillingham, 2013]
Raster I/O Simplification (RIOS)

- A flexible Python framework for image processing
- Numpy and Scipy functions are easily accessible
- Very fast and simple development but scalable
- Framework for accessing GDAL raster attribute tables (RATs)
  - Object Oriented classification
- Developed by Sam Gillingham and Neil Flood
  - Queensland Government and Landcare Research, NZ
- [Gillingham and Flood, 2013]
TuiView

- Free and open source image viewer
  - Using GDAL and Python
- Fast and easy to use
- Light weight can be used from a remote server
- Developed by Sam Gillingham, Landcare Research, NZ
Scikit-learn

- Open source machine learning library for Python.
- [Pedregosa et al., 2011]
- RSGISLib uses scikit-learn to solve a number of classification and clustering problems.
- scikit-learn provides tools for:
  - Classification
  - Regression
  - Clustering
  - Dimensionality reduction
  - Model selection
  - Preprocessing
Python

- High level scripting language which is interpreted, interactive and object-oriented.
- Has clear and understandable syntax.
- Many libraries available for a variety of applications (try Googling an application and Python).
- To fully utilise the Python bindings of RSGISLib, and the other tools described here, knowledge of Python syntax is required.
- For this course a basic knowledge of Python will be helpful but is not required.
Getting Help

- http://www.rsgislib.org
- https://spectraldifferences.wordpress.com
- rsgislib-support@googlegroups.com
- https://bitbucket.org/petebunting/rsgislib
Software Installation

- Binaries available through the conda package management system for MacOS and Linux.
- Windows – use Linux virtual machine, for the moment.
The command line

- The software used for this course are accessed from the command line.
- Steeper learning curve then GUI tools but offer more power.
- Possible to script and batch process multiple data sets.
- Easy to translate to High Performance Computing (HPC) environments.
Topics covered

- Getting started with some bands math.
  - Band processing
  - Command line tool
  - Multiple Cores
- Data pre-processing
  - Mosaicking
  - Re-projection
  - Subsetting
  - Masking
  - Filtering
  - Raster-to-Vector / Vector-to-Raster
- Image Classification – using machine learning
  - Pixels
  - Objects/Segments/Clumps
Course structure

Series of worked examples.

- I’ve provided all the scripts – so focus is on running those in the first instance rather than writing them.
- Worksheet provides instructions and order to apply the scripts – also points to documentation.
- Datasets provided but outputs from one stage feed into the next.
Datasets

We are using datasets downloaded as part of the Global Mangrove Watch, in this case for the Sundarbans.

- PALSAR-2 (4 scenes)
- Landsat-8 (Subsetted mosaic of 2 scenes)
- ESRI Shapefiles – training regions for classifier.
  - This were draw specifically for this training course with arbitrary class definitions.
Need to download datasets / scripts

- https://www.dropbox.com/s/h91mss6284e036m/RSGISLibIntroTraining_Datasets.tar.gz

Python scripts are within ‘Scripts’ and images are within ‘Datasets’

Assume scripts are being run from data working directory.
The flow of Python code is dependent on the indentation.

```python
if x == 1:
    x = x + 1
else:
    x = x - 1
```

The following won’t work:

```python
if x == 1:
    x = x + 1
else:
    x = x - 1
```

Be consistent – recommend 4 spaces (and don’t use tabs).
Introduction

- To run RSGISLib we need to create a python script
  - A text file with the file extension .py
- RSGISLib is a series of functions
  - Think of these link GUI dialogs asking for options to run a function.
  - e.g. `imageutils.createImageMosaic(inputList, outImage)`
Create Working Directory

- **Working directory:**

  ```
  cd Desktop
  mkdir RSGISLibTraining
  cd RSGISLibTraining
  mkdir Exercise1
  ```

- Copy the file N22E089_15_MOS_F02DAR.tar.gz into Exercise1
Extract N22E089_15_MOS_F02DAR.tar.gz using the tar command:

```
tar -zxf ./N22E089_15_MOS_F02DAR.tar.gz
```
In this example we will calculate a three band composite for a PALSAR-2 scene. The output image will have three image bands:

- HH Power
- HV Power
- HH/HV
Write Script in Comments

```python
#/usr/bin/env python

# Import python modules

# Calculate Power for HH image

# Calculate Power for HV image

# Calculate HH / HV

# Stack images into a single band

# Calculate overview pyramids and image statistics
# to make visualisation faster.
```
The script file 01_BandMaths.py has this code, copy it into your working directory Exercise1 and run it.

```
python 01_BandMaths.py
```
View Result

tuiview

or

tuiview N22E089_15_sl_F02DAR_powstack.kea
Batch Processing: Set up

```
rm -Rf Exercise1
mkdir Exercise2
```

Copy all PALSAR-2 files into that directory.
Batch Processing: basics

Going to need to add functionality for:

- Extracting tar.gz
- Finding HH and HV values in extracted data
- Apply functionality from first script.
- Clean up just leaving the outputs.
#!/usr/bin/env python

# Import python modules

def createPALSARStack(inputTAR, outputStackImg, tmpDir):
    # Extract tar.gz file

    # Find the HH and HV images.
    # Calculate Power for HH image
    # Calculate Power for HV image
    # Calculate HH / HV

    # Stack images into a single band
    # Calculate overview pyramids and image statistics to make visualisation faster.

    # Clean up so only the stack remains.

createPALSARStack('N22E088_15_MOS_F02DAR.tar.gz', 'N22E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N22E089_15_MOS_F02DAR.tar.gz', 'N22E089_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E088_15_MOS_F02DAR.tar.gz', 'N23E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E089_15_MOS_F02DAR.tar.gz', 'N23E089_15_MOS_F02DAR_Stack.kea', './tmp')
Define as function.

Therefore we can batch process by calling the function multiple times:

```python
createPALSARStack('N22E088_15_MOS_F02DAR.tar.gz',
                    'N22E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N22E089_15_MOS_F02DAR.tar.gz',
                    'N22E089_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E088_15_MOS_F02DAR.tar.gz',
                    'N23E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E089_15_MOS_F02DAR.tar.gz',
                    'N23E089_15_MOS_F02DAR_Stack.kea', './tmp')
```

Look at script 02A_PALSARStack.py
Python has a really useful module called multiprocessing which allows multiple processing cores to be used. The easiest way to use this functionality is by defining a function which takes a single input. The Pool function can take that function and a list of inputs and use multiple cores.

# Old functions
def createPALSARStack(inputTAR, outputStackImg, tmpDir):

# New function interface
def createPALSARStack(params):
    inputTAR = params[0]
    outputStackImg = params[1]
    tmpDir = params[2]
inputParams = []

inputParams.append(['N22E088_15_MOS_F02DAR.tar.gz',
                    'N22E088_15_MOS_F02DAR_Stack.kea', './tmp'])

inputParams.append(['N22E089_15_MOS_F02DAR.tar.gz',
                    'N22E089_15_MOS_F02DAR_Stack.kea', './tmp'])

inputParams.append(['N23E088_15_MOS_F02DAR.tar.gz',
                    'N23E088_15_MOS_F02DAR_Stack.kea', './tmp'])

inputParams.append(['N23E089_15_MOS_F02DAR.tar.gz',
                    'N23E089_15_MOS_F02DAR_Stack.kea', './tmp'])

# find the number of cores available on the system.
numCores = multiprocessing.cpu_count()

with multiprocessing.Pool(numCores) as p:
    p.map(createPALSARStack, inputParams)
Look at and run 02C_PALSARStack.py – copy it into your working directory.
As a command line tool

It is really useful if you can call your code from the terminal as a tool which takes in the inputs. Python makes this very easy using the `argparse` module.
As a command line tool

```python
# Only run this code if it is called from the terminal
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input", required=True,
                         type=str, help="Input tar.gz file")
    parser.add_argument("-o", "--output", required=True,
                         type=str, help="Output stacked file")
    parser.add_argument("-t", "--tmpath", required=True,
                         type=str, help="Temporary path which will be generated and removed during processing."")

    args = parser.parse_args()

    createPALSARStack(args.input, args.output, args.tmpath)
```
As a command line tool

```bash
python 02D_PALSARStack.py -i N22E088_15_MOS_F02DAR.tar.gz \\ 
-o N22E088_Stack.kea -t tmp
```
To tidy up your files and save space only the final stacked KEA files from your Exercise2 directory are needed for the next exercises, the rest can be deleted.
The first thing is to create the directory for your output files Exercise3. Copy the outputted stacked files into that directory.
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
Processing Steps

- **Mosaicking**
  - 03A_MosaicImages.py
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

# List of the files to be mosaicked.
inputList = ['N22E088_15_MOS_F02DAR_stack.kea',
             'N22E089_15_MOS_F02DAR_stack.kea',
             'N23E088_15_MOS_F02DAR_stack.kea',
             'N23E089_15_MOS_F02DAR_stack.kea']

outImage = './Sundarbans_15_MOS_F02DAR.kea'
imageutils.createImageMosaic(inputList, outImage, 0.0, 0.0, 1, 0,
                             'KEA', rsgislib.TYPE_32FLOAT)
imageutils.popImageStats(outImage, usenodataval=True,
                         nodataval=0, calcpyramids=True)
Processing Steps

- Mosaicking
- **Re-projection**
  - 04_Resample2Landsat.py
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
Re-projection

```python
#/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inRefImg = 'Sundarbans_201511_Landsat.kea'
inProcessImg = 'Sundarbans_15_MOS_F02DAR.kea'
outImg = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'

imageutils.resampleImage2Match(inRefImg, inProcessImg, outImg,
                                 'KEA', 'cubic')

imageutils.popImageStats(outImg, usenodataval=True,
                          nodataval=0, calcpyramids=True)
```
Processing Steps

- Mosaicking
- Re-projection
- **Define Valid Extent**
  - 05_ValidImageryMask.py
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
Define Valid Extent

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import rastergis

landsatImg = 'Sundarbans_201511_Landsat.kea'
palsarImg = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'
validMask = 'Sundarbans_validmsk.kea'

imageutils.genValidMask(inimages=[landsatImg,palsarImg],
                        outimage=validMask, format='KEA',
                        nodata=0.0)

rastergis.populateStats(clumps=validMask, addclrtab=True,
                         calcpyramids=True, ignorezero=True)
```
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- **Raster-to-Vector (Vector-to-Raster)**
  - 06A_RasterToVector.py
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
#_usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import vectorutils

inputImg = 'Sundarbans_validmsk.kea'
outShp = 'Sundarbans_validmsk_shp.shp'
vectorutils.polygoniseRaster(inputImg, outShp, imgBandNo=1,
                              maskImg=inputImg, imgMaskBandNo=1)
Vector-to-Raster

```python
#/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import vectorutils

inputVec = 'Sundarbans_validmsk_shp.shp'
inputImage = 'Sundarbans_201511_Landsat.kea'
outImage = 'Sundarbans_ValidMask_Landsat_tmp.kea'
vectorutils.rasterise2Image(inputVec, inputImage, outImage,
                             gdalFormat='KEA', burnVal=1)
```
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- **Spatial Subset**
  - 07_Subset2ROI.py
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
Spatial Subset

```python
#/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inputvector = 'Sundarbans_validmsk_shp.shp'

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub.kea'
imageutils.subset(inputimage, inputvector, outputimage, 'KEA', rsgislib.TYPE_32FLOAT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)

inputimage = 'Sundarbans_201511_Landsat.kea'
outputimage = 'Sundarbans_201511_Landsat_sub.kea'
imageutils.subset(inputimage, inputvector, outputimage, 'KEA', rsgislib.TYPE_16UINT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)
```
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- **Masking**
  - 08_MaskImage.py
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

imagemask = 'Sundarbans_validmsk.kea'

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea'

imageutils.maskImage(inputimage, imagemask, outputimage,
                     'KEA', rsgislib.TYPE_32FLOAT, 0.0, 0.0)

imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=0, calcpyramids=True)

inputimage = 'Sundarbans_201511_Landsat_sub.kea'
outputimage = 'Sundarbans_201511_Landsat_sub_msk.kea'

imageutils.maskImage(inputimage, imagemask, outputimage,
                     'KEA', rsgislib.TYPE_16UINT, 0.0, 0.0)

imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=0, calcpyramids=True)
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- **Band Subset**
  - 09_BandSubset.py
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
Band Subset

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
imageutils$selectImageBands(inputimage, outputimage, 'KEA',
                           rsgislib.TYPE_32FLOAT, [1,2])

imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=0, calcpyramids=True)
```
Processing Steps

- Mosaicking
- Re-projection
- Define ValidExtent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
  - 10_LeeFilter.py
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
```python
#/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import imagefilter

timeimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea'
imagefilter.applyLeeFilter(timeimage, outputimage, 5, 5, 'KEA', rsgislib.TYPE_32FLOAT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)
```
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
  - 11_Convert2dBs.py
- Band Maths (NDVI / WBI)
```python
#/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import imagecalc

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_dB.kea'
imagecalc.imageMath(inputimage, outputimage, 'b1?0?10*log10(b1):999', 'KEA', rsgislib.TYPE_32FLOAT)
imageutils.popImageStats(outputimage, usenodataval=True, nodataval=999, calcpyramids=True)

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee_dB.kea'
imagecalc.imageMath(inputimage, outputimage, 'b1?0?10*log10(b1):999', 'KEA', rsgislib.TYPE_32FLOAT)
imageutils.popImageStats(outputimage, usenodataval=True, nodataval=999, calcpyramids=True)
```
Processing Steps

- Mosaicking
- Re-projection
- Define Valid Extent
- Raster-to-Vector (Vector-to-Raster)
- Spatial Subset
- Masking
- Band Subset
- Filtering
- Image Maths (dBs)
- Band Maths (NDVI / WBI)
  - 12_CalcReflIndices.py
Band Maths (NDVI / WBI)

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imagecalc
from rsgislib import imageutils

inputimage = 'Sundarbans_201511_Landsat_sub_msk.kea'

outputimage = 'Sundarbans_201511_Landsat_sub_msk_ndvi.kea'
bandDefns = [imagecalc.BandDefn('red', inputimage, 4),
             imagecalc.BandDefn('nir', inputimage, 5)]
imagecalc.bandMath(outputimage, 'nir==0?999:(nir-red)/(nir+red)', 'KEA',
                    rsgislib.TYPE_32FLOAT, bandDefns)
imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=999, calcpyramids=True)

outputimage = 'Sundarbans_201511_Landsat_sub_msk_wbi.kea'
bandDefns = [imagecalc.BandDefn('blue', inputimage, 2),
             imagecalc.BandDefn('nir', inputimage, 5)]
imagecalc.bandMath(outputimage, 'nir==0?999:(blue/nir)', 'KEA',
                    rsgislib.TYPE_32FLOAT, bandDefns)
imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=999, calcpyramids=True)
```
Combing together

- 13_CombinedPreProcessing.py demonstrates a script with all these steps combined into a single script.
- This would commonly be how I would undertake my processing.

```python
python 13_CombinedPreProcessing.py
```
Tidy up

Remove all files so you just keep the following files:

- Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea
- Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea
- Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea
- Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_dB.kea
- Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee_dB.kea
- Sundarbans_201511_Landsat_sub_msk.kea
- Sundarbans_201511_Landsat_sub_msk_wbi.kea
- Sundarbans_201511_Landsat_sub_msk_ndvi.kea
- Sundarbans_validmsk.kea
Image classification is the process of assigning regions (either pixels or segments) to a thematic class based on the available information (i.e., backscatter, spectral reflectance etc.).

- Rule Base – i.e., manually defined decision trees.
- Unsupervised – i.e., clustering algorithms such as KMeans and ISOData.
- Statistical Supervised
  - Minimum Distance
  - Paralleled Pipe
  - Mahalanobis Distance
  - Gaussian Maximum Likelihood
- Machine Learning – basically more advanced statistical supervised classifiers.
  - K- nearest neighbour
  - Decision Trees
  - Support Vector Machines
  - Random Forests
  - Neural Networks
Steps for Performing Pixel Base Classification

- Rasterise the vector layers providing the training regions for the classification.
- Sample the pixels within the raster regions for each class (i.e., normalise the number of training samples for each class).
- Define the layers to be used for the classification and for which the training data needs to be extracted.
- Extract the training data from the input images for the sampled training pixels and save as a HDF5 file.
- Define the classification classes with the HDF5 file holding the training data and the colour to be used to visualise the classification.
- Create the scikit-learn classifier – any classifier in the library can be defined with the required parameters.
  - Note. the function classimgutils.findClassifierParametersAndTrain can be used to find the optimal classifier parameters and train the classifier.
- Train the classifier
- Apply the classifier
Now look at the script 14_PixelBasedClass.py:

```
python 14_PixelBasedClass.py
```
Pixel Classification Result

(a) Landsat Image

(b) Pixel based classification

Figure: Pixel based classification result.
Object Classification Steps

- Perform a segmentation.
- Populate the segmentation (clumps file) with the variables you wish to use for the classification. Building the raster attribute table. For this study we will populate with:
  - Mean: HH and HV Power, Landsat Reflectance, NDVI, WBI
- Convert mean HH and HV power values to dBs.
- Define the training data and add it to the RAT.
- Balance the training data so the number of samples are similar/same for each class.
- Define variables to be used for the classification
- Search for the optimal classifier parameters (GridSearch) and train the classifier.
- Define the class output colours
- Applying the classifier
- Collapse the clumps file to just the classification
Shepherd et al., Segmentation

**Segmentation Flowchart**

- 2.1. Seeding (KMeans/ISOData)
- 2.1. Image Sub-sampling
- 2.2. Pixel Labelling and Clumping
- 2.3. Elimination
- 2.4. Relabelling

- Seeded using a KMeans clustering \((k)\).
- Iteratively eliminates (from small to large) segments which are below a size threshold \((n)\).
Segmentation Script

```python
#!/usr/bin/env python

import rsgislib
from rsgislib.segmentation import segutils

landsatImg = 'Sundarbans_201511_Landsat_sub_msk.kea'
palsar2Img = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea'

# Select the image bands to be used for segmentation from the landsat image.
landsatImg564 = 'Sundarbans_201511_Landsat_sub_msk_b564.kea'
rsgislib.imageutils.selectImageBands(landsatImg, landsatImg564, 'KEA', rsgislib.TYPE_16UINT, [5,6,4])

# Stack the selected landsat bands with the PALSAR-2 imagery.
segTmpImg = 'Sundarbans_SegTempImg.kea'
rsgislib.imageutils.stackImageBands([landsatImg564, palsar2Img], None, segTmpImg, None, 0, 'KEA', rsgislib.TYPE_32FLOAT)

# Perform the segmentation
clumpsImg = 'Sundarbans_Clumps.kea'
segutils.runShepherdSegmentation(segTmpImg, clumpsImg, tmpath='./segtmp', numClusters=120, minPxls=50, distThres=100, sampling=100, kmMaxIter=200)
```
Classification Process

Rule based Classification Workflow

- Segmentation
- Populate Attribute Table
- Apply Classifier
- Clumping
What is a raster attribute table?

<table>
<thead>
<tr>
<th>FID</th>
<th>MeanB1</th>
<th>MeanB2</th>
<th>NDVI</th>
<th>Elev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.2</td>
<td>54.3</td>
<td>0.1</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>31.4</td>
<td>56.4</td>
<td>0.25</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>25.7</td>
<td>52.1</td>
<td>0.01</td>
<td>220</td>
</tr>
<tr>
<td>4</td>
<td>18.0</td>
<td>51.3</td>
<td>0.6</td>
<td>330</td>
</tr>
<tr>
<td>5</td>
<td>39.1</td>
<td>58.6</td>
<td>0.466</td>
<td>280</td>
</tr>
<tr>
<td>6</td>
<td>34.5</td>
<td>62.3</td>
<td>0.2</td>
<td>230</td>
</tr>
<tr>
<td>7</td>
<td>24.4</td>
<td>51.3</td>
<td>0.3</td>
<td>150</td>
</tr>
</tbody>
</table>

- MeanB1 and MeanB2 are mean values of two bands in a raster image.
- NDVI is the Normalized Difference Vegetation Index.
- Elev is the elevation value.
Now look at the script 16_PerformObjClass.py:

```python
python 16_PerformObjClass.py
```
Object Classification Result

(a) Landsat Image

(b) Object based classification

Figure: Object based classification result.
Conclusions

- Hopefully you now have an idea about how to use the functionality in RSGISLib
- However, to really learn it you need to apply it to your own problems.
  - That's what we'll do now...
What we haven’t covered

- **Image Calibration** – tools for calibrating imagery, primarily optical data and used within the ARCSI (www.rsgislib.org/arcsi) software for performing atmospheric correction of optical satellite imagery.
- **Elevation** – tools for doing things with DEMs.
- **Image Morphology** – functions for applying image morphology operations
- **Image Registration** – function for applying an automated image-to-image registration
- **Zonal Statistics** – functions for retrieving information into a vector attribute table from a raster image.

